

MPLAB Harmony Help

MPLAB Harmony

Table of Contents

1 Start Here	1-1

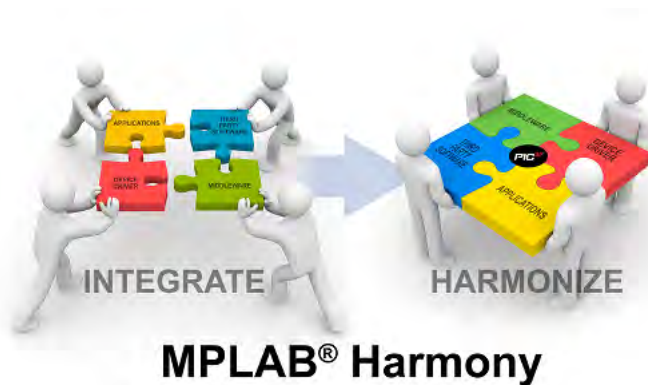
1 Start Here

This topic describes how to get started using the MPLAB Harmony Integrated Software Framework.

1. First Time Users

Name	Description
Before You Start	Before you start using MPLAB Harmony, be sure to install the MPLAB X IDE Plug-In modules.
What is MPLAB Harmony?	This topic provides an overview of key concepts necessary to understand the MPLAB Harmony architecture.
Project Layout	This topic explains how an MPLAB Harmony project is organized.
The Main File	This topic describes the logic of the "main.c" file and the C-language "main" function in an MPLAB Harmony project.
The Application File(s)	This topic describes the normal structure of MPLAB Harmony application files.
More Information	Describes what is available from the MPLAB Harmony web page.

Description



Software Building Blocks

MPLAB Harmony is a framework of software modules that serve as building blocks for your application. If this is the first time you've used MPLAB Harmony, the "First Time Users" section is the best place to start learning. If you're an experienced MPLAB Harmony developer, be sure to look at the "[Release Notes & Information](#)" section to find out what is in this release and what is new.

1.1 1. First Time Users

1.1.1 Before You Start

Before you start using MPLAB Harmony, be sure to install the MPLAB X IDE Plug-In modules.

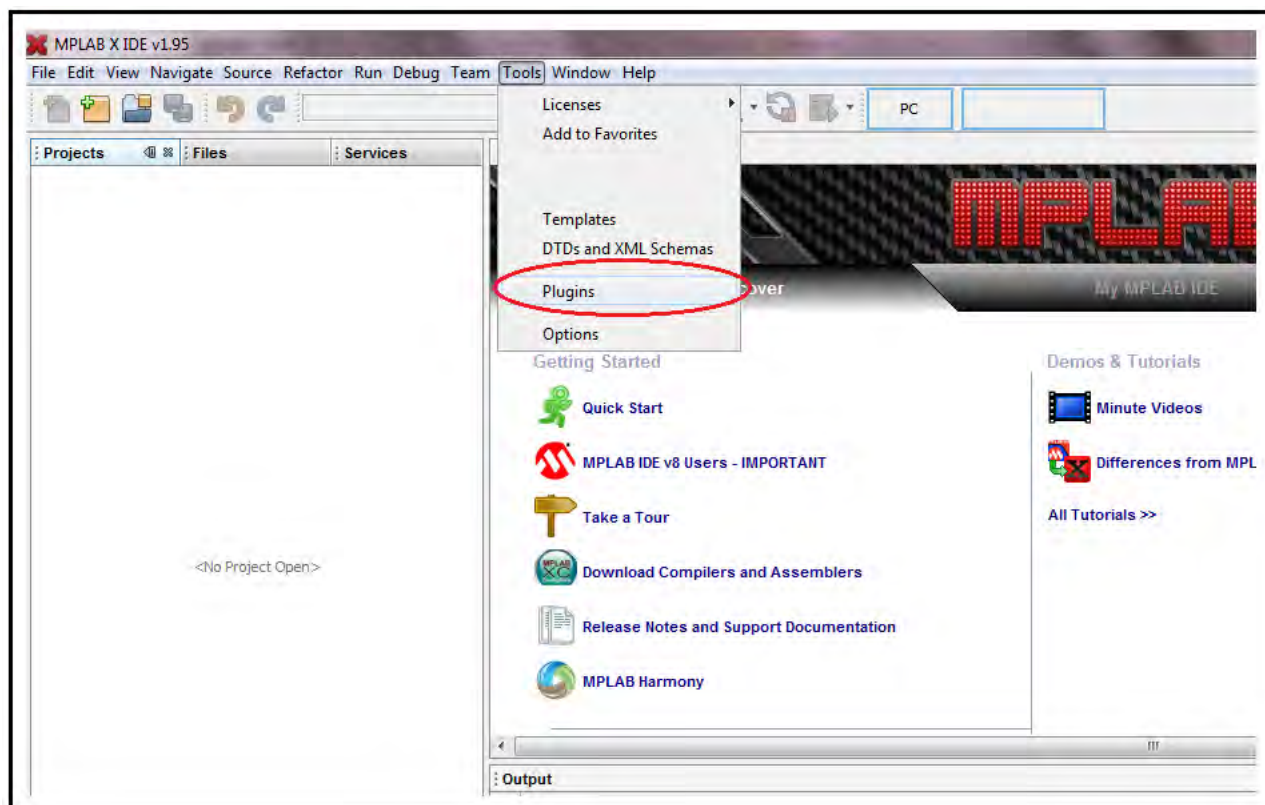
Description

Before you start using MPLAB Harmony, you should install the following MPLAB X IDE plug in modules, included in the MPLAB Harmony installation.

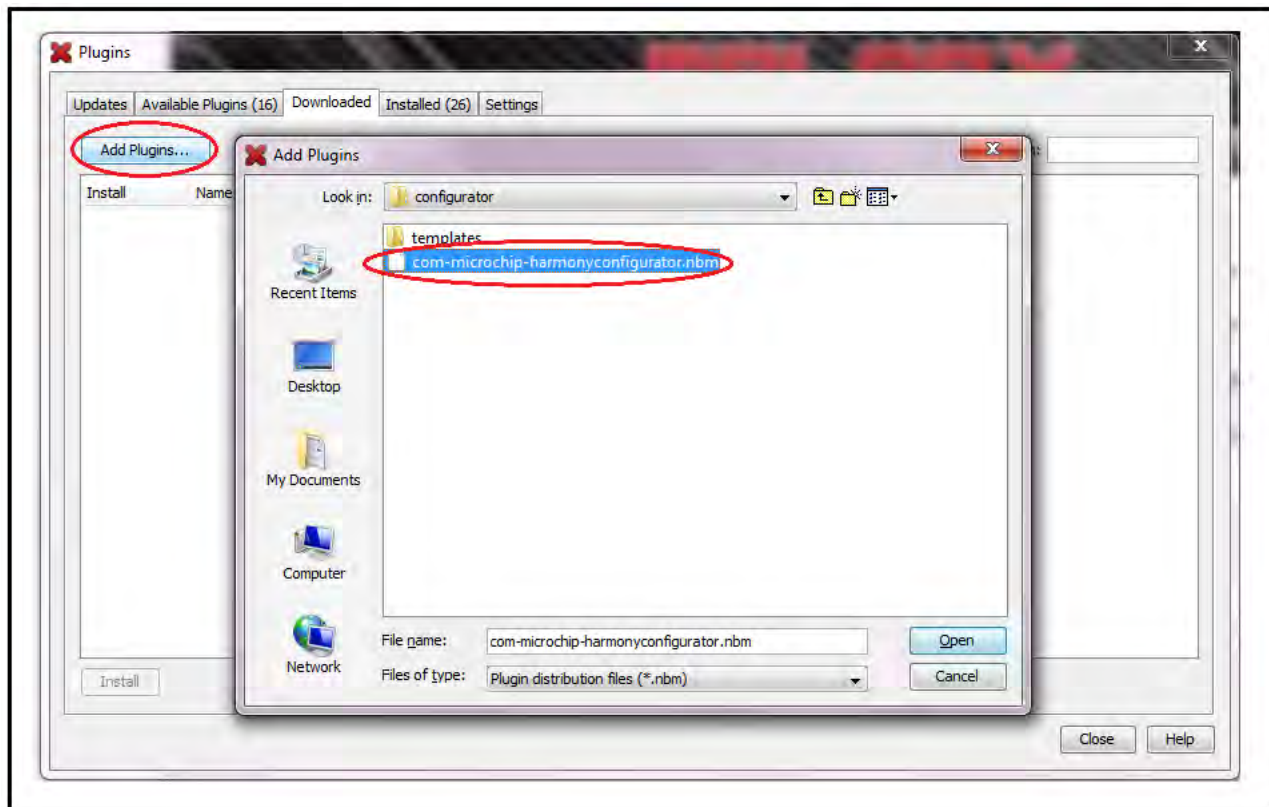
1. The MPLAB Harmony "Help" plug-in module. (Located at: <install-root>/doc/org-microchip-harmony_help.nbm)
2. The MPLAB Harmony Configurator plug-in module. (Located at: <install-root>/utilities/configurator/com-microchip-harmonyconfigurator.nbm)

Installing the Plug In Modules

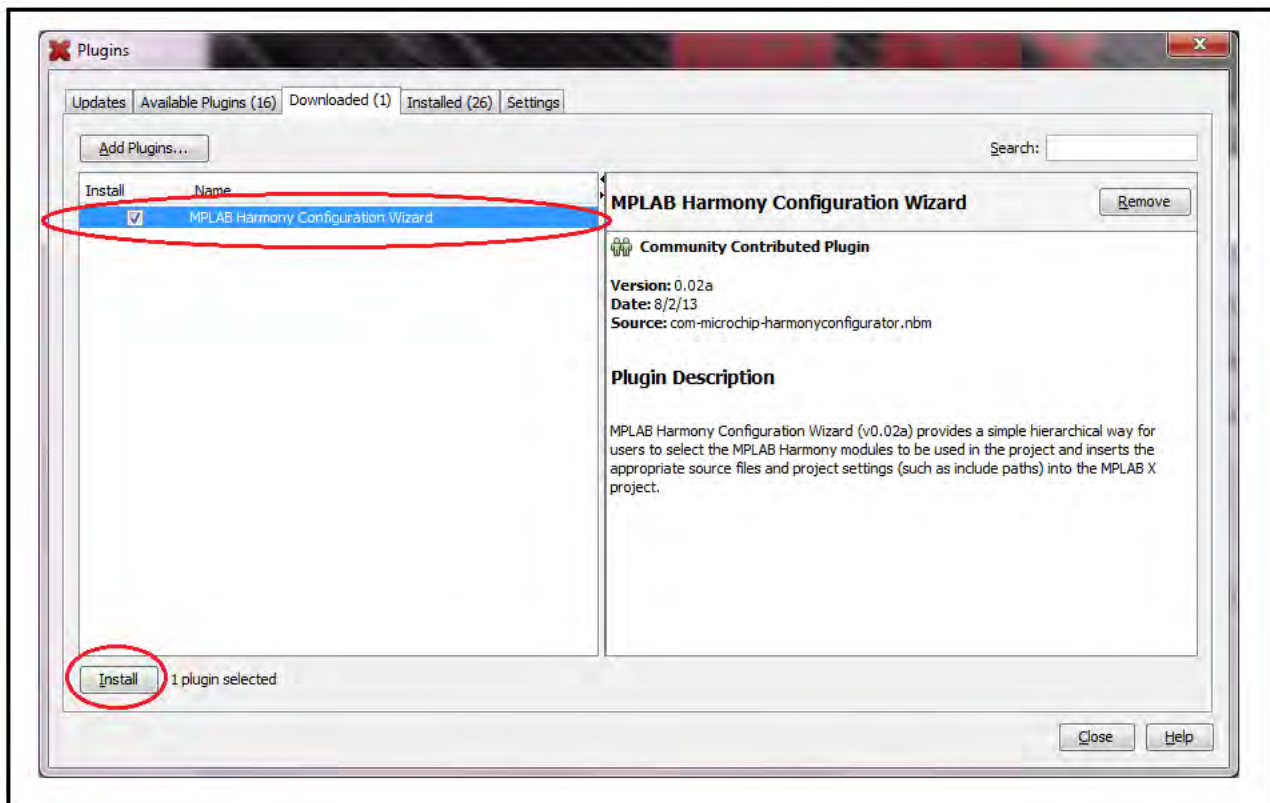
1. Start MPLAB X IDE and select "Plugins" from the "Tools" menu as shown below.



2. Click the "Add Plugins..." button, navigate to and open the "com-microchip-harmonyconfigurator.nbm" plug-in file, as shown below.



3. Ensure that the box under "Install" for the "MPLAB Harmony Configurator" plug-in is selected and click "Install" as shown below.



4. Follow the prompts from the installation and continue until the installation completes. (Do not worry if the version you're installing is "signed" but not "trusted". Just click "continue"). Once the installation has finished you can close the "Plugins" dialog box.

You should now see a "MPLAB Harmony Configurator" option under the "Tools" menu. This indicates that the project Configurator has been successfully installed.

1.1.2 What is MPLAB Harmony?

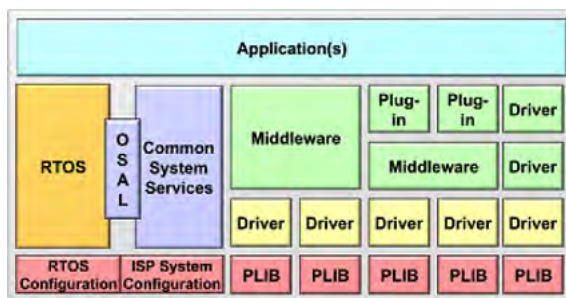
This topic provides an overview of key concepts necessary to understand the MPLAB Harmony architecture.

Description

Introduction

Microchip MPLAB® Harmony is the result of a holistic, aggregate approach to creating firmware solutions for embedded systems using Microchip microcontrollers.

MPLAB® Harmony Block Diagram



MPLAB Harmony Overview

Designed almost completely in the C language (with support for C++), MPLAB Harmony takes key elements of modular and object-oriented design, adds in the flexibility to use a Real-Time Operating System (RTOS) or work without one if you prefer, and provides a framework of software modules that are easy to use, configurable for your specific needs, and that work together in complete harmony.

Portability

Portability is a concern that is often overlooked when a silicon manufacturer provides software. However, breadth of solutions is a hallmark strength of Microchip, and MPLAB Harmony provides simple libraries to abstract away part-specific details and make Microchip microcontrollers easy to use, regardless of which part you choose. Any time you design a new product or update an existing one you have to balance capabilities against cost, but cost is more than just the bill of materials. It's also the Non-Refundable Engineering (NRE) cost to design and develop your solution. MPLAB Harmony provides peripheral libraries, device drivers, and other libraries that use clear and consistent interfaces, requiring little or no change in your application code and minimizing the engineering time and effort for each new design.

Device Drivers

The primary purpose of an MPLAB Harmony device driver (or "driver") is to provide a simple and highly abstracted interface to a peripheral, allowing your application (or other module in the system) to interact with a peripheral through a consistent set of functions. A device driver is responsible for managing access to a peripheral, so that requests from different modules do not conflict with each other, and for managing the state of that peripheral so that it always operates correctly.

Peripheral Libraries

A Peripheral Library (PLIB) is a simple access library that provides a consistent (but very low level) interface to a peripheral that is "on board" the microcontroller. PLIBs hide register details, making it easier to write drivers that support multiple microcontroller families, but they are not normally used by applications directly to interact with peripherals, as they provide little abstraction, and because they require the caller to manage the detailed operation of a peripheral (including preventing conflicting requests from other modules). Because of the lack of conflict protection in a PLIB, only one module in a system should directly access the PLIB for a peripheral. Therefore, PLIBs are primarily used to implement device drivers (and some system services) to make them portable.

Modularity

MPLAB Harmony libraries are modular software "building blocks" that allow you to divide-and-conquer your firmware design. The interface to each library consists of a highly cohesive set of functions (not globally accessible variables or shared registers), so that each module can manage its own resources. If one module needs to use the resources of another module, it calls that module's interface functions to do so. Interfaces between modules are kept simple with minimal inter-dependencies so that modules are loosely coupled to each other. This approach helps to eliminate conflicts between modules and allows them to be more easily used together like building blocks to create the solutions you need.



Middleware Libraries

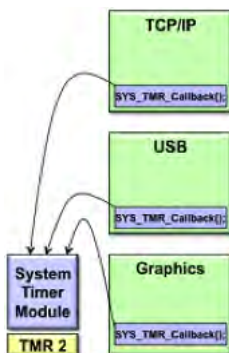
The normal usage models of some of the more complex peripherals, (i.e., USB or network interfaces) requires interpreting complex protocols or may require substantial additional processing to produce useable results, such as drawing graphical images on an LCD screen with an LCD controller peripheral. Therefore, while a device driver may be completely sufficient for a simple peripheral like a UART, some peripherals require what is frequently called “middleware” (aptly named because it sits between your application and the hardware abstraction layer or “driver” layer). MPLAB Harmony provides several middleware library “stacks” to manage these more complex peripherals and provide the functionality you need and expect.

MPLAB Harmony middleware “stacks” are usually built upon device drivers and system services (explained below) so that they can be supported on any Microchip microcontroller for which the required driver or service is supported. However, special purpose implementations may be available that integrate the driver, certain services, and various modules within the “stack” for efficiency.

System Services

MPLAB Harmony system services are responsible for managing shared resources so that other modules (drivers, middleware, and applications) do not conflict on shared resources. For example, if the TCP/IP, USB, and Graphics stacks attempted to concurrently use the Timer2 peripheral to perform some periodic task, they would very likely interfere with each other. However, if instead they used a timer system service (as the following image illustrates), it is the responsibility of the system service to keep the separate requests from interfering with each other. The timer service can be configured as desired for a specific system (for example, you might decide to use Timer3 instead of Timer2) isolating the necessary changes to the configuration of a single module and preventing potential conflicts.

The use of a system service is very similar the use of a device driver, except that a driver normally requires the caller to “open” it to create unique client-to-driver association. A system service does not normally require the caller to open the service before using it because system services are frequently shared by many clients within the system.

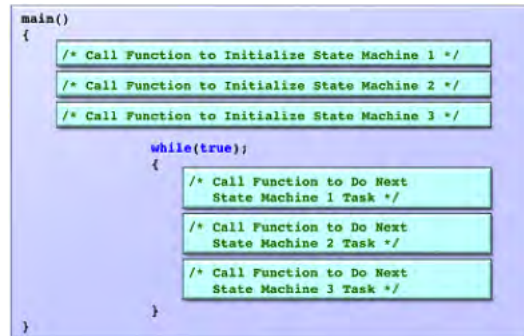


Compatibility

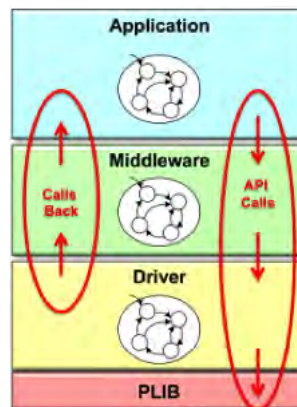
MPLAB Harmony modules (drivers, system services, and middleware – excluding PLIBs) are “active”. This means when an application calls a module’s interface function, the call will usually return immediately and the module will continue working on its own to complete the operation. Most modules will then provide a notification mechanism so the caller (i.e., client) can determine when the operation has finished.

Harmony modules are implemented as cooperative state machines. The following image shows the basic idea of how this works. Each module has an “Initialize” function and each module has one (or more) “Tasks” function(s) to maintain its state machine(s). The state machines of all modules are initialized, shortly after the system comes out of reset in “main”. After that (in a polled

configuration, with no OS), the system drops into a “super loop” where each module’s state machine function is repeatedly called, one after the other, to allow it to do the next “task” necessary to keep its state machine running. This allows the system to keep all modules running using a cooperative or shared “multi-tasking” technique. Modules (under control of your application) interact with each other by calling the interface functions of other modules (as illustrated below) and the system-wide “super loop” keeps all modules in the system running so they stay “active” and do their jobs.

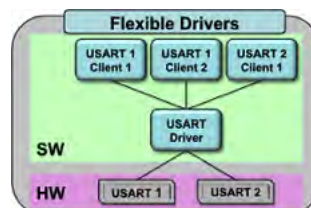


This method is not suitable for all needs; therefore, other configurations are possible. However, a polled configuration is the simplest to understand and it best illustrates the basic concept of how MPLAB Harmony modules work together, as shown below. It is also easy to implement, easy to understand, and easy to debug.



Flexibility

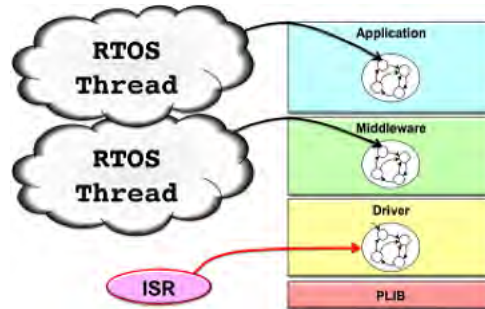
The basic MPLAB Harmony model of cooperating state-machine driven modules, when combined with a little configurability, becomes flexible enough to meet the needs of almost any embedded system. For example, if you're using multiple identical peripherals, MPLAB Harmony has “dynamic” driver implementations that can manage all instances of a peripheral with a single instance of the driver code. You might also have a need for multiple “client” modules to use the same instance of a peripheral at the same time (like the timer example, described previously). To manage this need, MPLAB Harmony has driver implementations that are intelligent enough to manage requests from multiple clients. On the other hand, your needs may be simpler than that. So, static and single client drivers are also available to help reduce the amount of code and data storage needed for your system.



Or, your system may need to combine several middleware stacks and multiple, potentially independent, applications. If that is the

case, the simple polling operation, using the “super loop” method frequently seen in simple embedded systems may not be sufficient. However, when you start adding more modules, it becomes more and more difficult to meet the timing requirements of all peripherals using a simple polled super loop.

Fortunately, MPLAB Harmony modules are written so that (where appropriate) their state machines can be run directly from an Interrupt Service Routine (ISR) or an RTOS thread. Using an ISR allows you to eliminate the latency of waiting for the execution of other modules in the loop to finish before a time-critical event is serviced, and it allows you to use the interrupt prioritization capabilities available on some Microchip devices to ensure that your system responds to events in the real world in real-time.



Additionally, the ability to schedule and prioritize different tasks for different modules can be obtained for modules that are not associated with a specific processor interrupt (such as many middleware modules and your application) using a Real-Time Operating System (RTOS). In fact, that is one of the main reasons to use an RTOS. When your system becomes complex enough that you start struggling to meet your timing requirements using the super loop method, it's time to use an RTOS.

Fortunately, the MPLAB Harmony module state machines functions can be called from a loop in an RTOS thread just as easily as they can be called from a polled “super loop” in a system without an RTOS. To allow this, modules are designed to be “thread safe” by calling semaphore and mutex operations (and a few others, as necessary) through an Operating System Abstraction Layer (OSAL). The OSAL provides a consistent set of functions to call, regardless of which RTOS is being used (or even if no RTOS is used). The choice of RTOS to use, if any, is a configuration option. MPLAB Harmony supports several OS and non-OS configurations and support for more operating systems is possible. All that is required is to implement the OSAL functions appropriately for the desired OS.

Configurability

Most MPLAB Harmony libraries support a variety of build-time configuration options.

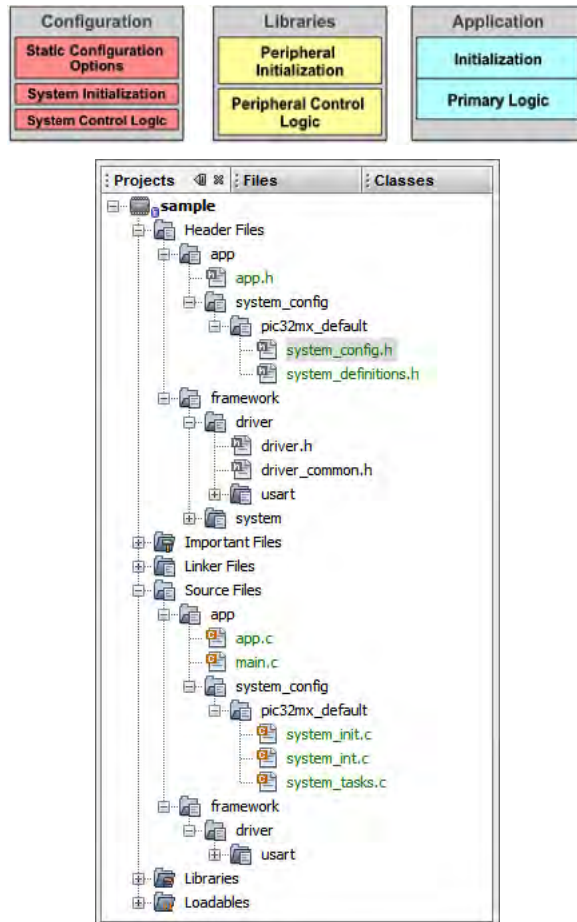
- Selection of supported Microchip microcontroller
- Interrupt-driven or polled execution
- Static-or-dynamic peripheral instance selection (for drivers)
- Single-or-multi-client support (for drivers)
- Other library-specific options

The need for selection of the microcontroller and execution environment has been previously explained. In addition, most drivers and other library modules are designed to allow you to select a variety of configuration options to tailor them to your specific usage. For example, you may be able to select buffer sizes for data transfer modules or clock sources for timer modules. The set of configuration options for each module is identified and explained in the help documentation (along with the interface and usage information) and peripheral initializer (MPLAB Harmony “Pi”) and configuration tools are provided to help simplify the process of configuring your system exactly the way you want and to get you started with a set of initial source files for your project.

Project Structure

To facilitate configurability, MPLAB Harmony projects are normally structured in a way that isolates the code necessary to

configure a “system” from the library modules themselves and from your application code. The following illustrates this concept, while the second figure shows how the files might appear in an MPLAB X IDE project.



In an MPLAB Harmony project, the “main.c” file is kept very simple and consistent (containing primarily, just the super-loop previously discussed). The application files (app.c and app.h in the example to the left) are separate from the configuration files in the “system_config” folder, so it is possible for a single application to have more than one configuration. (Usage of this capability can be seen in example and demonstration projects included with the installation of MPLAB Harmony.) The library modules that make up the MPLAB Harmony framework (in the “framework” folder) use the definitions provided in the selected configuration header (system_config.h, highlighted in the illustration) to specify the configuration options you selected when you configured the project. Finally the processor-specific peripheral libraries (if used) are provided as both a prebuilt binary (“a” linker file) and as in-line source code to allow for maximum build efficiency for your firmware projects.

Conclusion

MPLAB Harmony provides a complete framework for developing your firmware solutions using Microchip microcontrollers and development tools. The firmware libraries and tools that make up the MPLAB Harmony framework are modular and compatible, making them simple to use. They’re flexible and configurable, making them easy to tailor to your specific needs. And, they’re portable across a wide range of Microchip microcontrollers so you’re sure to find a supported part that meets your needs.

1.1.3 Project Layout

This topic explains how an MPLAB Harmony project is organized.

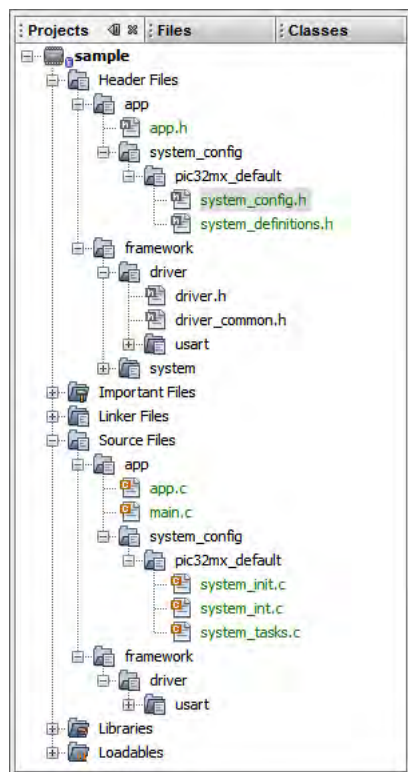
Description

A sample project has been created to show you the structure of an MPLAB Harmony project. The "sample" project is available in the following folder, directly under your MPLAB Harmony installation root folder.

<installation-root>/apps/examples/sample/firmware/sample.X

You should open this project in MPLAB X IDE and follow along with this guide.

An MPLAB Harmony Project is organized as shown below within MPLAB X IDE.



This organization consists of a few key "logical" folders and C-language files, as described below.

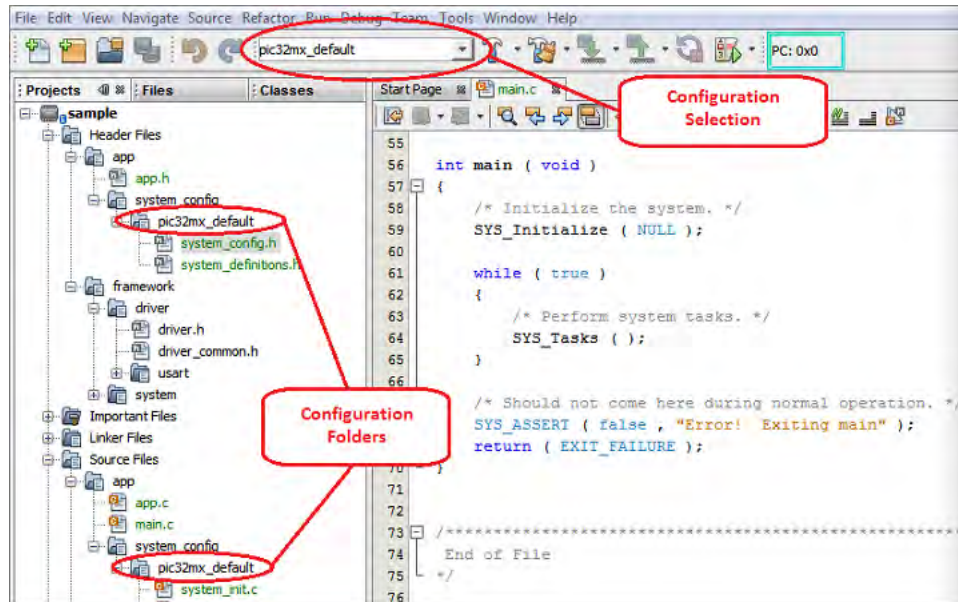
The "app" folder, which contains:

- The "main.c" file
- The "app.c" file

The "app" (short for "application") folder holds all of the application's firmware source files for the project. In a simple project, the "app" folder will directly contain the "main.c" file and the "app.c" file. More complex projects will very likely contain additional files, as needed. In an MPLAB Harmony project, the "main.c" file normally contains the C-language "main" function and little or nothing else. This "main" logic is usually consistent across all MPLAB Harmony projects and should not need to be changed. The "app.c" file normally contains the logic of the application itself. This is where the desired over-all behavior of your system is usually implemented (although complex applications may have additional files).

The "system_config" folder

The "system_config" folder contains one or more subdirectories, each of which corresponds to a configuration of your application or, more accurately, "system". MPLAB Harmony projects may have multiple configuration. In each configuration, you can select a different set of libraries or modules, different build parameters for each module and different source files for your application. A configuration consists of a specific set of properties (tools settings) in MPLAB X IDE and a set of source files that define the build parameters and source code that control which modules are initialized and maintained in your system.



In MPLAB X IDE, the tools configuration can be selected by using a "pull down" menu in the tool bar at the top of the window or by "Right-clicking" on the project name and selecting "Properties". In most example and demo projects distributed with MPLAB Harmony, name of each MPLAB X IDE configuration will match the name of the associated folder under the "system_config" folder in the project (the "pic32mx_default" folder in the sample project). When a specific MPLAB X IDE configuration is selected, the configuration files for that configuration are included in the build and the configuration files in other configuration folders are excluded from the build.

Note: This is project convention used by example and demo projects provided with MPLAB Harmony. You, of course, may organize your own projects any way you wish. But, we recommend following this convention if you use multiple configurations in your projects. We think you'll appreciate the power and flexibility of it.

Configuration Files

- The "system_config.h" file
- The "system_init.c" file
- The "system_tasks.c" file
- The "system_int.c" file (optionally)

A set of three or four files normally make up a complete configuration of the system. The purpose of each of these files is described in more detail in the following sections. But, the basic idea is that you may want different configurations of your application for different physical hardware boards, different Microchip microcontrollers, or different feature sets, depending on your specific needs.

Allowing different configurations of the same application logic makes your application more flexible and provides a well-organized way to deal with the sort of variation that usually occurs in any project of sufficient size and complexity. This technique helps to eliminate the duplication of code (and labor) that would otherwise be necessary to manage multiple related projects. Of course, if you don't need or want that flexibility, all of these files are specifically created for your project and you can make any modifications to them that you like. The choice is always yours.

Important!

The relative path, from the MPLAB X IDE project folder to the configuration folder (containing the "system_config.h" file) for each project configuration must be placed in the "Includes directories" list in the compiler properties for each configuration of the MPLAB X IDE project.

The "framework" folder

The "framework" logical folder contains the source files for the MPLAB Harmony framework and libraries. Depending on your project configuration, there can be many, many files and sub folders under this folder. These files are for MPLAB Harmony libraries that you should not need to edit. In fact, the framework source files are not normally located in your project. Instead, these files are included in your project directly from the MPLAB Harmony installation (using relative directory paths). All of the actual files stay in the MPLAB Harmony installation folder, out of the way.

Notes

1. You always have the option of copying the framework files directly into your project's source folder, if desired. In fact, doing so is a good idea if you plan to move or distribute your project separately from the MPLAB Harmony installation.
2. In most cases, the "logical folder" organization within the MPLAB X IDE project matches exactly with the physical directory organization within the MPLAB Harmony installation (and within your project directory) on your disk drive. This is done to keep things simple and consistent so you only need to learn a single layout. But, there are a couple of notable exceptions.
 - MPLAB has a convention of splitting out "Header Files" (".h" files) and "Source Files" (".c" files), so the virtual folder organization in the MPLAB X IDE project separates the files in to these two groups and the physical directories on disk do not.
 - In an MPLAB Harmony example or demo project, the "app" folder will correspond to the "src" directory on disk under the application's main folder.

1.1.4 The Main File

This topic describes the logic of the "main.c" file and the C-language "main" function in an MPLAB Harmony project.

Description

The C-language entry point for an MPLAB Harmony embedded application is the "main" function. This function is defined in the "main.c" file, contained in the project's "app" folder (or "src" directory on disk). The "main" function (see example, below) implements a simple "super loop", commonly used in embedded applications that do not make use of an operating system.

Example "main" Function Logic

```
int main ( void )
{
    /* Initialize the system. */
    SYS_Initialize ( NULL );

    while ( true )
    {
        /* Perform system tasks. */
        SYS_Tasks ( );
    }

    /* Should not come here during normal operation. */
    SYS_ASSERT ( false , "Error! Exiting main" );
    return ( EXIT_FAILURE );
}
```

The "SYS_Initialize" Function

The first thing the main function does is call a function called "[SYS_Initialize](#)". The purpose of the "[SYS_Initialize](#)" function is to initialize every software module in the system. MPLAB Harmony is based upon a model of cooperating state machines. So, this function must ensure that every module's state machine is placed in a valid initial state. The implementation of this function is one of the things that must be done to configure an MPLAB Harmony system. This function's definition is normally located in the "system_init.c" file and is normally implemented by you since you're the only one who knows which software modules you want to include in your project. However, MPLAB Harmony provide a project Configurator to help you get started.

Note: The "[SYS_Initialize](#)" function signature has a "void *" input parameter. This is so that it may later be implemented in a library and an arbitrary initialization data structure may be passed into it. However, for a statically implemented "[SYS_Initialize](#)" function (which will normally be the case if you implement it yourself), this parameter is unnecessary and can be passed as "NULL". (Note: You could eliminate the parameter, but that may lead to future incompatibilities and the compiler will remove it automatically for you when optimizations are enabled.)

The "Super Loop"

After all of the modules in the system have been initialized, the "main" function executes an infinite loop to keep the system running. This is commonly called a "super loop" as it is the outer-most loop, within which the entire system operates. This loop never exits. So, the code that exists after the end of that loop should never be executed and is only included there for safety, clarity, and syntactical completeness.

The "[SYS_Tasks](#)" Function

Inside of the "super loop", the "main" function calls the "[SYS_Tasks](#)" function. The purpose of the "[SYS_Tasks](#)" function is to poll every module in the system to ensure that it continues to operate. This is how the system maintains the state machine of all polled modules. (Note that some modules may be interrupt driven and thus, not called from the "[SYS_Tasks](#)" function.) The implementation of the "[SYS_Tasks](#)" function is also one of the things that you must normally do as part of the system configuration, again because you are the only one who knows which modules you want to include in your project. You might also have guessed that this function's definition is normally located in the "system_tasks.c" file. Of course, the MPLAB Harmony project Configurator can help with this also.

1.1.5 The Application File(s)

This topic describes the normal structure of MPLAB Harmony application files.

Description

From the point of view of an MPLAB Harmony system, an application consists of two basic functions.

- APP_Initialize()
- APP_Tasks()

The application's initialization function (APP_Initialize) is normally called from the "[SYS_Initialize](#)" function, called from "main" before entering the top-level loop. The application's "tasks" function (APP_Tasks) is normally called from the "[SYS_Tasks](#)" function, called from "main" from inside the top-level loop. This is how the application's state machine is initialized and "polled" so that it can do its job. The "[SYS_Initialize](#)" function is normally implemented in the "system_init.c" file and the "[SYS_Tasks](#)" function is normally implemented in the "sys_tasks.c" file. That is the convention for example and demo projects distributed with MPLAB Harmony and that is the case for the sample project. You may do as you choose in your own projects, but we recommend following this convention as it will make it easier to manage multiple configurations if you need them and it will be consistent with the MPLAB Harmony project Configurator and other tools.

Application Initialization

An application's initialization function places the application's state machine in its initial state and may perform additional initialization if necessary. This function must not block and it should not call the routines of any other modules that may block. If

something needs to be initialized that may take time to complete, that initialization should be done in the application's state machine (i.e. in its "Tasks" function).

Sample Application Initialization Function:

```
void APP_Initialize( void )
{
    /* Copy the message string to a buffer. */
    strncpy(appData.buffer, APP_HELLO_STRING, APP_BUFFER_SIZE);

    /* Prepare a buffer object for transfer using the USART driver. */
    appData.bufferObject.buffer = appData.buffer;
    appData.bufferObject.bufferSize = min(APP_BUFFER_SIZE, strlen(appData.buffer));
    appData.bufferObject.flags = DRV_USART_BUFFER_FLAG_WRITE;

    /* Place the App state machine in it's initial state. */
    appData.state = APP_STATE_INIT;
}
```

The sample project's initialization function initializes an internal buffer to prepare a "Hello World" message for transmission on a USART and places the application's state machine in its initial state by assigning the "APP_STATE_INIT" enumeration value into the "state" member of the data structure that contains all of the data required by the application (appData). This structure is defined globally, but is only ever accessed by the application itself. The application's initialization function is called from the "SYS_Initialize" function (defined in "system_init.c") which is called from "main" after a system reset. Using this technique, the application is initialized (along with the rest of the system) whenever the system comes out of reset.

Application Tasks

The application's state machine breaks up the job that the application must do into several short "tasks" that it can complete quickly, but between which it must wait for some other module to complete some tasks of its own. (In this case the other module is the USART driver.) Once each short task has completed successfully, the application transitions to another state to perform the next short task.

Example Application Tasks Function:

```
void APP_Tasks( void )
{
    /* Status of USART driver */
    DRV_USART_CLIENT_STATUS usartStatus;

    /* Status of buffer submitted to USART */
    DRV_USART_BUFFER_STATUS bufferStatus;

    /* Check the application state*/
    switch ( appData.state )
    {
        /* Keep trying to open the driver until we succeed. */
        case APP_STATE_INIT:
        {
            /* open an instance of USART driver */
            appData.usartHandle = DRV_USART_Open(SYS_USART_DRIVER_INDEX,
            DRV_IO_INTENT_READWRITE);
            if (appData.usartHandle != DRV_HANDLE_INVALID )
            {
                /* Update the state */
                appData.state = APP_STATE_WAIT_FOR_READY;
            }
            break;
        }

        /* Send the message when the driver is ready. */
        case APP_STATE_WAIT_FOR_READY:
        {
```

```

    /* Get the USART driver status */
    usartStatus = DRV_USART_ClientStatus( appData.usartHandle );
    if ( usartStatus == DRV_USART_CLIENT_STATUS_READY )
    {
        /* Submit buffer to USART */
        appData.usartBufferHandle = DRV_USART_BufferAdd(appData.usartHandle,
&appData.bufferObject);
        if ( appData.usartBufferHandle != DRV_HANDLE_INVALID )
        {
            /* Buffer is accepted. Driver will transmit. */
            appData.state = APP_STATE_WAIT_FOR_DONE;
        }
    }
    break;
}

case APP_STATE_WAIT_FOR_DONE:
{
    bufferStatus = DRV_USART_BufferStatus(appData.usartHandle,
appData.usartBufferHandle);
    if ( DRV_USART_BUFFER_COMPLETED == bufferStatus )
    {
        /* Work is done, move to idle state. */
        appData.state = APP_STATE_IDLE;
    }
}

/* Idle state (do nothing) */
case APP_STATE_IDLE:
default:
    break;
}
}

```

Sample Application States

The sample application's "tasks" function breaks the operation of the application down in to the following states using a "switch" statement with the following "cases".

- APP_STATE_INIT
- APP_STATE_WAIT_FOR_READY
- APP_STATE_WAIT_FOR_DONE
- APP_STATE_IDLE

The sample application is placed into the "APP_STATE_INIT" state when by the application's initialization function before the "tasks" function is ever called. So, the first time the "APP_Tasks" function is called, the switch statement executes the code under this case and the first short "task" the sample application attempts to do is open the USART driver to obtain a handle so that it can transfer data over the USART. Notice that the application checks the value of the handle returned from the "[DRV_USART_Open](#)" function to ensure that it is valid before it transitions to the "APP_STATE_WAIT_FOR_READY" state. If the value of the handle returned by the driver's "open" function is invalid (equal to "[DRV_HANDLE_INVALID](#)"), then the application stays in the "APP_STATE_INIT" state and keeps trying to open the USART driver every time it's "tasks" function is called.

Once the application has a valid handle to the USART driver, it executes the code under the "APP_STATE_WAIT_FOR_READY" case the next time its "APP_Tasks" function is called. In this state, the application calls a USART driver status routine ([DRV_USART_ClientStatus](#)) to see if the driver is ready to transfer data. If the driver is ready, the application calls a USART driver data transfer routine ([DRV_USART_BufferAdd](#)) to [send](#) the data buffer it prepared in the "APP_Initiaize" function to the USART. Then, it checks the handle returned by the "[DRV_USART_BufferAdd](#)" routine to see if it is valid. If the buffer handle is valid, it indicates that the USART driver has accepted the buffer and will take responsibility for the

data transfer from that point forward. The application does not have to do anything else to cause the data transfer to occur, but it may want to know when the transfer has completed, so it transitions to the "APP_STATE_WAIT_FOR_DONE" state. However, if the buffer is not accepted by the driver (in which case the handle returned by the "[DRV_USART_BufferAdd](#)" function would be invalid), the application stays in the "APP_STATE_WAIT_FOR_READY" and tries again the next time the "APP_Tasks" function is called by the system (i.e. the "[SYS_Tasks](#)" function).

Once the application is in the "APP_STATE_WAIT_FOR_DONE" state, it calls a buffer status routine ([DRV_USART_BufferStatus](#)) to check and see if the buffer it sent has completed. Note that it passes the handle value returned from the "[DRV_USART_BufferAdd](#)" function (stored in the "usartBufferHandle" member of the global "appData" structure) into the buffer status function to identify the buffer on which it wishes to check status. If the buffer status routine returns "DRV_USART_BUFFER_COMPLETED", then the transfer has completed and the application transitions to the "APP_STATE_IDLE" state where it stays and does nothing any time its "tasks" function is called. It's job is done! A more complex application would go on to some other task or potentially begin the process again. But, this is a simple "Hello World" sample application.

Note:

The application is normally initialized last, after all other modules in the system have been initialized. But, it should never assume that any other module that it must call has completed its initialization when the application is initialized or when its "tasks" function is first called. Instead, it should always check the return value or status from any other module it calls to ensure that the call succeeded before moving on to the next state. Following this rule makes applications more robust and allows them to handle errors more effectively.

1.1.6 System Configurations

1.1.6.1 system_config.h

This topic describes the purpose of system configuration header file.

Description

System Configuration

In MPLAB Harmony, most library modules have a set of build time configuration options that define a variety of parameters (such as buffer sizes, maximum or minimum limits, and default behavior). These configuration options normally have acceptable default values. However, if you want to configure a library for your specific needs, its configuration options can be defined using C-language pre-processor "#define" statements. The set of configuration options supported is described for each library in the "Configuring the Library" section of its help document and most libraries provide a template and example configuration header files in a "config" sub-folder under their "src" folder.

To obtain its build configuration options, every library includes the same common top-level configuration file that must be called "system_config.h" and that must be defined as part of your application or, more specifically, as part of your system configuration. The relative directory path to configuration directory that contains this file must be defined in the build properties of your project so that the compiler can find it in its include file search path.

Example Configuration "system_config.h" Header

```
// *****
// *****
// Section: Application Configuration
// *****
// *****
```

```

/* Define the size of the application's message buffer. */
#define APP_BUFFER_SIZE    15
#define APP_UART_BAUDRATE  9600

/* Define the application's "hello world" message string. */
#define APP_HELLO_STRING   "Hello World\r\n"

// *****
// *****
// Section: System Services Configuration
// *****
// *****

/* Define the index for the driver we'll use. */
#define SYS_USART_DRIVER_INDEX    DRV_USART_INDEX_0

/* Define the hardware (PLIB) index associated with this instance of the driver. */
#define SYS_USART_ID              USART_ID_2

// *****
// *****
// Section: UART Driver Configuration
// *****
// *****
#define DRV_USART_INSTANCES_NUMBER    1
#define DRV_USART_CLIENTS_NUMBER      1
#define DRV_USART_INTERRUPT_MODE      false
#define DRV_USART_XFER_BUFFER_NUMBER  10
#define DRV_USART_INTERRUPT_SOURCE_TX  INT_SOURCE_USART_2_TRANSMIT
#define DRV_USART_CONFIG_BYTE_Q_SIZE_TX    1
#define DRV_USART_CONFIG_BYTE_Q_SIZE_RX    1
#define DRV_USART_CONFIG_BLOCK_DEVICE_MODE

#define DRV_USART_INTERRUPT_MODE      false

#define DRV_USART_INSTANCES_NUMBER    1

#define DRV_USART_CLIENTS_NUMBER      1

#define DRV_USART_INTERRUPT_SOURCE_TX  INT_SOURCE_USART_2_TRANSMIT

#define DRV_USART_INTERRUPT_SOURCE_RX  INT_SOURCE_USART_2_RECEIVE

```

The example above defines configuration options for the application, system services, and USART driver used in the "pic32mx_default" configuration of the sample project..

1.1.6.2 system_init.c

This topic describes the purpose of the system initialization file.

Description

In an MPLAB Harmony project, the "SYS_Initialization" function is called from the "main" function in order to initialize all modules in the system. This function is implemented as part of a system configuration, normally in a file called "system_init.c". This file may also include other necessary global system items that must be implemented in order to initialize a system such as processor configuration bits and system-wide global data structures.

Example "system_init.c" File

```

// *****

```



```

// *****
// Section: Configuration Bits
// *****
// *****

#pragma config FPLL0DIV = DIV_1, FPLLMUL = MUL_20, FPLLIDIV = DIV_2
#pragma config FWDTEN = OFF, FCKSM = CSECME, FPBDIV = DIV_1
#pragma config OSCIOFNC = ON, POSCMOD = XT, FSOSCEN = ON, FNOSC = PRIPLL
#pragma config CP = OFF, BWP = OFF, PWP = OFF
#pragma config ICESEL = ICS_PGx2

// *****
// *****
// Section: Driver Initialization Data
// *****
// *****

static DRV_USART_INIT uartInit =
{
    /* System module initialization */
    .moduleInit = {0} ,

    /* Identifies USART hardware module (PLIB-level) ID */
    .usartID = SYS_USART_ID ,

    /* Operation Modes of the driver */
    .operationMode = DRV_USART_OPERATION_MODE_RS232 ,

    /* Flags for the usart initialization */
    .initFlags = 0 ,

    /* Control the line control configuration */
    .lineControlMode = DRV_USART_LINE_CONTROL_MODE_8NONE1 ,

    /* Baud Rate value to be used, if not using auto baud */
    .brgValue = APP_UART_BAUDRATE ,

    /* Operation mode initialization data */
    .operationModeInit = {{0}},

    /* Handshake Mode */
    .handShakeMode = DRV_USART_HANDSHAKE_MODE_NONE ,

    /* Interrupt Source for TX Interrupt */
    .txInterruptSource = INT_SOURCE_USART_2_TRANSMIT ,

    /* Interrupt Source for RX Interrupt */
    .rxInterruptSource = INT_SOURCE_USART_2_RECEIVE ,

    /* Interrupt Source for Error Interrupt */
    .errorInterruptSource = INT_SOURCE_USART_2_ERROR,

    /* Receive Queue length */
    .rxQueueSize      = 3,

    /* Transmit Queue length */
    .txQueueSize      = 3
};

// *****
// *****
// Section: System Data
// *****
// *****

```

```

/* Structure to hold the sample application's system data. */
SAMPLE_SYSTEM_OBJECTS sysSample;

// *****
// *****
// Section: System Initialization
// *****
// *****

void SYS_Initialize ( void* data )
{
    /* Inialize the system */
    sysSample.usart = DRV_USART_Initialize(SYS_USART_DRIVER_INDEX, (SYS_MODULE_INIT
*)&uartInit );

    /* Initialize the Application */
    APP_Initialize ( );
}

```

In addition to the ["SYS_Initialize"](#) function implementation, the above example "system_init.c" file from the "pic32mx_default" configuration of the "sample" project defines the processor configuration bits, a data structure used to initialize the USART driver, and a global "sysSample" data structure used to hold a global handle to the USART driver object returned by the driver's initialization function.

Notice that the ["SYS_Initialize"](#) function initializes first the driver, then the application. These are the only two active modules in this system.

1.1.6.3 system_tasks.c

This topic describes the purpose of the system tasks file.

Description

Since MPLAB Harmony modules are state machine driven, they each have a "Tasks" function that must be called repeatedly (from the system-wide "super" loop in "main" or from an ISR or OS thread). The "Tasks" functions are all called from the top-level ["SYS_Initialize"](#) function that is normally implemented in a file called "system_tasks.c" that is part of a system configuration.

Example "system_tasks.c" File

```

void SYS_Tasks ( void )
{
    /* Call the "tasks" functions for any modules in the system. */
    DRV_USART_TasksTX(sysSample.usart);

    /* Call the application's tasks routine */
    APP_Tasks ( );
}

```

The "system_tasks.c" file for the "pic32mx_default" configuration of the "sample" project, contains only the implementation of the ["SYS_Tasks"](#) function for that configuration. This function calls the USART driver's transmitter tasks function ["DRV_USART_TasksTX"](#), passing in the driver's object handle returned from the driver's initialization routine, and it calls the application's tasks function "APP_Tasks" to keep both the driver and the application's state machines running.

1.1.6.4 system_int.c

This topic describes the purpose of the system interrupts file.

Description

In an interrupt-driven configuration, any modules (such as drivers or system services) that can be driven from an interrupt must have their interrupt-capable tasks function(s) called from an Interrupt Service Routine (ISR) "vector" function instead of from the "[SYS_Tasks](#)" function. The form of the definition of the ISR vector function is dependent on what type of PIC microcontroller on which the system is running. So, any vector functions required are normally implemented as part of the a specific system configuration in a file normally called "system_int.c"

Example "system_int.c" File

```
void __ISR ( _UART_2_VECTOR ) _InterruptHandler_USART_2_stub ( void )
{
    if ( SYS_INT_SourceStatusGet ( APP_USART_INT_SOURCE_TX ) )
    {
        /* Call the USART driver's "Tasks" routine */
        DRV_USART_TasksTX ( sysSample.usart );
    }
}
```

Notice that, in this example, the ISR function must determine if the interrupt occurred because of a transmitter (TX) interrupt because on the PIC32 part used by this configuration of the application, this interrupt vector is shared with other types of interrupts for this USART. Once it has determined that the interrupt is indeed a transmitter interrupt, the ISR calls the USART driver's TX tasks routine and passed in the USART object handle returned from the driver's initialization function.

Note: The USART driver's "[DRV_USART_TasksTX](#)" function is interrupt capable when the USART driver is built with the correct configuration options.

1.1.7 More Information

Describes what is available from the MPLAB Harmony web page.

Description

Information on MPLAB Harmony is available from microchip.com/harmony.

From this landing page, information on additional downloads, links to a MPLAB Harmony discussion forum, among others is available.

1.2 2. Creating Your Own Applications

This section describes the steps necessary to create your own MPLAB Harmony applications.

Description

To support the framework, the application (or MPLAB Harmony "system") is responsible for the following:

- 1. Create a New Project:** This can be accomplished by either running the MPLAB X IDE new project wizard or copying and using a basic template located in the <installation-root>/apps/examples/templates directory. If you copy the template to a new location, you may be required to modify your include paths. To use the project wizard, select *Microchip Embedded > Standalone Project*. Next, choose the device. The device you choose will determine what peripheral libraries are included in your MPLAB Harmony project. Continue in the new project Wizard, selecting your hardware and software tools. When you name and locate your project, there are two choices: 1) You can locate it in the 'apps' folder of the MPLAB Harmony installation or 2) You can locate it outside of the Harmony installation. Placing the project in the 'apps' folder of MPLAB Harmony will require you to copy it if you install a new version of MPLAB Harmony. Installing your project outside of the MPLAB Harmony installation will require you to modify your include paths when a new version of MPLAB Harmony is installed. It is recommended to use relative paths when installing in the 'apps' folder in MPLAB Harmony. Make sure that your new project is set to the main (active) project.
- 2. Determine which MPLAB Harmony Modules are needed for your Application:** Modules are divided into four categories; Peripheral Libraries, System Services, Device Drivers and other libraries (commonly called middleware). To determine your needs, read the [MPLAB Harmony Overview](#) and the Introductory sections for the modules of interest. Some modules have dependencies while others do not. Peripheral Libraries have no dependencies. Drivers depend on peripheral libraries. System Services depend on Peripheral Libraries and or Drivers. Your application or its middleware will typically use drivers or system services.
- 3. Add the required Module Source Files to your Project:** This can be accomplished by reading the help for each module to determine what source files you need and manually entering them into your project. Alternatively, the MPLAB Harmony Configurator can be used to select the modules from a list or tree and it will automatically insert the source files into your project and optionally generate a skeletal template for your project. Note that some modules have multiple implementations each optimized for different purposes. Others may implement optional features in multiple files. Be sure to select the source file right for your implementation and be sure to include the pre-built peripheral library binary ".a" file for the processor you selected in your project.
- 4. Configure The Modules:** With the exceptions of the Peripheral Libraries, every MPLAB Harmony module will require some set of build time configuration options implemented using 'c' language macros '#define'. Descriptions of the supported options are available in the "Configuring the Library" section in the help document for every MPLAB Harmony library. These options must be defined in the system_config.h header that is part of your project template generated by the MPLAB Harmony Configurator. Every MPLAB Harmony source file that supports build time configuration options will include this header. As such, this file must be part of every MPLAB Harmony Project.
- 5. Initialize System:** This task involves completing the system_init.c file. First add the necessary configuration bits for your processor. These are defined in the Special Features section of your device's data sheet. The next step is to complete the [SYS_Initialize](#) function. Finish any necessary processor initialization (wait states, cache control, etc.) MPLAB Harmony may provide services to accomplish this in the System Services Module. Initialize any features specific to the board that are not initialized by other modules such as drivers or system services. Next, initialize all library modules selected in step 2 above used by your application. To do this you must first define the initialization data structure for each module. This can be done statically in the system_init.c file and passed in by pointer to the appropriate initialization function for each library. Needed initialization structures can be found in the help section for each module. Add a call to the initialization function for each library module into the [SYS_Initialize](#) function. Each initialization function will return an object handle for the module instance. These should be captured so that they can be passed to the module task routine. Finally, add the call to APP_Initialize function into the [SYS_Initialize](#) function.
- 6. Call System State Machine Tasks:** The system must call the various task state machine functions at appropriate times. This is accomplished by either adding the call into the [SYS_Tasks](#) function for polled modules or adding the call to the interrupt handler for interrupt driven modules. The object handle saved during the initialization process is used as an argument for the various task state machine functions. The [SYS_Tasks](#) function is typically implemented in the system_tasks.c file while the interrupt handlers are defined in the system_int.c file.

7. **Develop Application State Machine:** You are now ready to develop your application. Since Harmony is based on cooperative state machines, your application should also be state machine driven. At minimum, the APP_Initialize function must put your application in its initial state. It may perform other initialization as long as it does not have to wait for other modules. Any initialization that depends on other modules should be implemented as part of the application state machine. Finally, complete your application functionality by implementing the APP_Tasks state machine function.

1.3 3. Release Notes & Information

This section contains release notes for this release and information on version numbers and release types.

1.3.1 Release Notes

Release notes for MPLAB Harmony v0.70b. Target Release Date: November 18, 2013.

Description

Please refer to [Release Contents](#) for a complete list of modules that are provided in this release.

MPLAB Harmony is a framework of system services, device drivers, and other libraries that are built upon a base of portable peripheral libraries to provide flexible, portable, and consistent software “building blocks” which you can use to develop your embedded PIC32 applications.

This is the first fully public release of MPLAB Harmony. Previous releases were internal to Microchip and had very limited external distribution.

A complete list of the contents of this release is provided in the [Release Contents](#) section. You can find these contents under the installation folder you selected in the following sub-folders:

- “apps” – Contains all demo and example applications provide with this installation.
- “bin” – Contains pre-built binary libraries (“.a” files) for the MPLAB Harmony peripheral libraries for each supported processor and for key libraries provided in binary form.
- “bsp” – Contains MPLAB Harmony board support packages provided for selected Microchip demo and development boards.
- “build” – Provides MPLAB X IDE projects that you can use to re-build the libraries provide under the “bin” folder (in case you want to use different build settings).
- “doc” – Contains the MPLAB Harmony help documentation in PDF and MPLAB X IDE “.nbm” plug-in formats.
- “framework” – Contains source code and API headers for all MPLAB Harmony libraries that are provided in source form.
- “third_party” – Contains MPLAB Harmony compatible libraries and Real-Time Operating Systems (RTOS) provided with this installation.
- “utilities” – Contains design and development utilities and MPLAB X IDE plug-in utilities provided with MPLAB Harmony.

Please review the following key help topics (and their subtopics) to gain an initial understanding of what MPLAB Harmony is and how to use it.

- Start Here > First Time Users
- [Start Here > Creating your Own Applications](#)
- [MPLAB Harmony Configurator Help](#)
- The “Introduction” sections of the individual help topics for the system services, drivers, and libraries that are of interest to you
- [Applications Help > Applications Overview](#)
- The “Introduction” sections of the individual applications help topics

Once you have a basic understanding of MPLAB Harmony, you can begin using the provided demo and example applications to explore it's capabilities and begin developing your own applications.

Before developing your own applications, be sure to look at the example “template” application provided in the “<install-dir>/apps/examples/templates” folder.

The following sections provide a brief overview of items that are new in this release and known issues at the time of the release. Refer to the “Release Notes” section under the help topic for each individual library, application, or utility for additional details.

Refer to the “Using the Library” section for each individual library for a description of how to use library functions to accomplish the tasks that the library was intended to support.

Refer to the “Library Interface” section for each individual library for a programmer’s reference or dictionary of functions and data types defined by that library’s interface.

New This Release:

- Significant SD Card Driver improvements.
- The SPI driver has had configuration and interoperability improvements.
- The NVM driver now handles sector erasing during writes.
- USB Driver demonstrations
- TCP/IP Stack updates (see the TCP/IP Stack Library Help)
- Graphics Library API has been updated
- File System Service Library has been updated
- The following System Services have been added: Clock and Device Control
- USART Driver has been updated
- USB Driver has been updated
- Peripheral Libraries now support PIC32MZ devices
- FreeRTOS now supports PIC32MZ devices
- CyaSSL Library has been added

Known Issues:

- The SD Card driver has had limited testing and is missing a few desired features
- The SPI driver does not support slave mode or DMA
- The NVM driver has only been tested in it’s current memory configuration
- NVM Driver demonstration has limited testing
- SPI Driver demonstration has limited testing and hardware support
- USART Driver demonstration does not support PIC32MZ devices
- USB demonstrations (see the related [Release Notes](#))
- USB Host Stack has limited testing and functional limitations (see the related [Release Notes](#))
- USB Device Stack has limited testing and functional limitations (see the related [Release Notes](#))
- No multi-display support for the Graphics Primitive Library
- File System Service Library does not support MIP16 mode, has limited testing and functional limitations (see the related [Release Notes](#))
- USART Driver has flow control and parity mode limitations and no multi-client support
- MRF24W Wi-Fi Driver Library has limited demonstration support and testing (see the related [Release Notes](#))

1.3.2 Release Contents

This topic lists the contents of this release and identifies the individual version numbers of each module.

Description

This table lists the contents of this release, including a brief description of each module.

Cryptographic Applications: <install-root>/apps/crypto/		
Crypto Demonstration	Crypto Library Demonstration	1.00
Example Applications: <install-root>/apps/examples/		
peripheral/blocking/pic32mz	Simple blocking PLIB examples.	1.00
peripheral/blocking/pic32mx	Simple blocking PLIB examples	1.00
state_driven	Simple MPLAB Harmony compliant state-driven Peripheral Library examples	1.00
sample	Sample "Hello World" demonstration used in "Getting Started" guide in Help system	N/A
DEMOS: Driver Application demos: <install-root>/apps/driver/		
nvm demo	NVM demonstration application	1.00
usart demo	USART demonstration application	1.00
spi demo	SPI demonstration application	1.00
DEMOS : RTOS Application demos: <install-root>/apps/rtos/		
freertos	FreeRTOS demonstration application	1.00
openrtos	OpenRTOS demonstration application	1.00
File System Applications: <install-root>/apps/fs/		
nvm_fat_single_disk	Single-disk Non-Volatile Memory FAT FS Demonstration	1.00
nvm_sdcard_fat_multi_disk	Multi-disk Non-Volatile Memory FAT FS Demonstration	1.00
sdcard_fat_single_disk	SD Card FAT Disk demonstration	1.00
nvm_mpfs_single_disk	Single-disk Non-Volatile Memory	1.00
nvm_sdcard_fat_mpfs_multi_disk	Multi-disk Non-Volatile Memory FAT MPFS Demostration	1.00
DEMOS: Graphics Applications: <install-root>/apps/gfx/		
lcc	Low-Cost Controllerless Graphics Demonstration	4.00b
object	Graphics Object Layer Demonstration	4.00b
primitive	Graphics Primitives Layer Demonstration	4.00b
s1d13517	Epson S1D13517 LCD Controller Demonstration	4.00b
ssd1926	Solomon Systech SSD1926 Controller Demonstration	4.00b

DEMOS: TCP/IP Applications & Utilities: <install-root>/apps/tcpip/		
tcpip_web_server	TCP/IP Web Server Demonstration	1.01
bsd_tcp_server	TCP/IP BSD Server Demonstration	1.00
bsd_tcp_client	TCP/IP BSD Client Demonstration	1.00
bsd_udp_server	TCP/IP BSD Server Demonstration	1.00
bsd_udp_client	TCP/IP BSD Client Demonstration	1.00
tcpip_tcp_server	TCP/IP TCP Server Demonstration	1.00
tcpip_tcp_client	TCP/IP TCP Client Demonstration	1.00
tcpip_udp_server	TCP/IP UDP Server Demonstration	1.00
tcpip_udp_client	TCP/IP UDP Client Demonstration	1.00
wolf_ssl	wolf ssl Demonstration	1.00
wifi_console	TCP/IP Wi-Fi Console Demonstration	
wifi_easyconf	TCP/IP Wi-Fi Easy Configuration Demonstration	
DEMOS: USB Device (Function) Applications: <install-root>/apps/usb/device/		
audio_speaker	USB Audio Device Speaker Demonstration	1.00
cdc_com_port_dual	CDC Dual Serial COM Ports Emulation Demonstration	1.00
cdc_com_port_single	CDC Single Serial COM Port Emulation Demonstration	1.00
generic_device	Microchip Generic USB Device Demonstration	1.00
hid_basic	Basic USB Human Interface Device (HID) Demonstration	1.00
hid_joystick	USB HID-Class Joystick Device Demonstration	1.00
hid_keyboard	USB HID-Class Keyboard Device Demonstration	1.00
hid_mouse	USB HID-Class Mouse Device Demonstration	1.00
msd_basic	USB Mass Storage Device Demonstration	1.00
cdc_serial_emulator	CDC Serial Emulation Demonstration	1.00
DEMOS: USB Host Applications: <install-root>/apps/usb/host/		
msd_simple_thumb_drive	USB MSD Host Simple Thumb Drive Demonstration	1.00
cdc_basic	USB CDC Basic Demonstration	1.00
cdc_serial	USB CDC Serial Demonstration	1.00
hid_keyboard	USB HID-Class Joystick Host Demonstration	1.00
hid_mouse	USB HID-Class Mouse Host Demonstration	1.00
generic_device	Microchip Generic USB Host Demonstration	1.00
Pre-built Binaries: <install-root>/bin/		
framework/peripheral	Pre-built Peripheral Libraries	1.00
framework/math/dsp	Pre-built DSP libraries for PIC32MZ devices	1.00
framework/math/libq	Pre-built fixed-point math libraries for PIC32MZ devices	1.00

Board Support Packages (BSPs): <install-root>/bsp/		
explorer16/pic32mx795f512l	Explorer 16 Development Board and PIC32MX795F512L PIM	1.00
sk_pic32_mx_usb	PIC32 USB Starter Kit	1.00
sk_pic32_mz	PIC32MZ Embedded Connectivity (EC) Starter Kit	1.00
usb_audio_pic32mx	PIC32 USB Starter Kit II	1.00
gfx	Graphics (GFX) boards	1.00
Documentation: <install-root>/doc		
doc/org-microchip-harmony_help.nbm	MPLAB Harmony Help MPLAB X IDE Plug-In	0.70b
doc/harmony_help.pdf	MPLAB Harmony Help	0.70b
Middleware & Libraries: <install-root>/framework/		
Crypto (Software)	Microchip Cryptographic Libraries	1.00
GFX	Graphics Library	4.00b
MATH/DSP	API header for PIC32MZ DSP Libraries	1.00
MATH/LIBQ	API header for PIC32MX fixed-point libraries	1.00
AAC/MP3	Audio Decoder Libraries	1.00
TCP/IP	TCP/IP Network Stack	7.10
USB Host	USB Host Stack	0.02b
USB Device	USB Device Stack	0.05b
Device Drivers: <install-root>/framework/driver/		
adc	Analog-to-Digital Converter Driver	0.01b
ethmac	Ethernet Media Access Controller Driver	7.10
ethphy	Ethernet Physical Interface Driver	7.10
gfx/gfx_lcc	Low-Cost Controllerless Graphics Driver	4.00b
gfx/gfx_otm2201a	OTM LCD Controller Driver	4.00b
gfx/gfx_s1d13517	Epson S1D13517 LCD Controller Driver	4.00b
gfx/gfx_ssd1926	Solomon Systech SSD1926 Controller Driver	4.00b
nvm	Non-Volatile Memory Driver	0.50b
pmp	Parallel Master Port Driver	0.50b
sdcard	SD Card Driver (Client of SPI Driver)	0.50b
spi	Serial Peripheral Interface (SPI) Driver	0.50b
tmr	Timer Driver	0.50b
usart	Universal Synchronous/Asynchronous Receiver/Transmitter (USART) Driver	0.50b
usb/usbhs	Universal Serial Bus (USB) Controller Driver	0.05b
wifi/mrf24w	Wi-Fi MAC driver for the MRF24W controller	3.0.0

Operating System Abstraction Layer (OSAL): <install-root>/framework/osal		
osal.h	OSAL Interface	0.52b
osal_freertos.h osal_freertos.c	OSAL Implementation for FreeRTOS	0.52b
osal_openrtos.h osal_openrtos.c	OSAL Implementation for OpenRTOS	0.52b
osal_impl_basic.h	OSAL “bare metal” (no OS) implementation	0.52b
osal_imple_none.h	OSAL implementation that removes the OSAL when not used	0.52b
osal_ucos3.h osal_ucos3.c	OSAL Implementation for Micrium μ COS-III	0.52b
osal_ucos2.h osal_ucos2.c	OSAL Implementation for Micrium μ COS-II	N/A
osal_threadx.h	OSAL Implementation for Express Logic ThreadX	N/A
osal_threadx.c		N/A
Peripheral Libraries: <install-root>/		
framework/peripheral	Peripheral Library Source Code for all Supported PIC32 Processors	0.70b
Build Framework: <install-root>/build/framework/		
crypto/test	Test Vector App for crypto/zlib	1.00
crypto/zlib	zlib library	1.00
crypto/crypto	crypto library	1.00
peripheral	Peripheral Library build project	1.00
math/libq	libq library build project	1.00
math/DSP	DSP library build project	1.00
System Services: <install-root>/framework/system/		
clk	Clock Services Library	0.04a
common	Common Services	0.02a
devcon	Devcon Services Library	0.02b
fs	File System Services Library	0.50b
int	Interrupt Services Library	0.04a
msg	Message Services Library	0.01a
ports	Input/Output Ports Services Library	0.02b
tmr	Software Timer Services Library	0.02a
wdt	Watchdog Timer Services Library	0.02b
Third-Party Software: <install-root>/third_party/		

rtos/FreeRTOSV7.5.2	FreeRTOS Source Distribution v7.5 with support for PIC32MZ devices	Yes (Vendor Version)
rtos/FreeRTOSVX.XX.XX	FreeRTOS Source Distribution v7.5 with support for PIC32MX devices	Yes (Vendor Version)
rtos/OpenRTOSVx.x.x	OpenRTOS Source Distribution vx.x.x with support for PIC32MZ devices	Yes (Vendor Version)
rtos/OpenRTOSVx.x.x	OpenRTOS Source Distribution vx.x.x with support for PIC32MX devices	Yes (Vendor Version)
uC/OS-III	Micrium uC/OS-III Binary Demonstration with support for PIC32MZ devices	Yes (Vendor Version)
uC/OS-III	Micrium uC/OS-III Binary Demonstration with support for PIC32MX devices	Maybe (Vendor Version)
uC/OS-II	Micrium uC/OS-II Binary Demonstration (PIC32MZ)	
uC/OS-II	Micrium uC/OS-II Binary Demonstration (PIC32MX)	
ThreadX	Express Logic (ThreadX) Demonstration with Support for PIC32MZ devices	
CyaSSL Embedded SSL Library	Open Source based demonstration	2.8.0 (Vendor Version)
CyaSSL Embedded SSL Library	Open Source based demonstration	Yes (Vendor Version)
InterNiche Technologies, Inc.	Suitable TCP/IP demonstrations	Yes (Vendor Version)
Utilities: <install-root>/utilities/		
MPLAB Configurator	MPLAB Harmony Project/Library Configurator MPLAB X IDE Plug-In	0.02a
gfx/gdd	Graphics Display Designer	2.00b
gfx/grc	Graphics Resource Compiler	4.00.39
mib2bib/mib2bib.jar	Compiles Custom Microchip MIB script (snmp.mib) to generate snmp.bib and mib.h	2.00
mpfs_generator/mpfs2.jar	TCP/IP MPFS File Generator and Upload Utility	2.02.06
tcpip_discoverer/tcpip_discoverer.jar	TCP/IP Microchip Node Discoverer Utility	2.00

1.3.3 Version Numbers

This section describes the version numbers and their meaning.

Description

MPLAB Harmony Version Numbering Scheme

Libraries released with MPLAB Harmony each have their own version number, using the following scheme.

<major>.<minor>[.<"dot">][<release type>]

- Major revision (architecture or paradigm shift)
- Minor revision (new features, regular releases)
- "Dot" release (bug fixes, unscheduled releases)
- Release Type ('a' = alpha, 'b' = beta, Full release versions 1.0 or above have no type letter)

Items in square brackets are only present when needed (see examples, below).

Examples:

- MPLAB Harmony USB Library v2.6 ("dot" of '0' implied)
- MPLAB Harmony Graphics Library v1.12.02

Note: The version number of the over all MPLAB Harmony release is not related to the version numbers of each individual library or module included in the release and has no bearing on the versions of any libraries it includes. It is merely a way to keep track of what libraries (and what versions of what libraries) are included in each release. Refer to the current release notes to identify what libraries are included and what their version numbers are.

1.3.4 Release Types

This section describes the release types and their meaning.

Description

MPLAB Harmony module releases can be of three different types, based on the version type.



Alpha Release

An alpha release version of a module is usually an initial release. Alpha releases will have complete implementations of their basic feature set, they are unit tested (usually by the original developer) and will build correctly in their basic configuration. An

alpha release is a great "preview" of what a new development Microchip is working on and it can be very helpful for exploring new features. However, it has not gone through a formal test process and it is almost certain that some of its interface will change before the full version is released. It is not recommended for production use.

Beta Release

A beta release version of a module has gone through the internal interface review process and has had formal testing of its functionality. Also, issues reported from the alpha release will have been fixed or documented. When a module is in a beta version, you can expect it to function correctly in normal circumstances and you can expect that its interface is very close to the final form (although changes can still be made if required). However, it has not had stress testing or been tested and it may not fail gracefully if used incorrectly. A beta release is not recommended for production use, but it can be used for development.

1.0+ Release

By the time a module is released as a 1.0 version or above, it is feature complete, fully tested, and its interface is "frozen". All known issues from previous releases will have been fixed or documented. The existing interface will not change in future releases. It may be expanded with additional features and additional interface functions, but existing interface functions will not change. This is stable code with a stable Application Program Interface (API) that you can rely on for production purposes.

1.4 4. Previous Releases

This topic contains information about past version(s) of MPLAB Harmony.

1.4.1 v0.51.03b

Release notes for MPLAB Harmony v0.51.03b.

Description

This topic contains the release notes for MPLAB Harmony v0.51.03b. Please refer to the [Release Contents](#) topic for a complete list modules that are provided in this release.

This release will support MPLAB X IDE v1.95 and XC32 v1.30 tc9 and above.

New this Release

- Updated the Documentation
- Updated Peripheral Pre-Built Binary Libraries (".a" files)
- Added IPv6 API for generation of the Unique Local Address (ULA)

Known Issues

1. No documentation available for the Crypto libraries and peripheral libraries example applications.
2. For TCP/IP stack running on PIC32MZ platforms the write-back mode is not supported yet. Write-through has to be used for now.
3. Early samples of the PIC32MZ silicon do not have a factory pre-programmed MAC address. This will cause the TCP/IP stack and related applications to not properly run in most network configuration. Manual update of the related "network_config.h" file with a valid MAC address is required.
4. The Wi-Fi driver for the TCP/IP stack and the supporting demos are not part of this release.

This table lists the contents of this release, gives a brief description

Module			Description	Version
File	System	Applications:		
<install-root>/apps/fs/				
nvm_fat_single_disk			Single-disk Non-Volatile Memory FAT FS Demonstration	0.02a
nvm_sdcard_fat_multi_disk			Multi-disk Non-Volatile Memory FAT FS Demonstration	0.02a
sdcard_fat_single_disk			SD-Card FAT Disk Demonstration	0.02a
nvm_mpfs_single_disk			Single-disk Non-Volatile Memory MPFS Demonstration	0.02a
nvm_sdcard_fat_mpfs_multi_disk			Multi-disk Non-Volatile Memory FAT MPFS Demonstration	0.02a
Graphics		Applications:		
<install-root>/apps/gfx/				
lcc			Low-Cost Controllerless Graphics Demonstration	0.51b
object			Graphics Object Layer Demonstration	0.51b

primitive	Graphics Primitives Layer Demonstration	0.51b
ssd1926	Solomon Systech SSD1926 Controller Demonstration	0.51b
TCP/IP Applications & Utilities: <install-root>/apps/tcpip/		
tcpip_web_server	TCPIP Web Server Demonstration	7.02a
utilities/mpfs_generator/mpfs2.jar	TCP/IP MPFS File Generator and Upload Utility	2.2.5
utilities/tcpip_discoverer/tcpip_discoverer.jar	TCP/IP Microchip Node Discoverer Utility	2.0
utilities/mib2bib/mib2bib.jar	Compiles Custom Microchip MIB script (snmp.mib) to generate snmp.bib and mib.h	2.0
USB Device (Function) Applications: <install-root>/apps/usb/device/		
audio_speaker	USB Audio Device - Speaker Demonstration	0.51b
cdc_com_port_dual	CDC Dual Serial COM Ports Emulation Demonstration	0.51b
cdc_com_port_single	CDC Single Serial COM Ports Emulation Demonstration	0.51b
generic_device	Generic USB Device Demonstration	0.51b
hid_basic	USB Human Interface Device (HID) Demonstration	0.51b
hid_joystick	USB HID Joystick Device Demonstration	0.51b
hid_keyboard	USB HID Keyboard Device Demonstration	0.51b
hid_mouse	USB HID Mouse Device Demonstration	0.51b
msd_basic	USB Mass Storage Device Demonstration	0.51b
USB Host Applications: <install-root>/apps/usb/host/		
cdc_serial	USB CDC Host Demonstration	0.51b
Crypto Applications: <install-root>/apps/		
crypto	Wolf SSL Cryptographic Demonstration	0.01a
PIC32MZ Applications: <install-root>/apps/		
pic32mz	PIC32MZ Device Demonstration	0.51.01b
Example Applications: <install-root>/apps/examples		
peripheral	Peripheral Demonstration	0.51b
sample	Sample Demonstration	0.51b
Board Support Packages (BSPs): <install-root>/bsp/		
explorer16/pic32mx795f512l	Explorer 16 Development Board and PIC32MX795F512L PIM	
pic32_sk/pic32_usb_sk	PIC32 USB Starter Kit	
pic32_sk/pic32mz_sk	PIC32MZ Starter Kit	
pic32mx_usb_audio	PIC32 USB Digital Audio Accessory Board	
Third Party Software: <install-root>/third_party/		
rtos/FreeRTOSV7.5.0	FreeRTOS Source Distribution v7.5.0	7.5.0
Middleware: <install-root>/framework/		

gfx	Graphics Library	0.51b
tcpip	TCP/IP Network Stack	7.02a
usb	USB Host & Device Stack	0.51b
Crypto Library: <install-root>/framework/		
crypto	Wolf SSL Cryptographic Libraries	0.01a
System Services: <install-root>/framework/system/		
clk	Clock Services Library	0.03a
debug	Debug Services Library	0.01a
fs	File System Services Library	0.02a
int	Interrupt Services Library	0.03a
msg	Message Services Library	0.01a
ports	Input-Output Ports Services Library	0.01a
reset	System Reset Services Library	0.01a
tmr	Software Timer Services Library	0.01a
wdt	Watchdog Timer Services Library	0.01a
Device Drivers: <install-root>/framework/driver/		
adc	Analog-to-Digital Converter Driver	0.01a
ethmac	Ethernet Media Access Controller Driver	0.03a
ethphy	Ethernet Physical Interface Driver	0.03a
gfx/gfx_lcc	Low-Cost Controllerless Graphics Driver	0.51b
gfx/gfx_s1d13517	Epson S1D13517 LCD Controller Driver	0.51b
gfx/gfx_ssd1926	Solomon Systech SSD1926 Controller Driver	0.51b
nvm	Non-Volatile Memory Driver	0.01a
pmp	Parallel Master Port Driver	0.01a
sdcard	SD Card Driver (Client of SPI Driver)	0.01a
spi	Serial Peripheral Interface (SPI) Driver	0.01a
tmr	Timer Driver	0.01a
uart	Universal Synchronous/Asynchronous Receiver/Transmitter (USART) Driver	0.01a
usb	Universal Serial Bus (USB) Controller Driver	0.51b
usbhs	Hi-Speed USB Controller Driver	0.51b
wifi	Wi-Fi Driver	0.01a
Peripheral Libraries (PLIBs): <install-root>/		
framework/peripheral	Peripheral Library Source Code	0.51b
bin/framework/peripheral	Peripheral Pre-Built Binary Libraries for all PIC32 Processors	0.51b
build/framework/peripheral	Project used for Peripheral Library Prebuilt Binary Libraries Creation	0.51b

Operating System Abstraction Layer (OSAL): <install-root>/framework/osal/		
osal.h osal.c	OSAL interface	0.51b
osal_impl_none.h	OSAL implementation that removes the OSAL when not used	0.51b
osal_impl_basic.h	OSAL "bare metal" (no OS) implemenrtation	0.51b
osal_freertos.h osal_freertos.c	OSAL implementation for FreeRTOS	0.51b
osal_ucos.h osal_ucos.c	OSAL implementation for Micrium uCOS-III	0.51b

1.5 5. Tips and Tricks

This topic provides a few helpful tips and tricks to keep in mind while developing your MPLAB Harmony projects.

Description

You may find the following MPLAB Harmony "tips" and "tricks" helpful when developing your own MPLAB Harmony projects.

1. After creating a new MPLAB X IDE project, use the project wizard to add MPLAB Harmony libraries to your project and create a set of initial files for your project. After that, your job is to ensure that the necessary initialization and configuration data and definitions are in place and to write your own application.
2. You can use MPLAB Harmony example and demo projects to put together a "first approximation" of your application.
 1. Copy the source files from the various example or demo apps to your project.
 2. Rename the "APP_Initialize" and "APP_Tasks" functions so that they are all different and will build in your application by simply appending a descriptive name to the end of the existing function names. (Note: you may also need to rename any global data "objects" that conflict with each other from the examples).
 3. Call the individual "APP_Initialize<new_name>" and "APP_Tasks<new_name>" functions from the "[SYS_Initialize](#)" and "[SYS_Tasks](#)" functions in your application's configuration files.
 4. Use the project wizard to add any required drivers, services, or middleware.
 5. At this point, you now have your application made out of several independent state machines. Now, you can begin to develop your own custom application code and modifying any of the code copied from the provided applications to behave as you desire.

1.6 6. Glossary

This section contains a glossary of general MPLAB Harmony terms.

Description

Glossary of Harmony Terms

Term	Definition
Application	One or more application modules define the over all behavior of an MPLAB Harmony system. Applications are either demonstrations or examples provided with the installation or are implemented by you, using MPLAB Harmony libraries to accomplish a desired task.
Client	A client module is any module that uses the services (calls the interface functions) of another module.
Configuration	An MPLAB Harmony configuration consists of static definitions (C-language "#define" statements), executable source code, and other definitions in a set of files that are necessary to create a working MPLAB Harmony system. (See Start Here for additional information.)
Configuration Options	Configuration options are the specific set of "#define" statements that are required by any specific MPLAB Harmony library to specify certain parameters (such as buffer sizes, minimum and maximum values, etc.) build that library. Configuration options are defined in the "system_config.h" system-wide configuration header.
Driver	A "driver" (AKA device driver) is an MPLAB Harmony software module designed to control and provide access to a specific peripheral, either built into the microcontroller or external to it.
Driver Index	Dynamic MPLAB Harmony drivers (and other dynamic modules) can manage the more than one instance of the peripheral (and other resources) that they control. The "driver index" is a static index number (0, 1, 2,...) that identifies which instance of the driver is to be used. Note: The driver index is not necessarily identical to the peripheral index. The association between these two is made when the driver is initialized.
Driver Instance	An instance of a driver (or other module) consists of a complete set of the memory (and other resources) controlled by the driver's code. Selection of which set of resources to control is made using a driver index. Note: Even though there may be multiple instances of the resources managed by a dynamic driver, there is only ever one instance of the actual object code. However, static drivers always maintain a 1:1 relationship between resource and code instances.
Framework	The MPLAB Harmony framework consists of a set of libraries (and the rules and conventions used to create those libraries) that can be used to create MPLAB Harmony systems.
Initialization Overrides	Initialization overrides are configuration options that can be defined to statically override (at build time) parameters that are normally passed into the "Initialize" function of a driver or other MPLAB Harmony module. This mechanism allows you to statically initialize a module, instead of dynamically initializing the module.
Interface	The interface to a module is the set of functions, data types, and other definitions that must be used to interact with that module.
Middleware	The term "middleware" is used to describe any software that fits between the application and the device drivers within an MPLAB Harmony system. It is something of a catch all term that is loosely used to describe libraries that use drivers to access peripheral and then implement communication protocols (such as TCP/IP, USB protocols, and graphics image processing) and other more complex processing that is required to use certain peripherals, but is not properly part of controlling the peripheral itself.
Module	An MPLAB Harmony software module is a closely related group of functions controlling a related set of resources (memory and registers) that can be initialized and maintained by the system. Most MPLAB Harmony modules provide an interface for client interaction. However, "headless" modules with no interface are possible.

Peripheral Index	A peripheral index is a static label (usually an C-language "enum" value) that is used to identify a specific instance of a peripheral. Note: Unlike a driver index, which always starts at '0', a peripheral index may be internally represented as any number, letter, or even a base address and the user should not rely on the value itself - only the label.
Peripheral Instance	An instance of a peripheral is a complete set of the registers (and internal physical resources) necessary to provide the core functionality of a given type of peripheral (either built into the microcontroller or external to it). Note: A specific peripheral instance is identified using a peripheral index.
System	An MPLAB Harmony system is a complete set of libraries, applications, and configuration items loaded and executing on a specific hardware platform (microcontroller, board, and external peripherals) or the source items necessary to build such a system. Note: Since a system can multiple configurations, one MPLAB Harmony project may support multiple systems through multiple supported configurations. See the demo applications included in the installation for examples.
System Service	A system service is an MPLAB Harmony module that provides access to and/or control of common system resources (memory and registers) with which other modules (drivers, middleware, libraries and application) may interact. Note: System services, much like drivers, manage sharing of resources so as to avoid conflicts between modules that would otherwise occur if each module attempted to manage the share resource itself. But, unlike drivers, system services do not normally need to be "opened" in order to use them.

2 MPLAB Harmony Configurator Help

2.1 MPLAB Harmony Configurator

2.1.1 Introduction

This topic provides an overview of the MPLAB Harmony Configurator.

Description

The MPLAB Harmony Configurator is a MPLAB X IDE plug-in. It must be installed into your MPLAB X IDE installation to be used.

2.1.2 Release Notes

MPLAB Harmony Version: v0.70b **Release Date:** 18Nov2013

Description

=====

Version 0.02a

MPLAB Harmony Configurator Plug-in

November 2013

=====

Operating System Support

MPLAB Harmony Configurator Plug-in was tested on the following operating systems:.

- Windows XP SP3 and Windows 7
- Linux OS - Ubuntu
- Mac OS v8

Supported Features:

- Provides a simple hierarchical way for users to select the MPLAB Harmony modules to be used in the project and adds the selected module source files with the necessary configuration and project settings (include path) into the created MPLAB X IDE project.
- Supports importing project related files from relative path.
- Adds predefined source and header template files to the MPLAB X IDE project.
- Library Configuration is supported only for the TCP/IP and USB libraries.

Unsupported Features:

- System-related API calls are not added automatically to the system_init.c, main.c, and system_tasks.c files. Only the standard C templates are added to the project. The application code must be added by the user in the application source and header

files for a successful compilation of the project.

- Library Configuration is not supported for the Graphics library, OSAL, Drivers and Third-Party libraries. However, the corresponding source and header files of the selected modules could be added to the created MPLAB X IDE project using this Configurator.

Notes for Users:

1. The terms "MPLAB Configurator" and "MPLAB Configuration Wizard" are used interchangeably throughout the MPLAB Harmony Help.
2. Check for "TODO" tags that are placed in added template files for directions on what sections in the code should be modified or code to be added by the user. Look for `system_config.h`, `system_init.c`, `system_int.c`, `system_tasks.c`, `app.c`, `app.h`, and `main.c` files.

Known Issues:

- The Configurator Plug-in generated template files are stored in the MPLAB X IDE project folder (inside the `MPLABX_Project.X` folder). Users need to add the current directory `'.'` Path in the MPLAB X Project "properties" > gcc > "include path settings" for removing "the template file not found error".
- The created project may compile but not work, as application and system related API calls are not added automatically to the `system_init.c`, `main.c` and `system_tasks.c`. Only the standard C templates are added to the project.
- The MPLAB Harmony Configurator version during installation or in the plug-in description is shown as 0.0.2 where as the actual version of the tool is "0.02a".
- Certain TCP/IP configuration files are getting modified from the default MPLAB Harmony installation directory if the TCP/IP stack is configured using the Configurator. It is recommended to backup these default configuration files: `tcpip_config.h`, `http_config.h`, `snmp_config.h`, `ssl_config.h`, and `telnet_config.h`, in case it is necessary to revert back to the default configuration.
- TCPIP `system_config.h` file is not modified in this release of the Configurator to add the configurations from `"* <tcpip module> *_config.h"`. As explained above, the specific module files get updated.
- The Configurator is not preserving the previous project configuration if the TCP/IP sub-modules are selected using the 'Select to add default modules' checkbox in the "TCPIP Stack -> tcpip" module tree.
- Warning message about 'template files getting overwritten (if already present)' is not shown if the USB library configuration is selected.

2.1.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

2.2 Using the Configurator

2.2.1 Installing the Configurator Plug-in

The MPLAB Harmony Configurator is a MPLAB X IDE plug-in. It must be installed into your MPLAB X IDE installation to be used.

Description

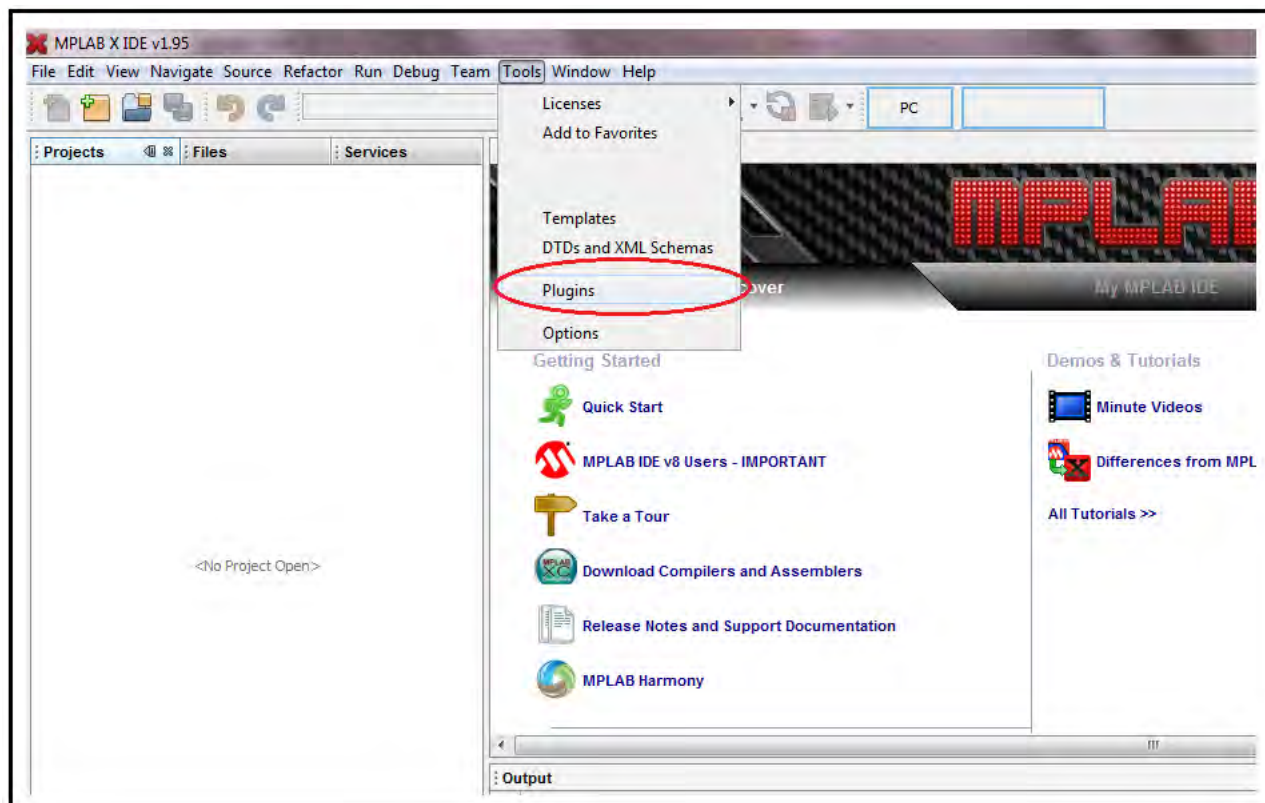
Location

The MPLAB Harmony Configurator (MPLAB X IDE plug-in) is located in the following folder in the MPLAB Harmony installation.

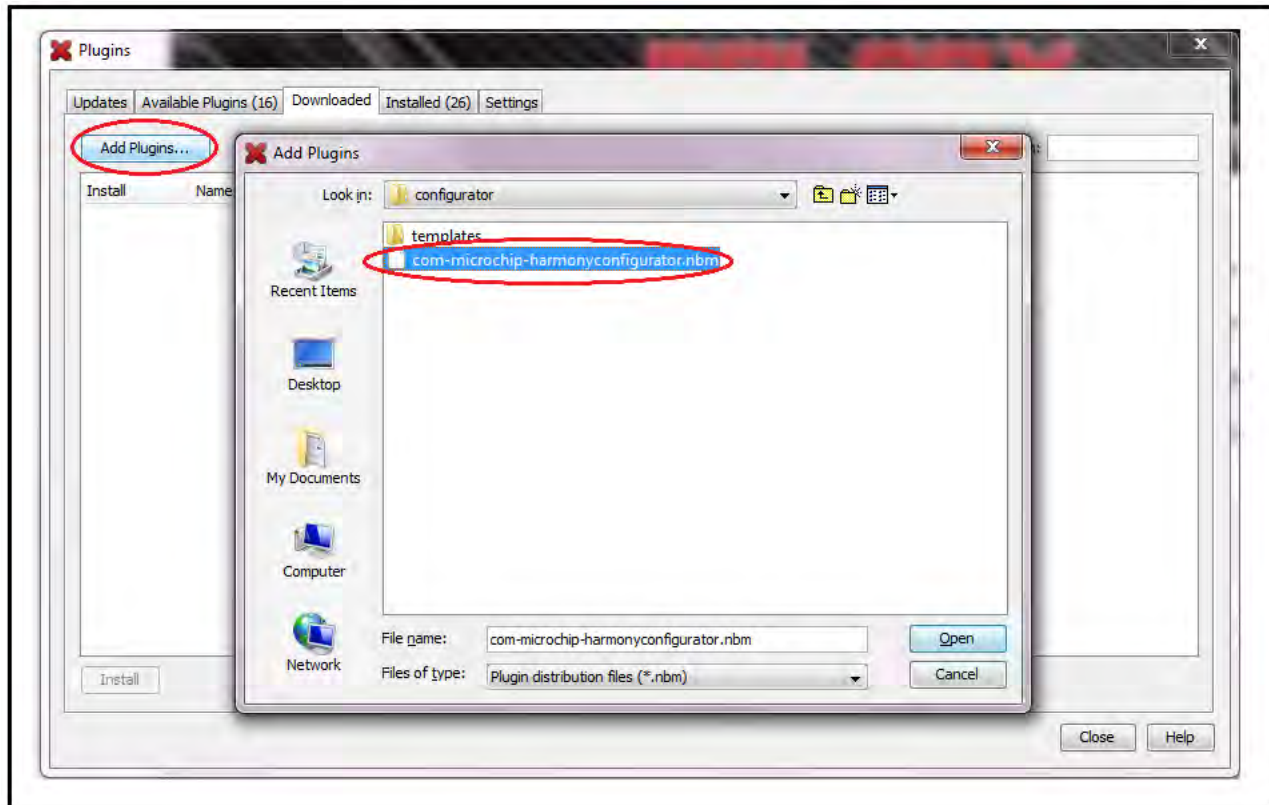
<install_root>\utilities\configurator\com-microchip-harmonyconfigurator.nbm

Installation

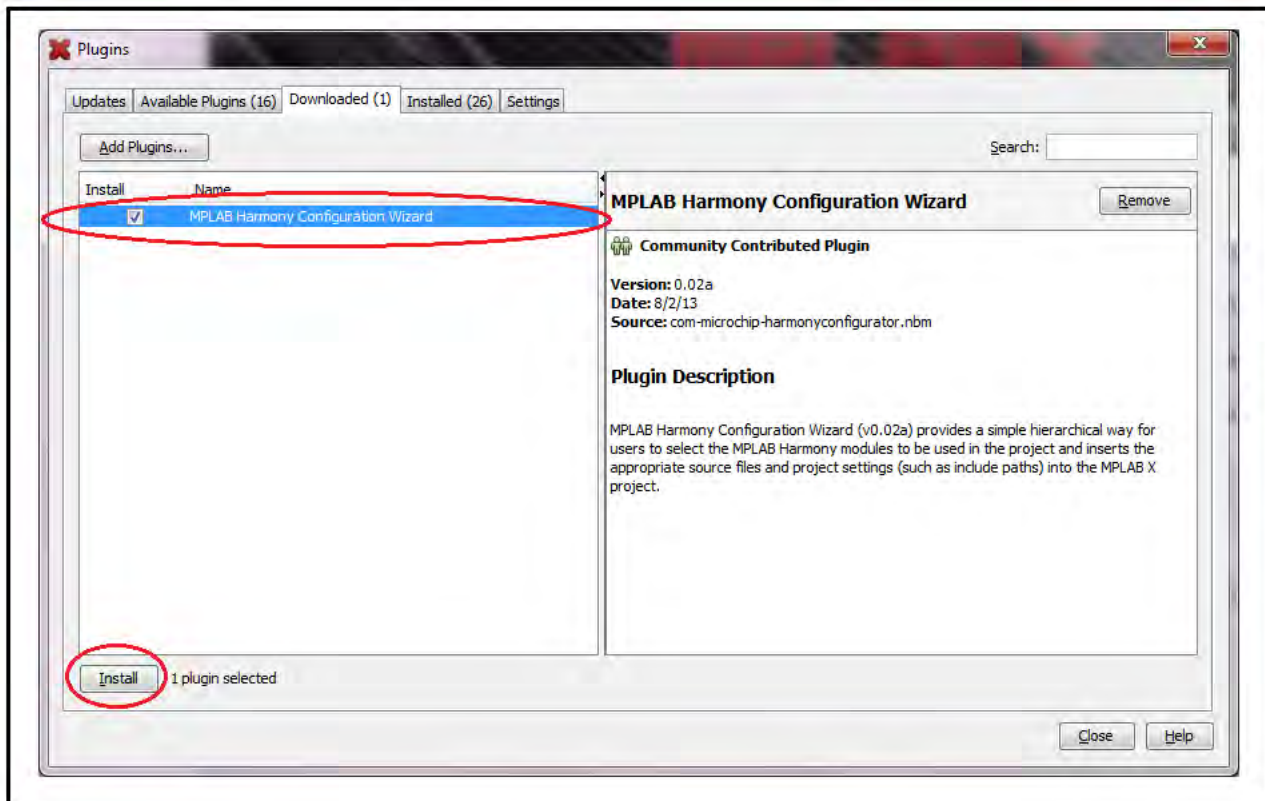
1. Start MPLAB X IDE and select "Plugins" from the "Tools" menu as shown below.



2. Select the "Downloaded" tab.
3. Click the "Add Plugins..." button, browse to <install_root>\utilities\configurator\ and open the plug in file "com-microchip-harmonyconfigurator.nbm" as shown below

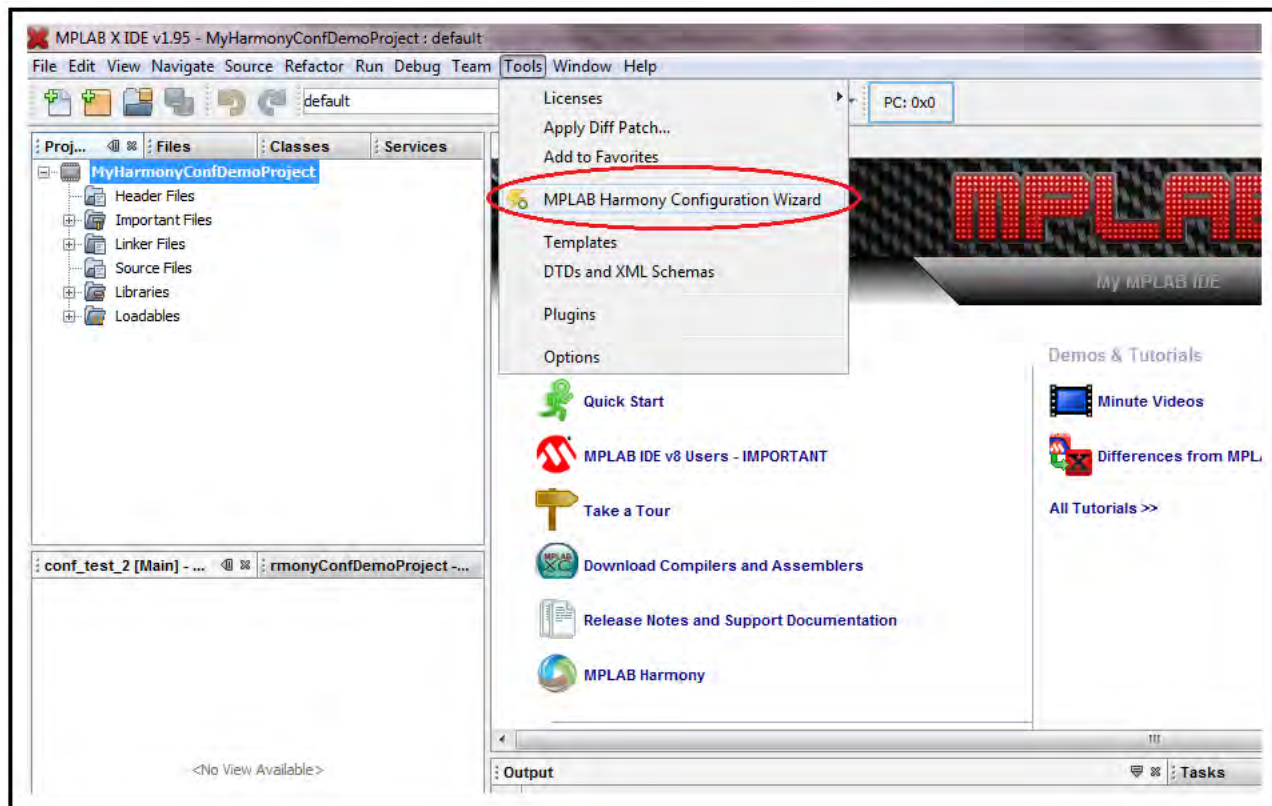


4. Ensure that the Check box for the "MPLAB Harmony Configurator" plug-in is selected. Click "Install" as shown below.



5. Follow the prompts from the installation wizard and continue until the installation completes. Just click "continue". Once the installation wizard has finished you can close the "Plugins" dialog box.

You should now see a "MPLAB Harmony Configurator" option under the "Tools" menu. This indicates that the project wizard was successfully installed.



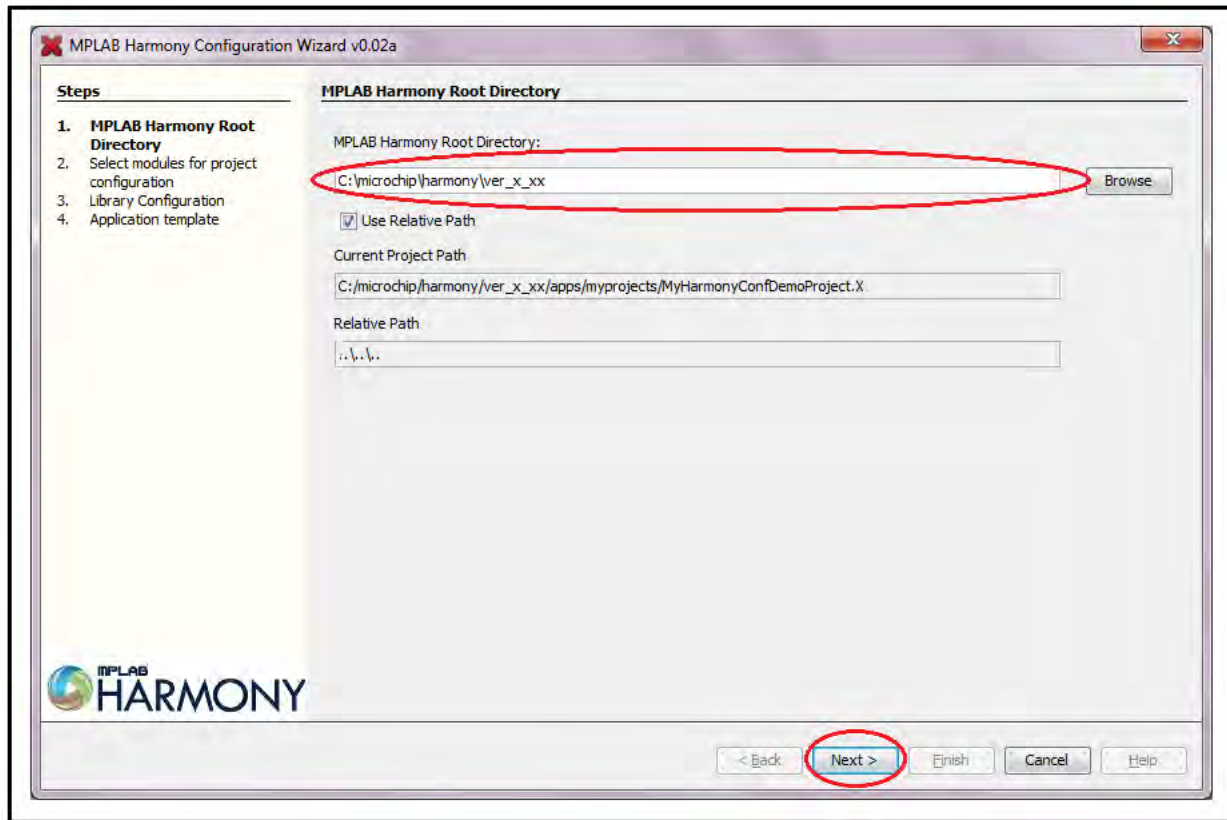
2.2.2 Running the Configurator Plug-in

This session describes how to run the MPLAB Harmony Configurator to add required files from selected modules into MPLAB X IDE project.

Note: This version of the Configurator provides Project Configuration (to add files into MPLAB X IDE project) and limited Library Configuration (only for configuring a few sub-modules of the TCP/IP and USB middleware) capability.

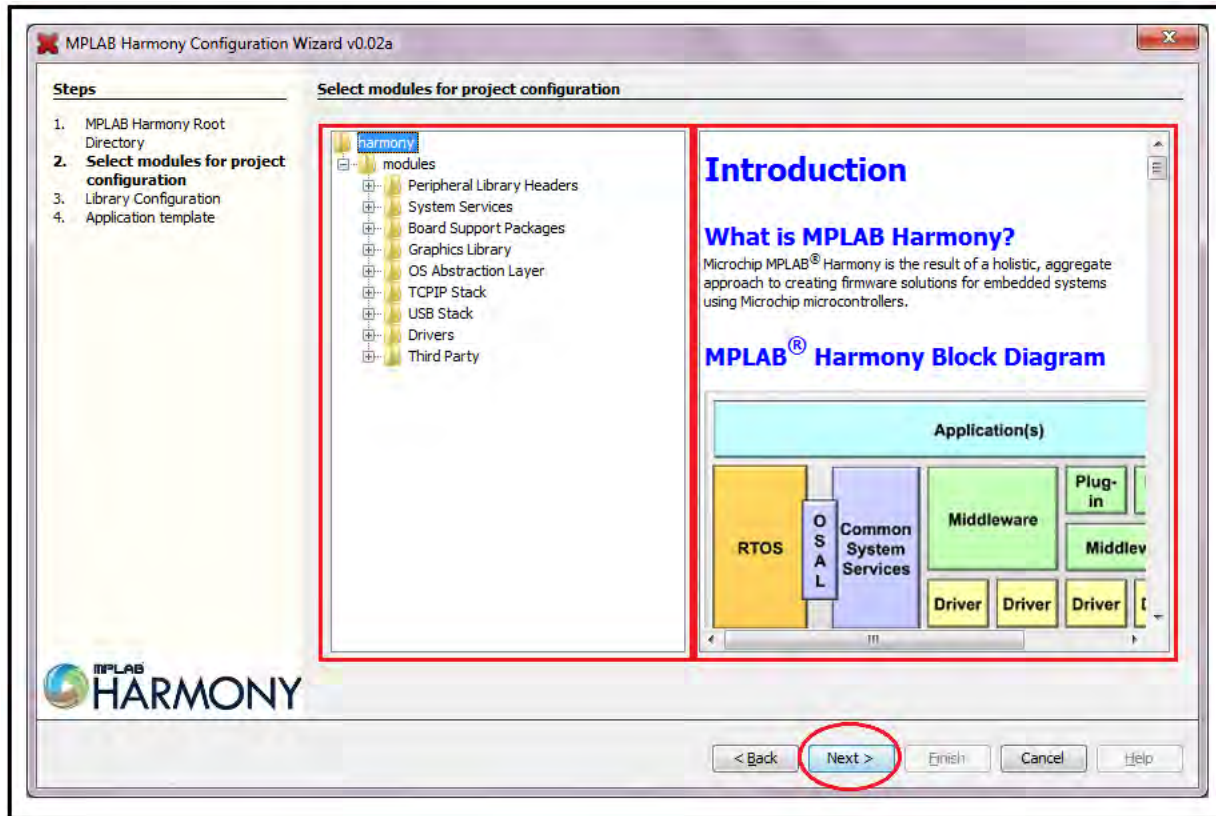
Description

1. To run the Configurator, select "MPLAB Harmony Configurator" from the "Tools" menu.
2. Browse to the root installation folder of the version of MPLAB Harmony that you're using. This folder must contain "manifest.xml". This file has the information and the path of various software modules available in the installed MPLAB Harmony version. manifest.xml file should not be moved or deleted from Root directory. Click "Next" when the root path is set.



Note: It is recommended that you create and maintain your MPLAB X IDE project under a subdirectory of the "<install_root>/apps" folder and use a relative path to this project.

3. In the Left-hand pane of the "Select Modules and Configurations" dialog, select the desired modules and respective sub-modules. The right-hand pane shows a brief introduction of the selected module. Click "Next" when the required modules for the project are selected.



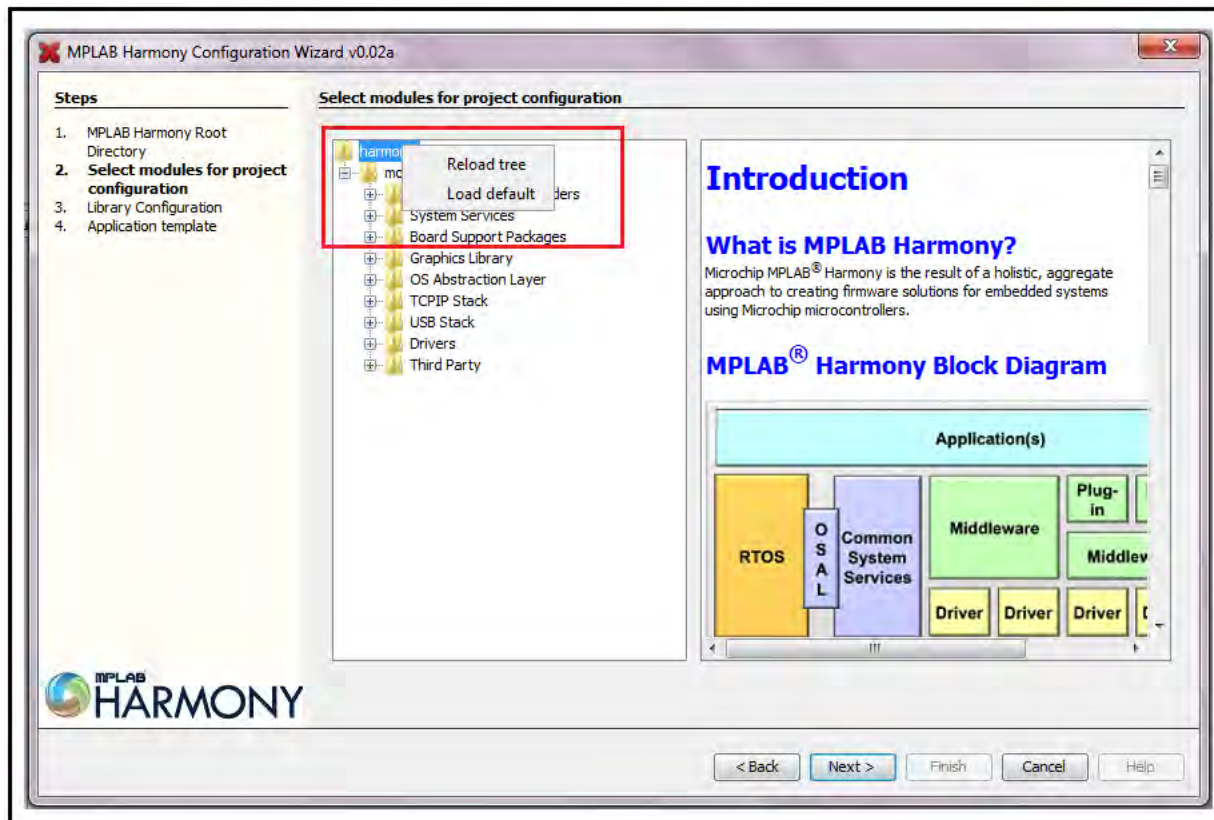
The tool has two configuration sections.

- **Project Configuration:** All the required modules and Sub modules could be selected here. The respective files would be added into the MPLAB X IDE project. (example: Graphics, TCP/IP, USB CDC, USB HID selection)
- **Library Configuration:** The individual configuration of each module could be done here. (Configuration for selected module from the selected section.)

If the user is required to add their custom module into the MPLAB Harmony directory structure and add the files into the project through Configurator, the manifest.xml should be modified to accommodate the custom module files. This would allow the Project Configurator to add the files to the MPLAB X IDE project from the manifest.xml for the custom module. This would not provide custom configuration capability, however the files can be added to the MPLAB X IDE project.

4. If there are any XML modifications those can be reflected by reloading the tree. To reload the project tree structure, right-click on "harmony" tree node and click "Reload Tree".

5. To get back to default project configuration of MPLAB Harmony, right-click on "harmony" tree node and click "Load default".

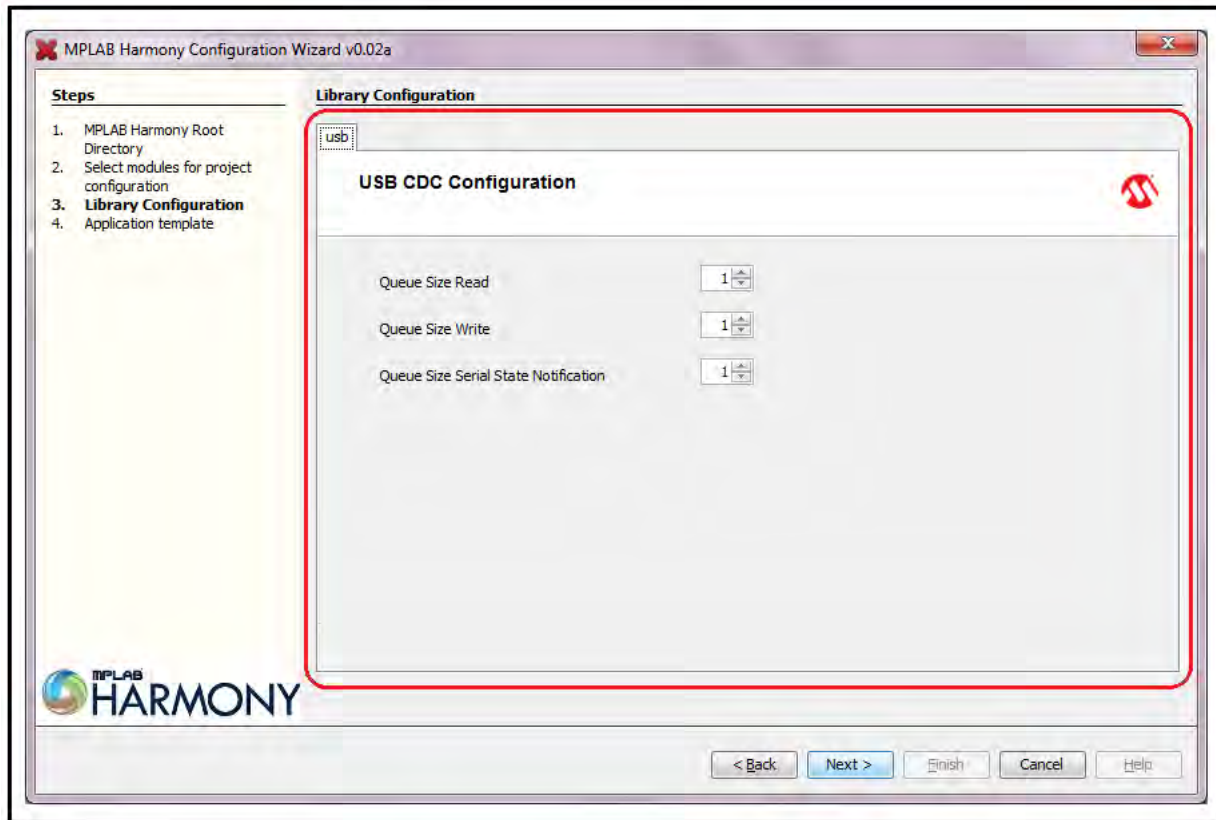


Note: Next window pane is for individual library (Modules and Sub modules) configuration. Currently only TCP/IP and USB library configuration is supported.

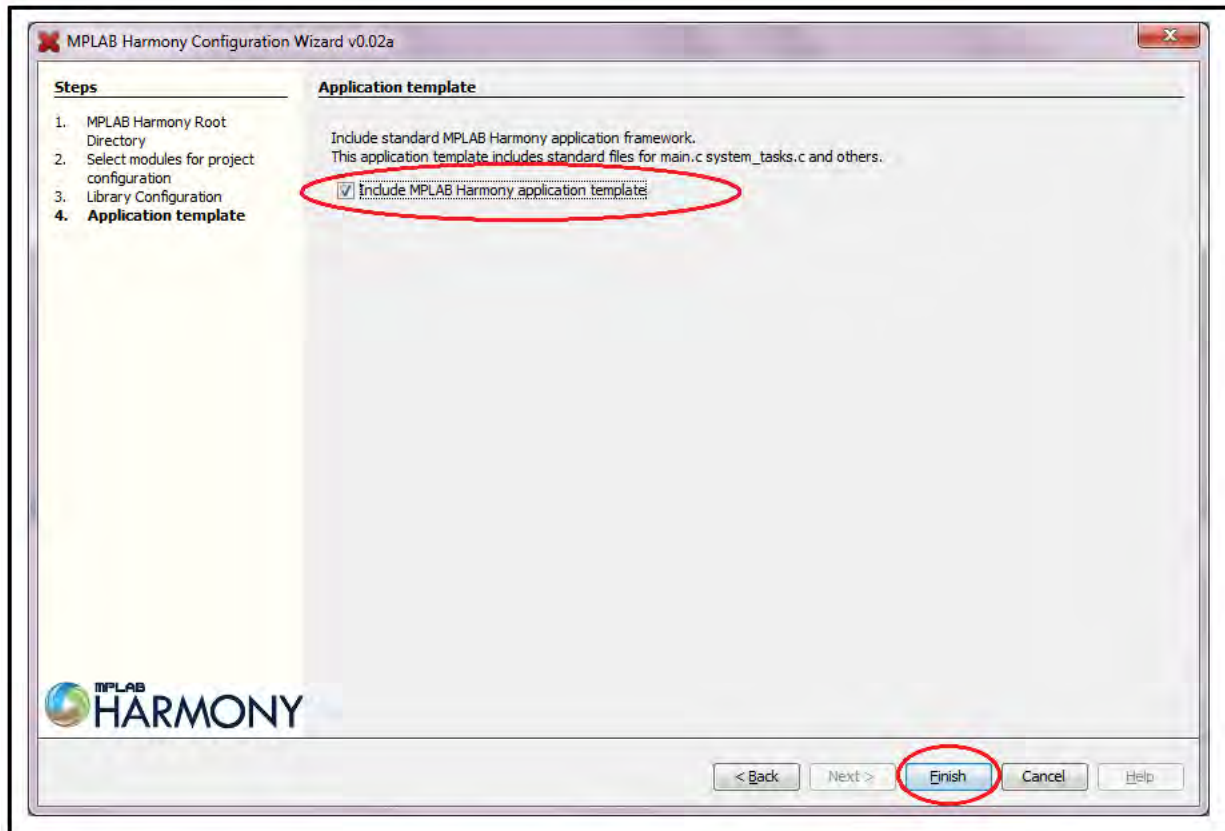
Though all the sub-modules of TCP/IP can be enabled or disabled through project configuration window pane, the individual sub module configuration is supported only for HTTP, SSL, SNMP and Telnet server. For USB, only CDC and HID class configuration basic support is provided.

Library Configuration

1. In the "Project configuration" pane, to configure USB CDC select the respective modules and click "Next". Next window pane is Library Configuration window as shown below. Configure USB CDC as required, click "Next".



2. In the "Application Template" pane, select the "Include application template" check box and select "Finish" to add the module files and template files to the MPLAB X IDE project.



Note: The generate project may compile for PIC32MZ devices, but not necessarily work on the hardware.

The set of template files added to the MPLAB X IDE project would require to be modified with additional configuration information and the custom application code.

Later versions of the configuration tool may provide additional configuration options and sample application code for each library, to create a executable project.

2.2.3 Tutorial

This tutorial explains about creating a new MPLAB X IDE project and configuring this project using the MPLAB Harmony Configurator. The tutorial provides step by step instructions on how to create a MPLAB X IDE project for the TCP/IP Stack.

Description

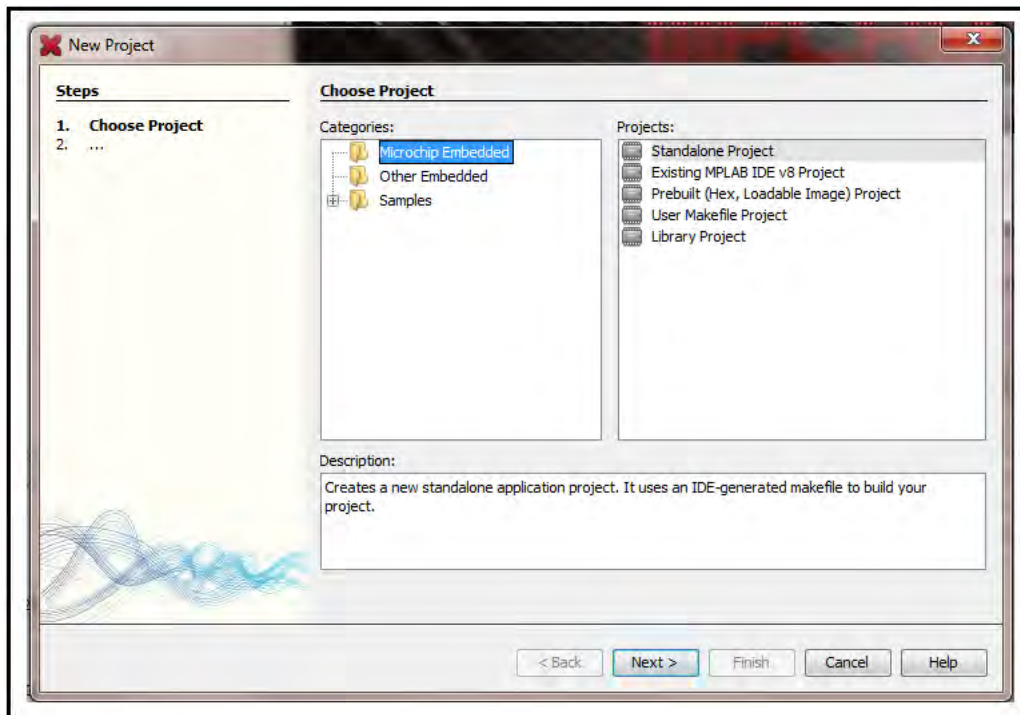
This tutorial explains about creating a MPLAB X IDE project for TCP/IP library by adding required configurations and TCP/IP library files, template files and Peripheral Library modules to the selected MPLAB X IDE project using the Configurator.

In this tutorial user would,

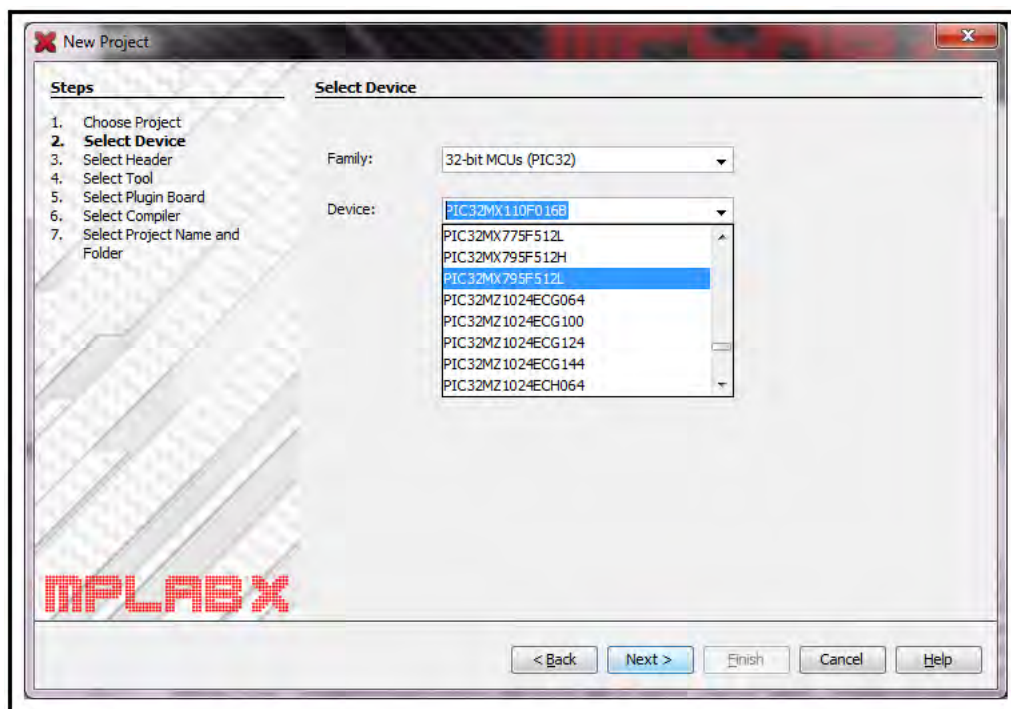
1. Create a new project in MPLAB X IDE.
2. Add required source and header files with required configurations for TCP/IP application using MPLAB Harmony Configurator to the MPLAB X IDE project.
3. Add the required Peripheral header files and template files to the MPLAB X IDE project.

Creating an MPLAB X IDE Project

1. Open MPLAB X IDE, and select File > New Project to open a project explorer window.
2. In Categories, select "Microchip Embedded" and in Projects, select "Standalone Project". Click Next.

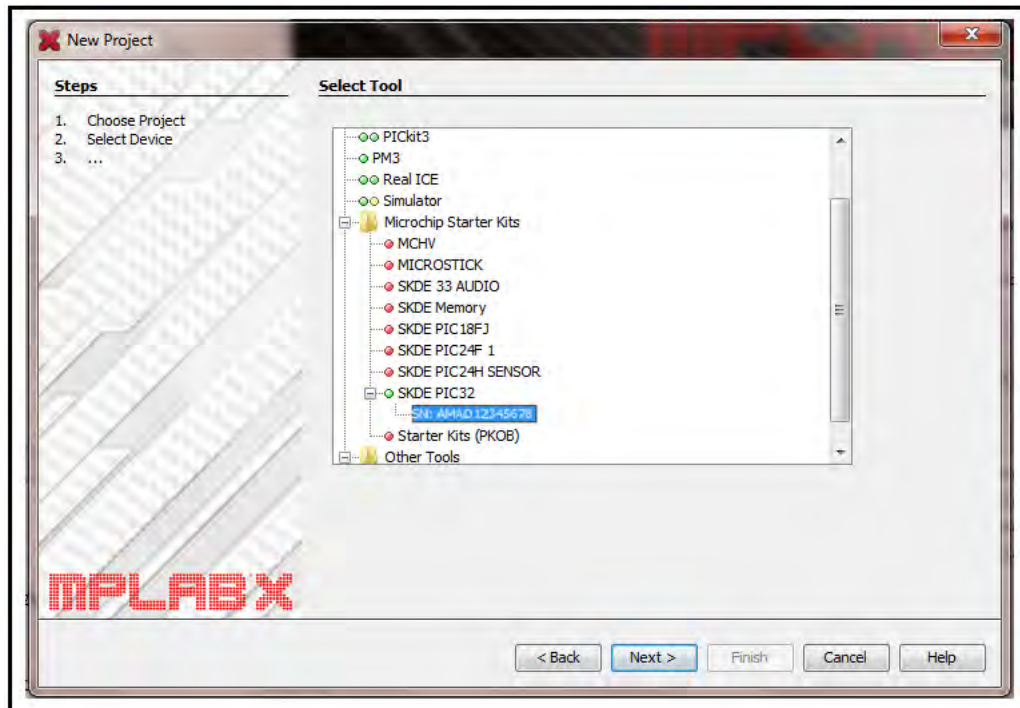


3. Select the PIC device.

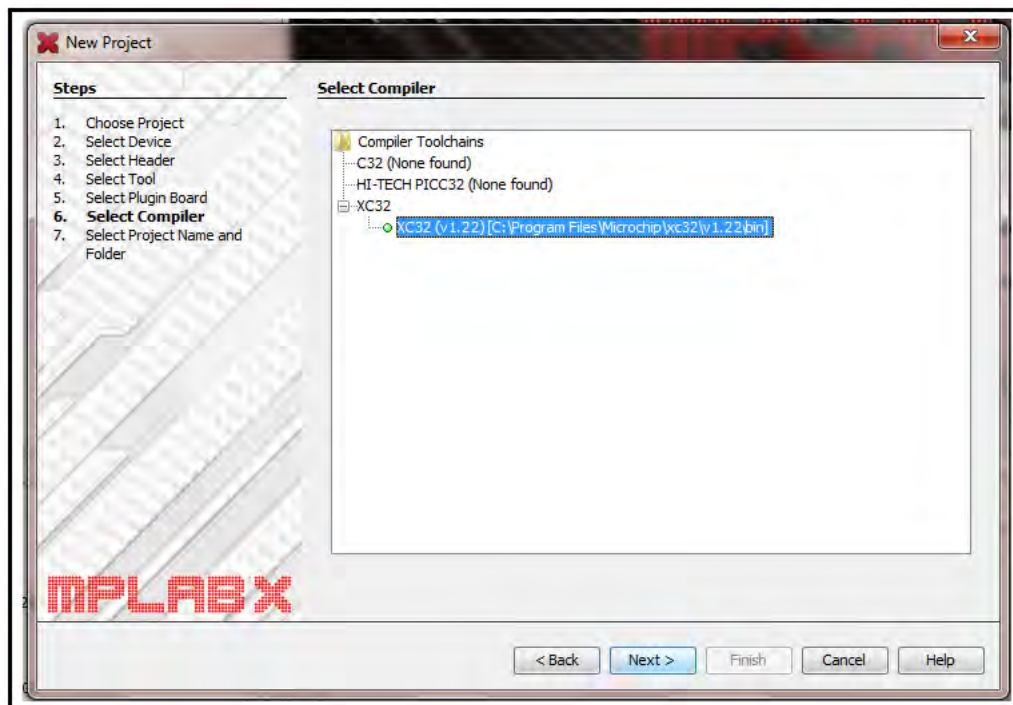


4. Select optional Debug Header. Click Next.

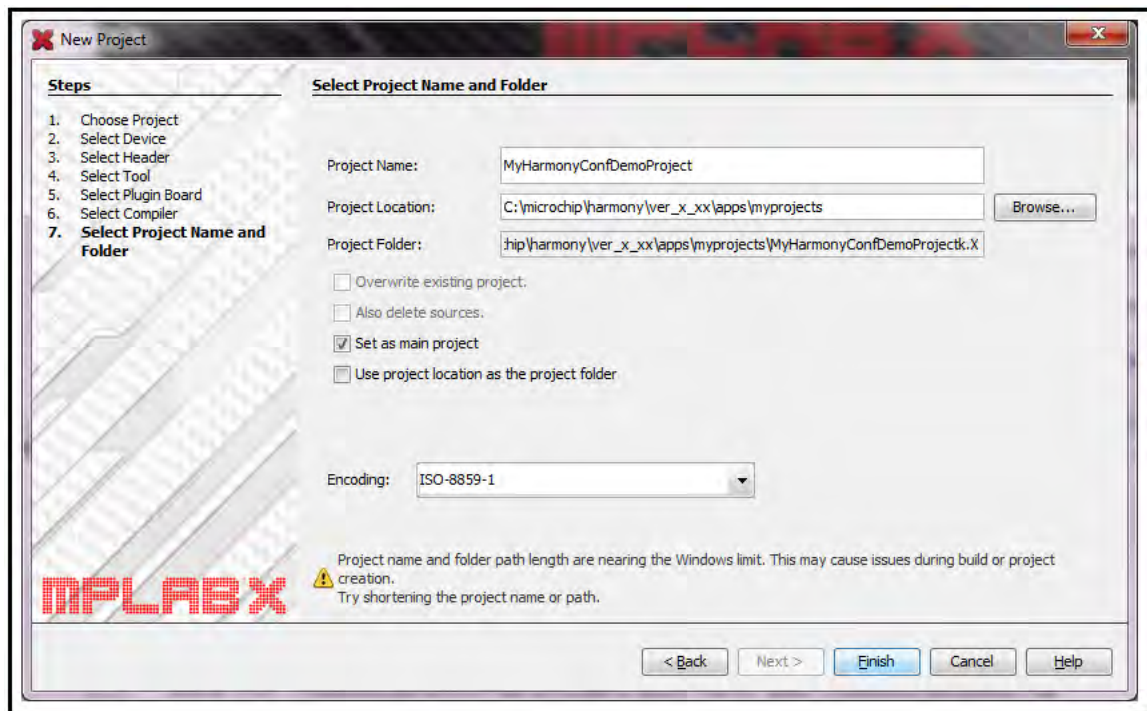
5. Select a hardware tool and Click Next.



6. Select the appropriate compiler. Click Next.



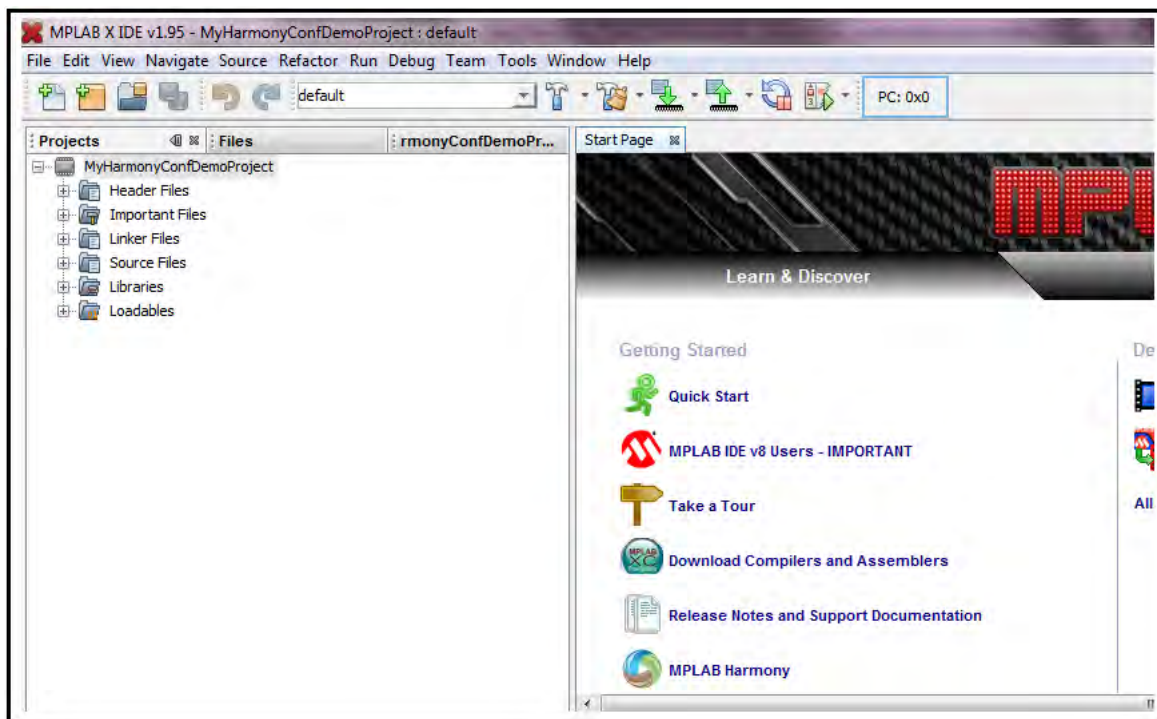
7. Name the MPLAB X IDE project and specify its path.



For this tutorial, create a project by name "MyHarmonyConfDemoProject". Set the project as main project. Click Finish. Now the project MyHarmonyConfDemoProject" is created.

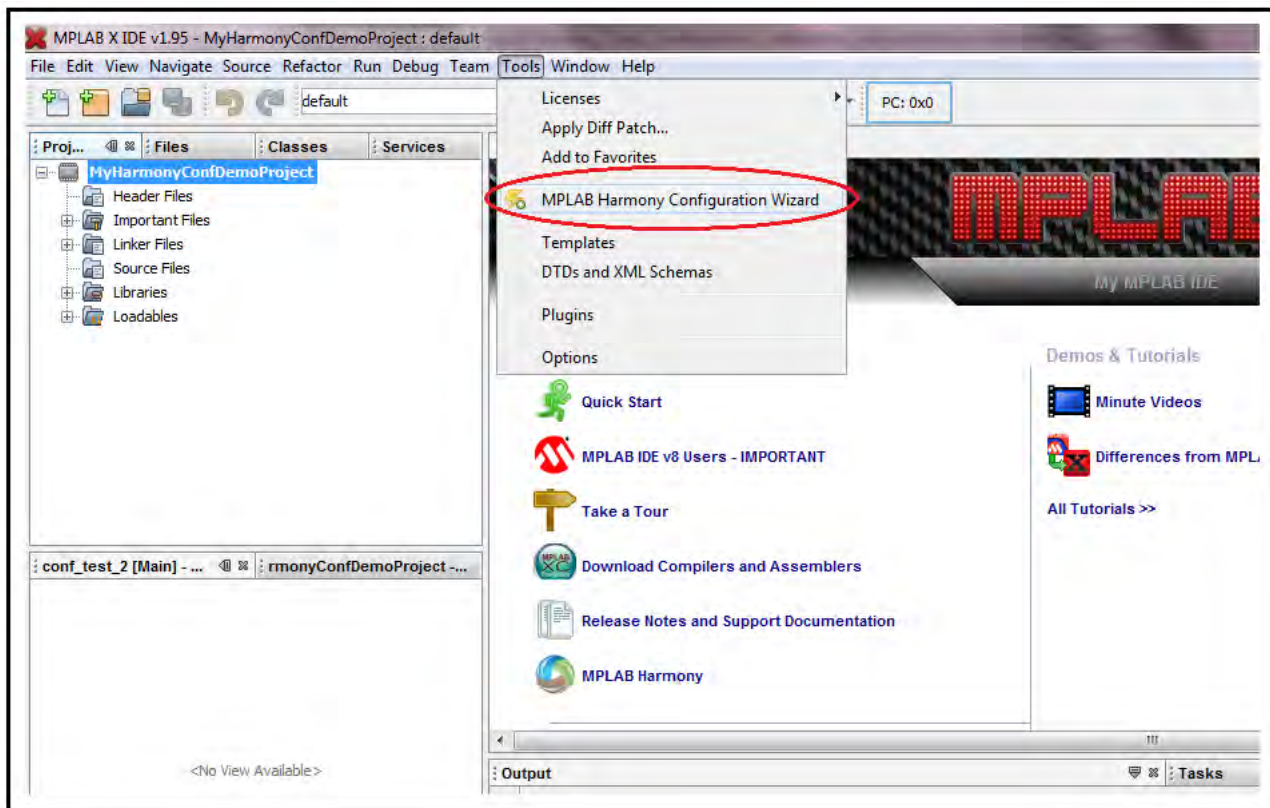
Note: It is recommended that you maintain your project under a subdirectory of the "<install_root>/apps" folder and use a relative path to your project.

Once the MPLAB X IDE project is created, you should see a screen similar to the picture as shown below.

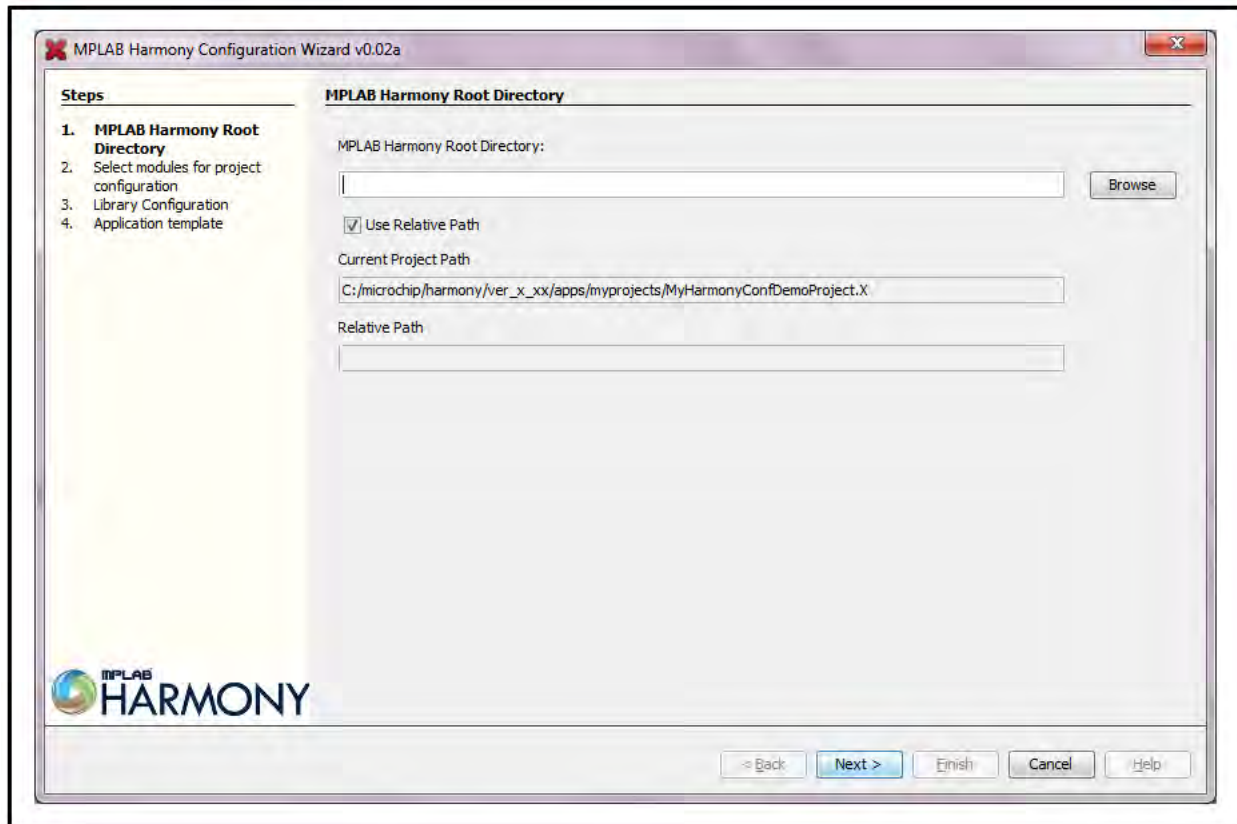


Using MPLAB Harmony Configurator

1. After creating the MPLAB X IDE project, select "Tools -> MPALB Harmony Configuration wizard" to open this plug-in with MPLAB X IDE.

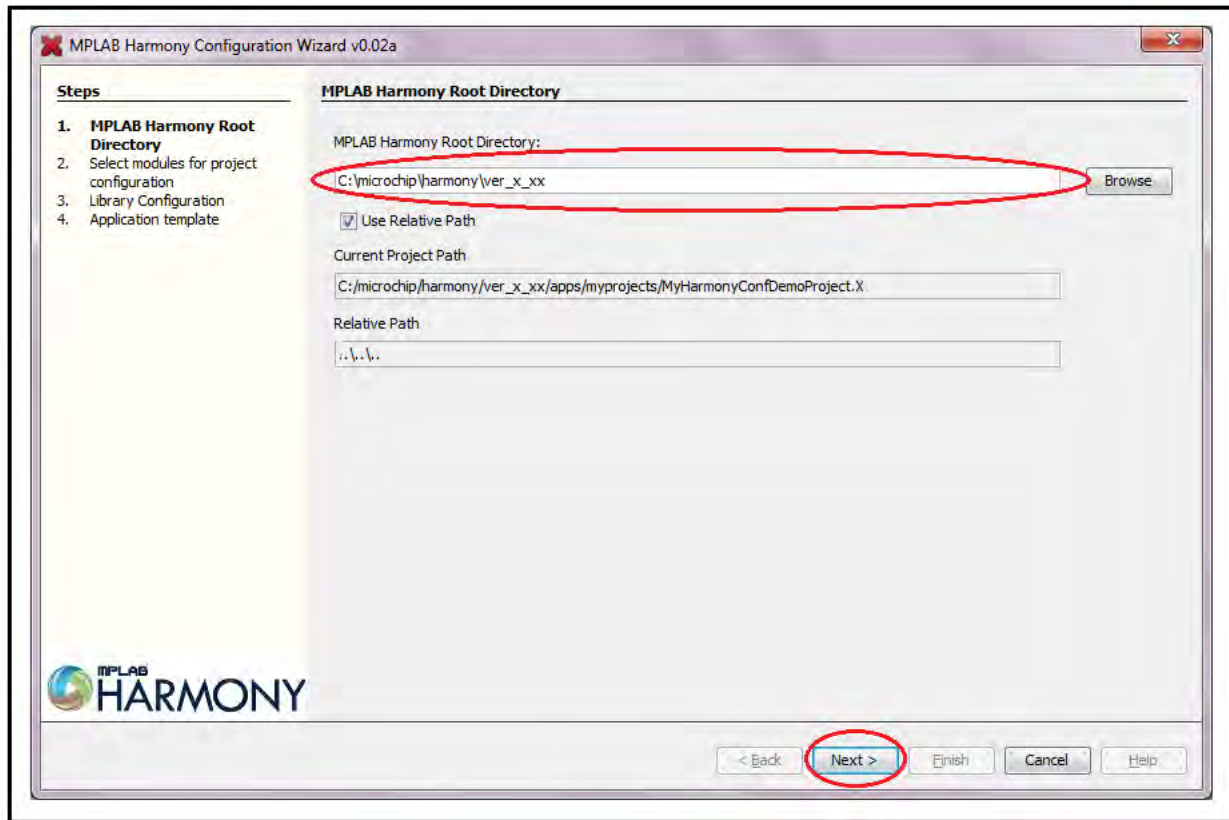


"MPLAB Harmony Configurator" will open in a new dialogue, as shown below.



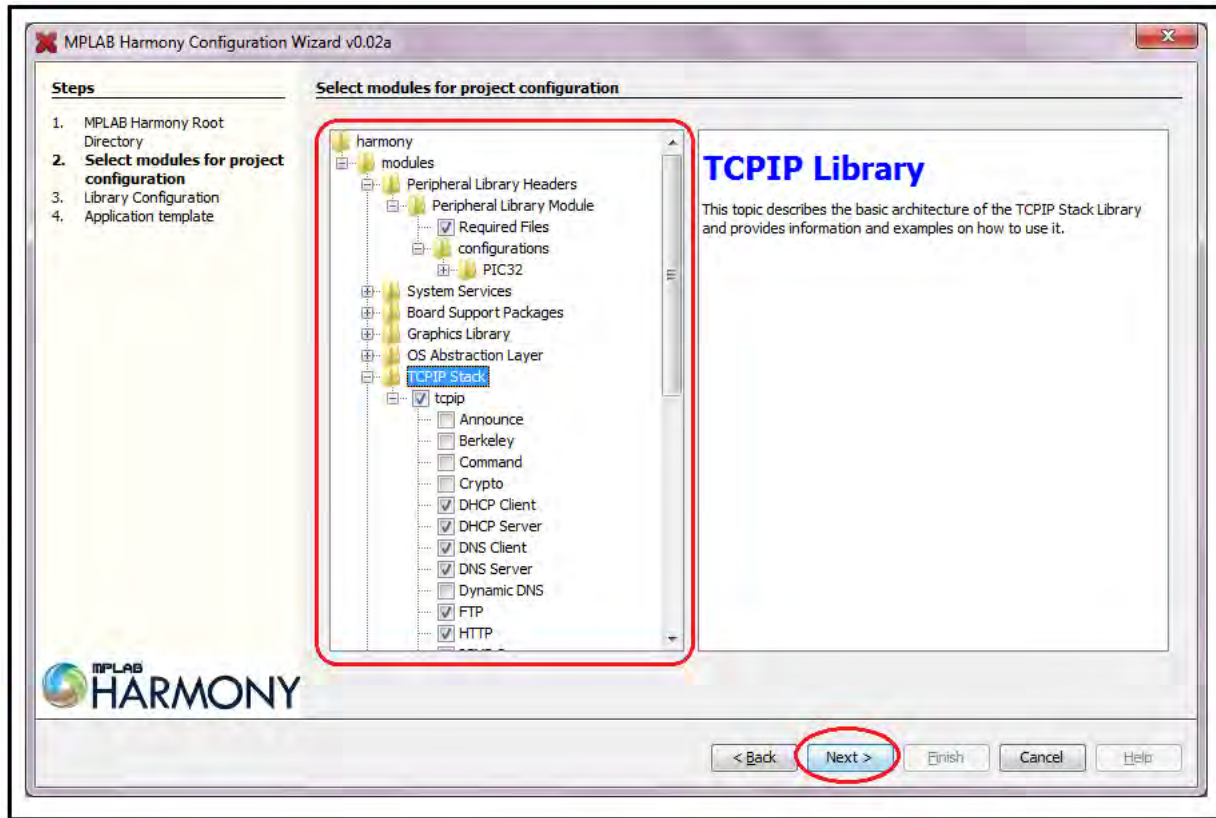
2. Browse to the root directory path under "MPLAB Harmony Root Directory" option. This path is remembered by the Configurator for subsequent usage.

Check or uncheck the "Relative Path" check box as required. Once the path is set, click "Next".

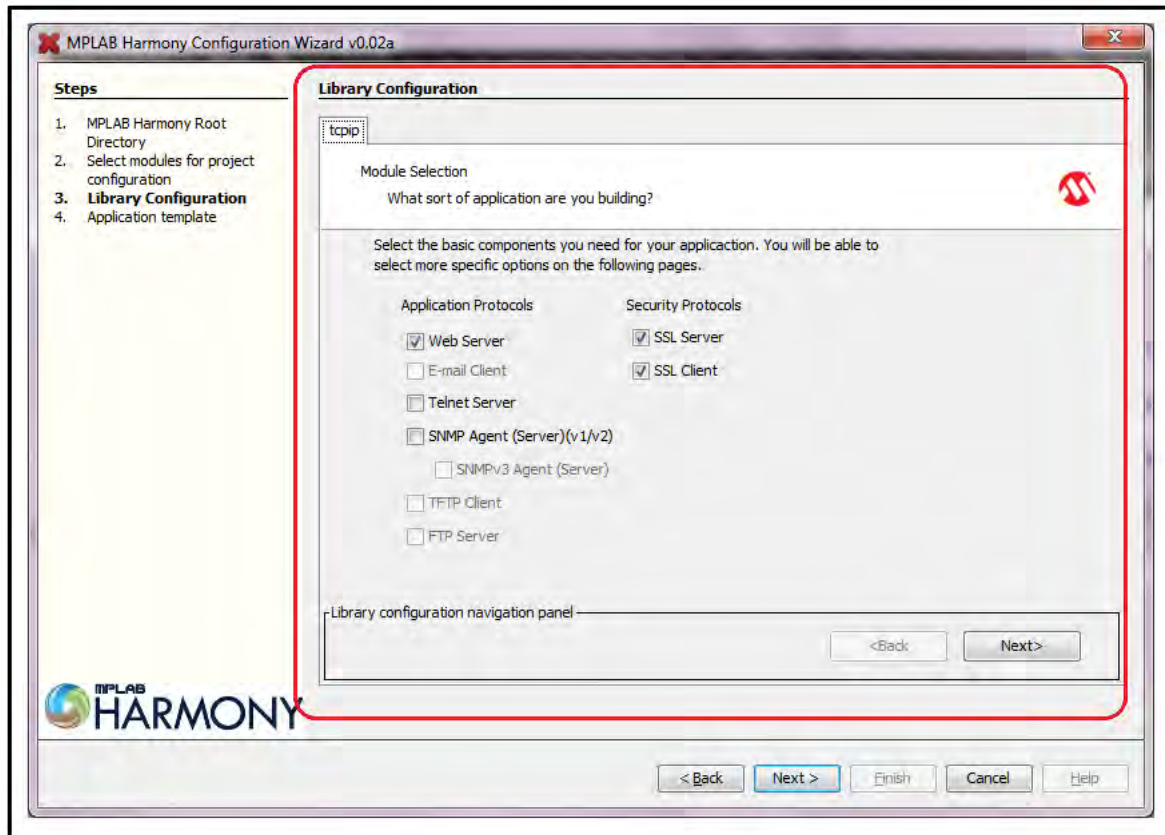


3. Now configure required middleware modules and Peripheral Library modules.

Peripheral Library headers are selected by default by the tool. For this tutorial user will configure TCP/IP Library. Select the required configuration modules of TCP/IP modules and also other middleware modules in the Tree structure. Click "Next" once the selection is complete.

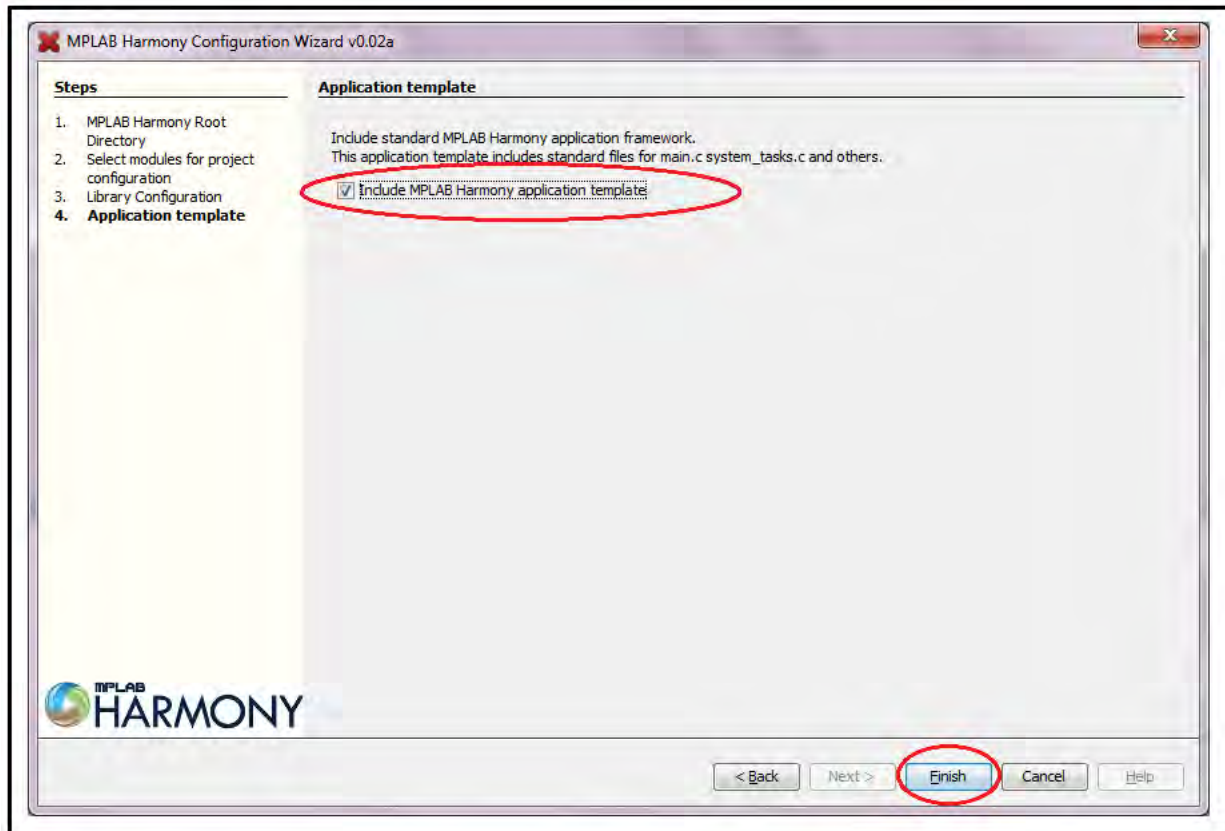


For this release, both the TCP/IP and USB module Library Configuration Pane will appear. For this Tutorial, the TCP/IP module is selected (default selection "tcpip "node) in tree, we get the Library Configuration as below.

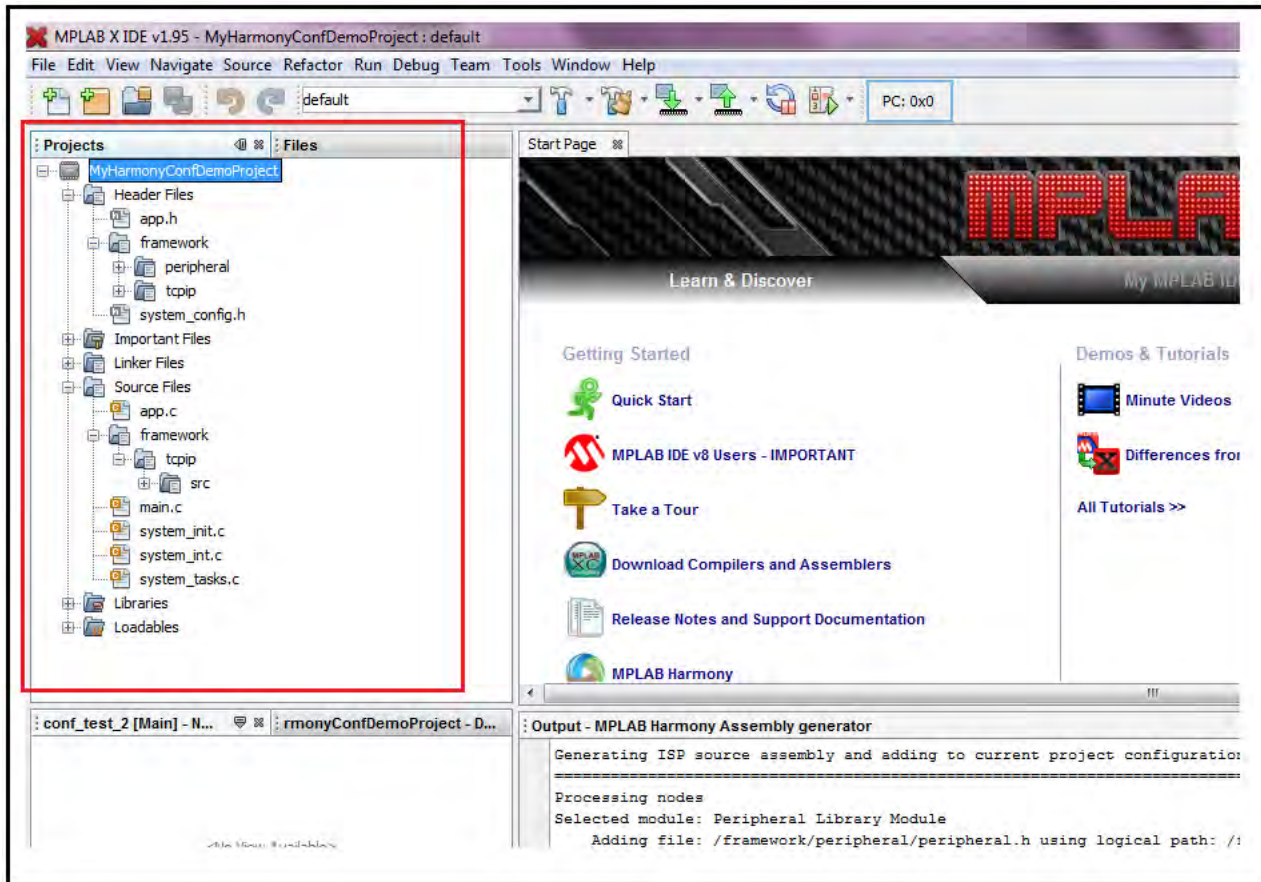


4. Select the required configurable items of TCP/IP, and click next (Next button in "Library configuration navigation pane" which takes to next TCP/IP configuration pane, and continue until the last configuration pane. Once TCP/IP configuration is done, click Next (Next button at the bottom of TCP/IP Library configuration pane) which takes into the next window "Application Template" to include template files.

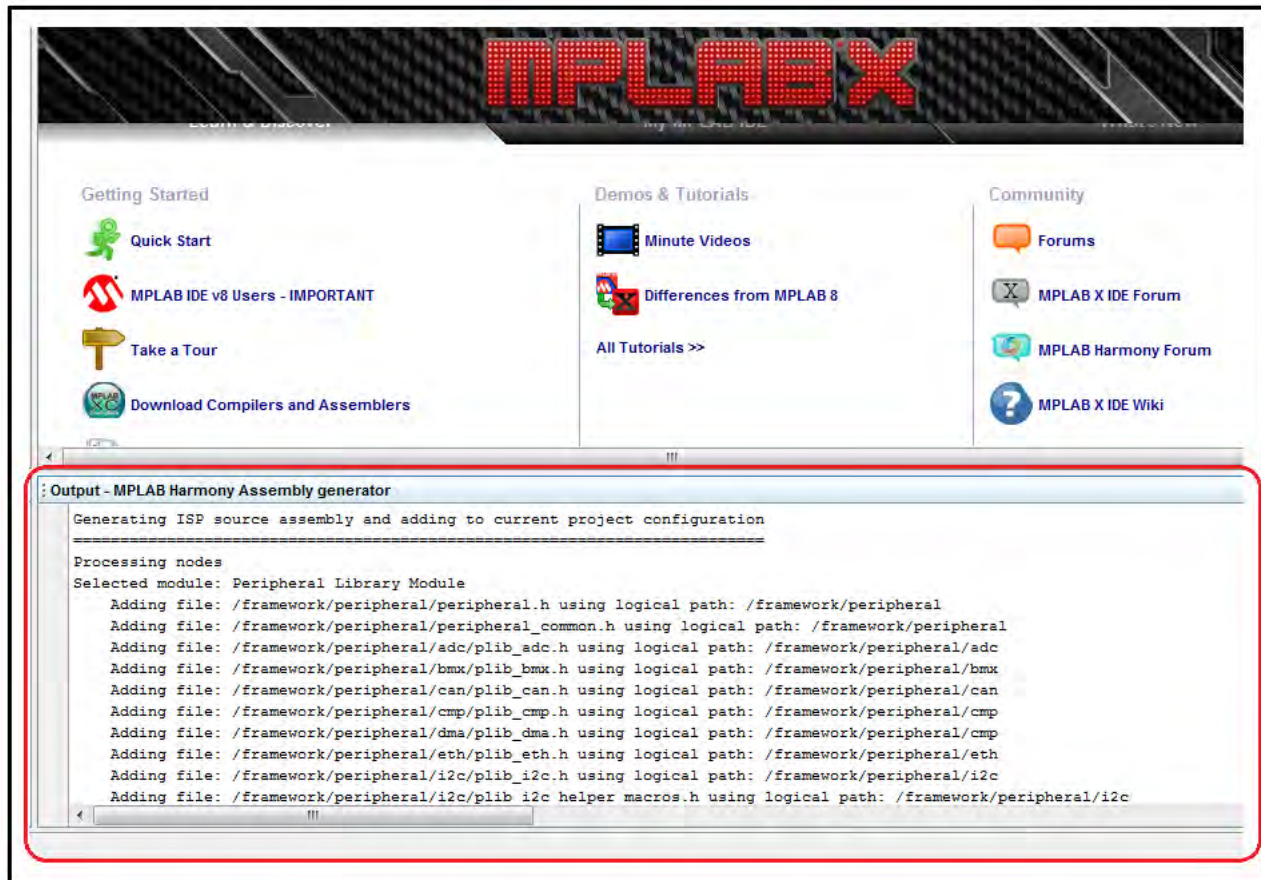
5. Select to Include MPLAB Harmony Application template files. Tool would add template files (main.c, app.c, system_init.c, system_tasks.c, system_int.c, system_config.h and app.h) to MPLAB X IDE project. Click "Finish".



6. Check for selected files being added in MPLAB X IDE project. The following figure shows the MPLAB X IDE project with the added files.



The "MPLAB Harmony Assembly Generator" window in MPLAB X IDE shows the list of files added from "MPLAB Harmony Configurator" to the MPLAB X IDE project.



3 Applications Help

3.1 Applications Overview

Applications use MPLAB Harmony libraries and define the behavior of a system.

Description

Applications determine how MPLAB Harmony libraries (device drivers, middleware, and system services) are used to do something useful. In an MPLAB Harmony system, there may be one main application, there may be multiple independent applications or there may be one or more Operating System (OS) specific applications. Applications interact with MPLAB Harmony libraries through well defined interfaces. Applications may operate in a strictly polling environment, they may be interrupt driven, they may be executed in OS-specific threads, or they may be written so as to be flexible and easily configured for any of these environments.

Applications generally fit into one of the following categories:

Demo Applications:

Demos applications are provided (with MPLAB Harmony or in separate installations) to demonstrate typical or interesting usage models of one or more MPLAB Harmony libraries. Demonstration applications can demonstrate realistic solutions to real-life problems.

Sample Applications:

Sample applications are extremely simple applications provided with MPLAB Harmony as examples of how to use individual features of a library. They will not normally accomplish anything useful on their own. They are provided primarily as "documentation" to show how to use a library.

Skeletal Applications:

Skeletal applications are provided as bare-bones "skeletons" (potentially targeted to a specific usage or market) that MPLAB Harmony developers can use as a starting point to for writing their own applications. Skeleton applications contain the necessary calls, configuration definitions and subroutine stubs with "To Do" items clearly marked for MPLAB Harmony developers to fill in as desired to create a working application.

Customer Applications:

Customer applications are (of course) not provide by Microchip. Rather, they are any applications developed by customers to suit specific hardware and software needs. Refer to the [Start Here](#) help topic for information on the structure and to learn how to develop your own MPLAB Harmony applications.

Application Directories

Applications are contained in the "<install-root>/apps" folder. Under the "apps" folder, they are grouped by technology, board, and/or market segment.

Examples:

Folder	Type of Demos
<install-root>/apps/examples/peripheral	Contains simple peripheral library example applications
<install-root>/apps/fs	Contains graphics library demonstration applications
<install-root>/apps/gfx	Contains TCP/IP networking stack demonstration applications
<install-root>/apps/tcpip	Contains USB device demonstration applications

<install-root>/apps/usb/device	Contains USB host demonstration applications

Additional Information

Refer to [Release Notes & Information](#) for a complete listing of applications included with this release.

Refer to the help documentation for each individual application for build instructions and information on required hardware and application usage.

3.1.1 System Overview

To support the framework, the application (or application developer) is responsible for the following:

- **Configuring the Framework** - Required, unless using a pre-built library. Each driver, middleware layer, and system service require build-time configuration options that must be defined for the specific application, board, and processor that is used by any particular demo or customer application. Please refer to the help file of that module on the list of configuration options, and how they apply to the application. The application can use the board support package configuration file or define its own configuration for the board.
- **Building the Framework Library** - Required, unless using a prebuilt library
- **Implementing the System Initialization service.** Please refer to the help file for the system initialization service for information on implementing it statically or dynamically
- **Implementing a Interrupt Service Routine (ISR)** - Required, unless polled. Please refer to the help file for system tasks service for examples on implementing ISRs.
- **Implementing the System Tasks routine** - Please refer to the help file for system tasks service for how to implement it statically or dynamically

3.2 File System Demonstrations

3.2.1 Introduction

MPLAB Harmony File System Demonstration Help

Description

This help file contains instructions and associated information about MPLAB Harmony File System demonstration applications, which are contained in the MPLAB Harmony Library distribution.

3.2.2 Release Notes

MPLAB Harmony Version: v0.70b **File System Demonstrations Version :** 0.01a **Release Date:** 18Nov2013

The interface can change in the beta and/or 1.0 release.

For the latest release information, please refer to the File System Service Library [Release Notes](#) section.

New This Release:

Nothing to report for this release.

Known Issues:

Nothing to report for this release.

3.2.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have

paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

3.2.4 Demonstrations

Provides instructions on how to run the demonstration applications.

Description

The following demonstration applications are available in this release of the MPLAB Harmony File System framework.

3.2.4.1 Overview

Provides an overview of how most of the File System Demonstration applications are structured.

Description

The MPLAB Harmony File System Demonstration Applications show example usage of the File System framework. These demonstration applications also contain example configurations of media drivers such as Internal Flash and SD Card, that can be used readily in the end application with the some changes to initialization parameters, if required. A demonstration application may have multiple MPLAB X IDE project configurations (for different demonstration hardware and microcontrollers). The relevant configuration should be selected in the MPLAB X IDE before compiling the demonstration. Figure 1 shows the dialog box in MPLAB X IDE that allows selection of different configurations.

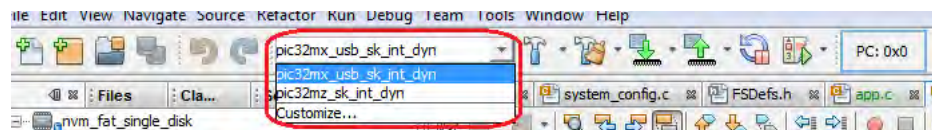


Figure 1: Demonstration MPLAB X IDE Configurations

The project will contain files which are configuration specific. Wherever applicable, these configuration files are placed in separate MPLAB X IDE project logical folders. Figure 2 shows how the configuration dependant `nvm_disk_images.c` file is included in the project. In this case two configurations (`pic32mx_usb_sk_int_dyn` and `pic32mz_sk_int_dyn`) are supported in the project.

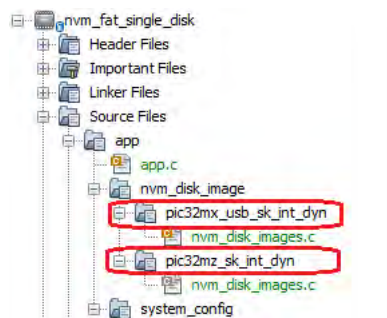


Figure 2: Configuration specific file in File System Demonstration Projects.

Note that some demonstration projects may not support all configurations. In general, the following the demonstration project contain the following MPLAB X IDE configurations:

- **pic32mx_usb_sk_int_dyn:** This configuration runs on the PIC32 USB Starter Kit. The media drivers are configured for interrupt and dynamic operation

- **pic32mz_sk_int_dyn:** This configuration runs on the PIC33MZ Embedded Connectivity (EC) Starter Kit. Some demonstration projects that support this configuration may need the Multimedia Expansion Board II (MEB II) to attached to the starter kit. The media drivers in this configuration are configured for interrupt and dynamic operation.

Unless specified, the File System demonstration application are structured as indicated in Figure 3.

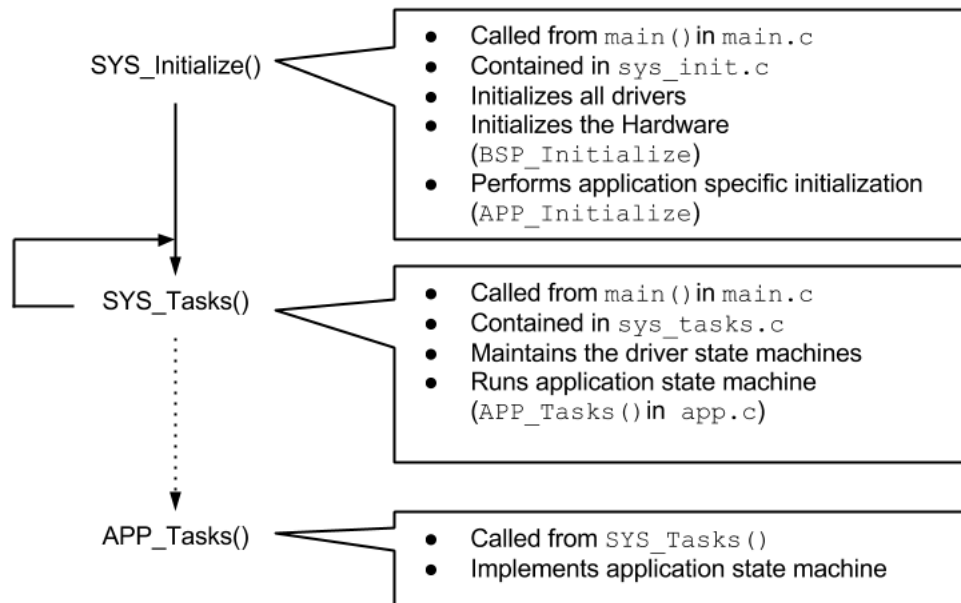


Figure 3: File System Demonstration Application General Architecture

Here are some general notes while working with the demonstration applications:

1. The system and application is initialized in `SYS_Initialize()` function, implemented in `sys_init.c`. The `sys_init.c` file is configuration specific. The `SYS_Initialize()` function initializes the application (`APP_Initialize()` function) and the BSP (`BSP_Initialize()` function).
2. The application then calls `SYS_Tasks()` function in an infinite loop. This function updates the state machines of all system components in the demonstration application and also calls the `APP_Tasks()` function. The is implemented in file `sys_tasks.c` which is configuration specific.
3. All Interrupt service routines are implemented in `sys_interrupt.c`. This file is configuration specific.

3.2.4.2 nvm_fat_single_disk

This demonstration uses a FAT12 image of file on NVM flash memory and demonstrates the working of all file system functions.

Description

This demonstration shows an example of implementing a FAT12 disk in device Flash memory. The demonstration contains a FAT12 disk image consisting of a Master Boot Record (MBR) sector, Logical Boot Sector, File Allocation Table, Root Directory Area. The demonstration opens an existing file named "FILE.TXT" and performs all file system related function calls on the file viz. `SYS_FS_FileStat`, `SYS_FS_FileSize`, `SYS_FS_FileSeek`, `SYS_FS_FileEOF`. Finally, the string "Hello World" is written to this file. The file is then closed and reopend for read. The read string is then compared with the string which was written to the file. If the string compare is successful, LED indication is provided.

3.2.4.2.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Following board is supported for PIC32MX devices: -

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-

Following board is supported for PIC32MZ devices: -

Demonstration Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.2.4.2.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

3.2.4.2.3 Running the Demonstration

Provides instructions on how to build and run the NVM FAT single disk demonstration.

Description

Select the MPLAB X IDE project configuration: -

- pic32mx_usb_sk_int_dyn -- For PIC32MX devices
- pic32mz_sk_int_dyn -- For PIC32MZ devices

Build the selected configuration in the MPLAB X IDE project and program the demonstration board. The execution status (pass/fail) of the demonstration is indicated by LEDs on the demonstration board.

Demonstration Board	Demonstration Successful	Demonstration Failure
PIC32 USB Starter Kit II	LED3	LED1
PIC32MZ Embedded Connectivity (EC) Starter Kit	Green LED	Red LED

About the demonstration:

This demonstration shows an example of

- implementing a FAT12 disk in device Flash memory.
- opening a file for read or write.
- implements all the file system functions.

- closing a file.

The demonstration contains a FAT12 disk image consisting of a Master Boot Record (MBR) sector, Logical Boot Sector, File Allocation Table, Root Directory Area. This image is implemented in the file `nvm_disk_images.c` (this is project configuration specific file and is contained in the `nvm_disk_images` logical folder in the MPLAB X IDE project). The image contains a single file named "FILE.TXT" which contains the string "Data".

The demonstration opens an existing file named "FILE.TXT" and performs all file system related function calls on the file viz. [SYS_FS_FileStat](#), [SYS_FS_FileSize](#), [SYS_FS_FileSeek](#), [SYS_FS_FileEOF](#). Finally, the string "Hello World" is written to this file. The file is then closed and reopened for read. The read string is then compared with the string which was written to the file. If the string compare is successful, LED indication is provided.

The demonstration application logic is implemented as a state machine in the `APP_Tasks()` function in the file `main.c`.

1. The disk is first mounted using the [SYS_FS_Mount\(\)](#) function. The `"/dev/nvma1"` path instructs the mount command to mount an internal flash volume. The volume is mounted against a FAT type file system and mounted at `"/mnt/myDrive/"`.
2. If the mount is successful, the application opens a file "FILE.TXT" for reading and writing (`SYS_FS_FILE_OPEN_READ_PLUS`) with a [SYS_FS_FileOpen\(\)](#) function. The valid file handle is received once a successful opening of file is performed.
3. If file open is successful, the status of file "FILE.TXT" is stored in the `appData.fileStatus` structure, using [SYS_FS_FileStat\(\)](#) function.
4. If the file status check is successful, the size of the "FILE.TXT" is checked by passing the file handle to the [SYS_FS_FileSize\(\)](#) function.
5. If the file size check is successful, the size of file is compared with the size element received earlier as a part of `appData.fileStatus` structure. If both value matches, then the code moves to the next step of file seek.
6. The file pointer is then moved by 4 bytes from the start of the file by calling the function [SYS_FS_FileSeek\(\)](#) and passing parameter as `SYS_FS_SEEK_SET` (seek from the beginning of the file).
7. If the file seek operation is successful, the file pointer should have reached the end of the file. This is because, the content of the FILE.TXT had only 4 byte string = "Data". The end of file is verified by calling the function [SYS_FS_FileEOF\(\)](#). If the function returns "true" (end of file reached), the next step is performed.
8. In the next step, the file pointer is moved again by `[(-1)*size of file]`, from the end of the file by calling the function [SYS_FS_FileSeek\(\)](#) and passing parameter `SYS_FS_SEEK_END` (seek from the end of the file).
9. If the file seek operation is successful, the file pointer should have reached the beginning of the file. 4 Bytes are read from the file using [SYS_FS_FileRead\(\)](#) function (into a buffer).
10. If the read is successful, the content of file (present in the buffer) is compared with the "Data" string. If the comparison passed, the code moves to the next step.
11. In the next step, the file pointer is moved again by `[(-1)*size of file]`, from the end of the file by calling the function [SYS_FS_FileSeek\(\)](#) and passing parameter `SYS_FS_SEEK_END` (seek from the end of the file).
12. If the file seek operation is successful, the string "Hello World" is written to the file using [SYS_FS_FileWrite\(\)](#) function.
13. If the write operation is successful, the file is closed.
14. The file "FILE.TXT" is opened again in read mode (`SYS_FS_FILE_OPEN_READ`).
15. If the file open is successful, the content of the file is read using [SYS_FS_FileRead\(\)](#) function (into a buffer).
16. If the read operation is successful, the content of the file (present in the buffer) is compared with the "Hello World" string using the `strcmp()` function. A LED indicates the success of the demonstration.

3.2.4.3 sd_card_fat_single_disk

This demonstration uses the SDTM Card with FAT file system as media and performs a read/ write / verify operation on the files using long file names (LFN).

Description

This demonstration shows an example of using the MPLAB Harmony File System to access and modify the contents of an SDTM Card. The demonstration opens a file "FILE_TOO_LONG_NAME_EXAMPLE_123.JPG" on the SD Card, reads the content of the file and writes the content into another file "FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG" (create a copy of one file into another file).

The input file "FILE_TOO_LONG_NAME_EXAMPLE_123.JPG" is not provided along with the release package. It could be any arbitrary JPG (picture) file chosen by user and then suitably renamed to "FILE_TOO_LONG_NAME_EXAMPLE_123.JPG". The reason for choosing a JPG file for test purpose is that the duplicate file "FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG" created by the FS demonstration could be easily verified for correctness by inserting the SD card in computer and opening the "FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG" file. If the file "FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG" opens for viewing on the computer, the test is deemed to have passed. Otherwise, if the "FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG" does not open (is corrupted), the test will be considered as failed.

3.2.4.3.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Following board is supported for PIC32MX devices: -

Demonstration Board (click link for board information)
PIC32MX795F512L Plug-In-Module (PIM)
PICtail Daughter Board for SD and MMC

Note: This board can not be used by itself. An [Explorer 16 Development Board](#) is required.

Following board is supported for PIC32MZ devices: -

Demonstration Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

Note: This board can not be used by itself. An [Multimedia Expansion Board II \(MEB II\)](#) is required to run the demonstration.

3.2.4.3.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[Explorer 16 Development Board Based Demonstrations](#)

Use the following instructions for all Explorer 16 Development Board-based demonstration boards:

PIC32MX795F512L PIM

1. Insert the PIM into the Explorer 16 Development Board PIM connector.

2. Jumpers JP1, JP2 and JP3 on the SD Card PICTail Daughter Board should [connect](#) to points 2 and 3 on their respective connectors as shown [here](#).
3. Insert the SD Card PICTail Daughter Board into the PICTail Plus connector on the Explorer 16 Development Board.
4. Insert a SD card into the SD card connector on the SD Card PICTail Daughter Board.
5. Power up the board.

[Multimedia Expansion Board II \(MEB II\)](#)

No hardware setting change is required. Insert the micro SD card into the connector and power up the board.

3.2.4.3.3 Running the Demonstration

Provides instructions on how to build and run the SD card FAT single disk demonstration.

Description

Select the MPLAB X IDE project configuration: -

- exp16_32mx_int_dyn -- For PIC32MX devices
- pic32mz_sk_int_dyn -- For PIC32MZ devices

Insert the SD card, which contains the file "FILE_TOO_LONG_NAME_EXAMPLE_123.JPG". The file can be of any size.

Compile the selected configuration in the MPLAB X IDE project and run the code. After a few seconds, once the LED glows, remove the SD card from the SD Card PICTail Daughter Board (for PIC32MX) or MEB2 Board (for PIC32MZ) and insert it into the SD card reader on a PC. Examine the contents of the SD card and another file named "FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG" would have been created. "FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG" will be a copy of "FILE_TOO_LONG_NAME_EXAMPLE_123.JPG". Verify both the files opens for viewing on the PC.

Demonstration Board	Demonstration Successful	Demonstration Failure
Explorer 16 Development Board	LED5 (D5)	LED6 (D6)
PIC32MZ Embedded Connectivity (EC) Starter Kit	Green LED	Red LED

About the demonstration:

This demonstration shows an example of

- implementing a FAT File system (FAT16/ FAT32) on SD card.
- implementing long file name (LFN).
- opening a file for read or write.
- closing a file.

The demonstration opens a file "FILE_TOO_LONG_NAME_EXAMPLE_123.JPG" on the SD Card, reads the content of the file and writes the content into another file "FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG" (create a copy of one file into another file).

The demonstration application logic is implemented as a state machine in the APP_Tasks() function in the file main.c.

1. The disk is first mounted using the [SYS_FS_Mount\(\)](#) function. The "/dev/mmcblk1" path instructs the mount command to mount a SD card volume. The volume is mounted against a FAT type file system and mounted at "/mnt/myDrive/".
2. If mount is successful, the volume is unmounted by passing the mount name "/mnt/myDrive" to [SYS_FS_Unmount\(\)](#) function. This unmounting is done for demonstration purpose only. Real application do not need to unmount unless it is required for the application.

3. If unmount is successful, the mounting process is repeated.
4. If the mount is successful, the application opens a file "FILE_TOO_LONG_NAME_EXAMPLE_123.JPG" for reading with a [SYS_FS_FileOpen\(\)](#) function.
3. If the file open is successful, the application opens a file "FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG" for writing with [SYS_FS_FileOpen\(\)](#) function. The attributes for file write is selected as "SYS_FS_FILE_OPEN_WRITE". Hence, if the file does not exist on the drive, the file is created.
4. If the file open is successful, 512Bytes from the file "FILE_TOO_LONG_NAME_EXAMPLE_123.JPG" is read into application buffer using [SYS_FS_FileRead\(\)](#) function.
5. If the file read successful, the 512Bytes is written from the application buffer to the FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG file using [SYS_FS_FileWrite\(\)](#) function.
6. If the write operation is successful, the end of file for FILE_TOO_LONG_NAME_EXAMPLE_123.JPG is checked. If the end of file is not reached, the process of reading and writing continues (step - 4 and step - 5).
7. Once end of file is reached, both the files are closed with [SYS_FS_FileClose\(\)](#) function.
8. Finally, a LED indicates the success of the demonstration.

3.2.4.4 nvm_sdcard_fat_multi_disk

This demonstration uses NVM and SDTM Card as 2 media and reads files from one media and write onto another media.

Description

This demonstration shows an example of using the MPLAB Harmony File System to access files across multiple media. The demonstration contains a FAT12 disk image consisting of a Master Boot Record (MBR) sector, Logical Boot Sector, File Allocation Table, Root Directory Area, placed in the internal flash memory (NVM). Also, a SD card is used as another disk, which might have FAT16 or FAT32 implemented on it (depends on the formatting of SD card).The demonstration reads the contents of "FILE.TXT" in NVM and copies the contents to "FILE.TXT" in the SDTM Card. If the write operation is successful, LED indication is provided.

3.2.4.4.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Following board is supported for PIC32MX devices: -

Demonstration Board (click link for board information)
PIC32MX795F512L Plug-In-Module (PIM)
PICtail Daughter Board for SD and MMC

Note: This board can not be used by itself. An [Explorer 16 Development Board](#) is required.

Following board is supported for PIC32MZ devices: -

Demonstration Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

Note: This board can not be used by itself. An [Multimedia Expansion Board II \(MEB II\)](#) is required to run the demonstration.

3.2.4.4.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

Explorer 16 Development Board Based Demonstrations

Use the following instructions for all Explorer 16 Development Board-based demonstration boards:

PIC32MX795F512L PIM

1. Insert the PIM into the Explorer 16 Development Board PIM connector.
2. Jumpers JP1, JP2 and JP3 on the SD Card PICTail Daughter Board should **connect** to points 2 and 3 on their respective connectors as shown [here](#).
3. Insert the SD Card PICTail Daughter Board into the PICTail Plus connector on the Explorer 16 Development Board.
4. Insert a SD card into the SD card connector on the SD Card PICTail Daughter Board.
5. Power up the board.

Multimedia Expansion Board II (MEB II)

No hardware setting change is required. Insert the micro SD card into the connector and power up the board.

3.2.4.4.3 Running the Demonstration

Provides instructions on how to build and run the NVM SD card FAT multi disk demonstration.

Description

Select the MPLAB X IDE project configuration: -

- exp16_32mx_int_dyn -- For PIC32MX devices
- pic32mz_sk_int_dyn -- For PIC32MZ devices

Insert the SD card, which contains the file "FILE.TXT" and the file contains "abc" (some arbitrary existing content).

Compile the selected configuration in the MPLAB X IDE project and run the code. After a few seconds, once the LED glows, remove the SD card from the Board and insert it into the SD card reader on a PC. Examine the contents of the file named "FILE.TXT". The file should contain the string "This data is from NVM Disk".

Demonstration Board	Demonstration Successful	Demonstration Failure
Explorer 16 Development Board	LED5 (D5)	LED6 (D6)
PIC32MZ Embedded Connectivity (EC) Starter Kit	Green LED	Red LED

About the demonstration:

This demonstration shows an example of:

- implementing a multi disk demonstration with FAT12 on internal flash memory (NVM) and FAT File system (FAT16/ FAT32) on SD card.
- opening two files from two different disks, read content of file from first disk and write into the file of second disk.
- closing the files.

The demonstration contains a FAT12 disk image consisting of a Master Boot Record (MBR) sector, Logical Boot Sector, File Allocation Table, Root Directory Area, placed in the internal flash memory (NVM). Also, a SD card is used as another disk, which

might have FAT16 or FAT32 implemented on it (depends on the formatting of SD card).The demonstration reads the contents of "FILE.TXT" in NVM and copies the contents to "FILE.TXT" in the SDTM Card. If the write operation is successful, LED indication is provided.

The demonstration application logic is implemented as a state machine in the APP_Tasks() function in the file main.c.

1. The first disks is mounted using the [SYS_FS_Mount\(\)](#) function. The "/dev/nvma1" path instructs the mount command to mount a internal flash volume. The volume is mounted against a FAT type file system and mounted at "/mnt/myDrive1/".
2. If the mount is successful, the second disks is mounted using the [SYS_FS_Mount\(\)](#) function. The "/dev/mmcbk1a1" path instructs the mount command to mount a SD card volume. The volume is mounted against a FAT type file system and mounted at "/mnt/myDrive2/".
3. If the mount is successful, the application opens a file "FILE.TXT" from "/mnt/myDrive1/" for reading and "FILE.TXT" from "/mnt/myDrive2/" for writing with a [SYS_FS_FileOpen\(\)](#) function.
4. If the file open is successful, the application reads 27Bytes from "FILE.TXT" of internal flash volume using [SYS_FS_FileRead\(\)](#).
5. If the read is successful, the application closes the "FILE.TXT" from internal flash volume and then writes the 27Bytes into "FILE.TXT" of SD card volume using [SYS_FS_FileWrite\(\)](#).
6. If the write is successful, the application closes the "FILE.TXT" from SD card volume.
7. If file close is successful, LED indicates the success of the operation.

3.2.4.5 nmv_mpfs_single_disk

This demonstration uses a MPFS image of 3 files on NVM flash memory and demonstrates the working of all file system functions.

Description

This demonstration shows an example of implementing a MPFS disk in device Flash memory. The demonstration contains a MPFS disk image in the internal flash memory. The disk image contains 3 files named: -

"FILE.txt" , Size = 11Bytes. The content of the file is -- "Hello World".

"ABC.txt", Size = 31744Bytes. The content of the file is not important. Just a file of sufficiently big size.

"TEST.txt" , Size = 72Bytes. The content of the file is -- "This file contains a test string and it is meant for testing. 1234567890".

The demonstration performs all file system related function calls on the file viz. [SYS_FS_FileRead](#), [SYS_FS_FileStat](#), [SYS_FS_FileSize](#), [SYS_FS_FileSeek](#), [SYS_FS_FileEOF](#). If all the test succeeds, LED indication is provided.

3.2.4.5.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Following board is supported for PIC32MX devices: -

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-

Following board is supported for PIC32MZ devices: -

Demonstration Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.2.4.5.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

3.2.4.5.3 Running the Demonstration

Provides instructions on how to build and run the NVM MPFS single disk demonstration.

Description

Select the MPLAB X IDE project configuration: -

- pic32mx_usb_sk_int_dyn -- For PIC32MX devices
- pic32mz_sk_int_dyn -- For PIC32MZ devices

Build the selected configuration in the MPLAB X IDE project and program the demonstration board. The execution status (pass/fail) of the demonstration is indicated by LEDs on the demo board.

Demonstration Board	Demonstration Successful	Demonstration Failure
PIC32 USB Starter Kit II	LED3	LED1
PIC32MZ Embedded Connectivity (EC) Starter Kit	Green LED	Red LED

About the demonstration:

This demonstration shows an example of

- implementing a MPFS disk in device Flash memory.
- opening a file for read.
- implements all the file system functions.
- closing a file.

This demonstration shows an example of implementing a MPFS disk in device Flash memory. The demonstration contains a MPFS disk image in the internal flash memory. The disk image contains 3 files named: -

"FILE.txt" , Size = 11Bytes. The content of the file is -- "Hello World".

"ABC.txt", Size = 31744Bytes. The content of the file is not important. Just a file of sufficiently big size.

"TEST.txt" , Size = 72Bytes. The content of the file is -- "This file contains a test string and it is meant for testing. 1234567890".

The demonstration application logic is implemented as a state machine in the APP_Tasks() function in the file main.c.

1. The disk is first mounted using the [SYS_FS_Mount\(\)](#) function. The "/dev/nvma1" path instructs the mount command to mount an internal flash volume. The volume is mounted against a MPFS2 type file system and mounted at "/mnt/myDrive/".
2. If the mount is successful, the application opens a file "FILE.txt" for reading with a [SYS_FS_FileOpen\(\)](#) function.
3. If the open is successful, the application opens another file "TEST.txt" for reading with [SYS_FS_FileOpen\(\)](#) function.
4. If open is successful, the application checks the status of another file "ABC.txt" using [SYS_FS_FileStat\(\)](#) function and stores the status of the file into the structure "appData.fileStatus".
5. If file status check is successful, the application compares the size of file (from "appData.fileStatus" structure) with the known value of 31744 (Bytes).
6. If the comparison is successful, the application checks for size of file "FILE.txt", by passing the handle obtained during file open, to the function [SYS_FS_FileSize\(\)](#).
7. If the file size matches the known value of 11 (Bytes), the application moves the file pointer for the file "TEST.txt" 10Bytes from the end of file, using the function [SYS_FS_FileSeek\(\)](#).
8. If file seek is successful, 10 Bytes of content of the file "TEST.txt" is read into the application buffer, using the function [SYS_FS_FileRead\(\)](#).
9. If read is successful, the application buffer content is compared with the known string of "1234567890" using the strcmp() function.
10. If the comparison is successful, the application checks if the file pointer for file "TEST.txt" has reached the end of file using the [SYS_FS_FileEOF\(\)](#) function.
11. If end of file is reached, a LED indicates the success of the demonstration.

3.2.4.6 nvm_sdcard_fat_mpfs_multi_disk

This demonstration uses NVM and SDTM Card with MPFS and FAT image of file and performs a read/ write/ verify operation from file of one media to another media.

Description

This demonstration shows an example of using the MPLAB Harmony File System to access files across multiple disks and multiple file system. The demonstration contains a MPFS disk image in the internal flash memory. The disk image contains a file named "abc.txt" with content "Hello World". Another disk is a SD card, which is formatted to FAT (FAT16 or FAT32). The demonstration reads the contents of "abc.txt" from the disk implemented on internal flash memory and writes the contents to "FILE.TXT" on the SDTM Card. A successful write is indicated by a LED glow.

3.2.4.6.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Following board is supported for PIC32MX devices: -

Demonstration Board (click link for board information)
PIC32MX795F512L Plug-In-Module (PIM)
PICtail Daughter Board for SD and MMC

Note: This board can not be used by itself. An [Explorer 16 Development Board](#) is required.

Following board is supported for PIC32MZ devices: -

Demonstration Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

Note: This board can not be used by itself. An [Multimedia Expansion Board II \(MEB II\)](#) is required to run the demonstration.

3.2.4.6.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[Explorer 16 Development Board Based Demonstrations](#)

Use the following instructions for all Explorer 16 Development Board-based demonstration boards:

PIC32MX795F512L PIM

1. Insert the PIM into the Explorer 16 Development Board PIM connector.
2. Jumpers JP1, JP2 and JP3 on the SD Card PICTail Daughter Board should [connect](#) to points 2 and 3 on their respective connectors as shown [here](#).
3. Insert the SD Card PICTail Daughter Board into the PICTail Plus connector on the Explorer 16 Development Board.
4. Insert a SD card into the SD card connector on the SD Card PICTail Daughter Board.
5. Power up the board.

[Multimedia Expansion Board II \(MEB II\)](#)

No hardware setting change is required. Insert the micro SD card into the connector and power up the board.

3.2.4.6.3 Running the Demonstration

Provides instructions on how to build and run the NVM SD card FAT MPFS multi disk demonstration.

Description

Select the MPLAB X IDE project configuration: -

- exp16_32mx_int_dyn -- For PIC32MX devices
- pic32mz_sk_int_dyn -- For PIC32MZ devices

Insert the SD card, which contains the file "FILE.TXT" and the file contains "abc" (some arbitrary existing content).

Compile the selected configuration in the MPLAB X IDE project and run the code. After a few seconds, once the LED glows, remove the SD card from the SD Card PICTail Daughter Board and insert it into the SD card reader on a PC. Examine the contents of the file named "FILE.TXT". The file should contain the string "Hello World".

Demonstration Board	Demonstration Successful	Demonstration Failure
Explorer 16 Development Board	LED5 (D5)	LED6 (D6)
PIC32MZ Embedded Connectivity (EC) Starter Kit	Green LED	Red LED

About the demonstration:

This demonstration shows an example of

- implementing a multi disk demonstration with MPFS on internal flash memory (NVM) and FAT File system (FAT16/ FAT32) on

SD card.

- opening two files from two different disks, read content of file from first disk and write into the file of second disk.
- closing the files.

The demonstration contains a MPFS disk image in the internal flash memory. The disk image contains a file named "abc.txt" with content "Hello World". Another disk is a SD card, which is formatted to FAT (FAT16 or FAT32). The demonstration reads the contents of "abc.txt" from the disk implemented on internal flash memory and writes the contents to "FILE.TXT" on the SDTM Card. A successful write is indicated by a LED glow.

The demonstration application logic is implemented as a state machine in the APP_Tasks() function in the file main.c.

1. The first disks is mounted using the [SYS_FS_Mount\(\)](#) function. The "/dev/nvma1" path instructs the mount command to mount a internal flash volume. The volume is mounted against a MPFS2 type file system and mounted at "/mnt/myDrive1".
2. If the mount is successful, the second disks is mounted using the [SYS_FS_Mount\(\)](#) function. The "/dev/mmcblk1" path instructs the mount command to mount a SD card volume. The volume is mounted against a FAT type file system and mounted at "/mnt/myDrive2".
3. If the mount is successful, the application opens a file "abc.txt" from "/mnt/myDrive1/" for reading and "FILE.TXT" from "/mnt/myDrive2/" for writing with a [SYS_FS_FileOpen\(\)](#) function.
4. If the file open is successful, the application reads 13Bytes from "abc.txt" of internal flash volume using [SYS_FS_FileRead\(\)](#).
5. If the read is successful, the application closes the "abc.txt" from internal flash volume and then writes the 13Bytes into "FILE.TXT" of SD card volume using [SYS_FS_FileWrite\(\)](#).
6. If the write is successful, the application closes the "FILE.TXT" from SD card volume.
7. If file close is successful, LED indicates the success of the operation.

3.2.5 Demonstration Board Information

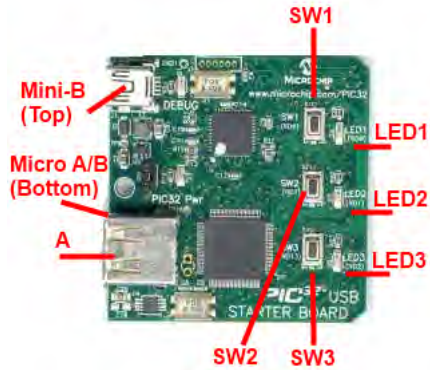
Information on Hardware used by the File System Demonstration Applications.

Description

The File System Library Demonstration Applications can be tried across different hardware boards. This section provide details on these hardware boards.

3.2.5.1 PIC32 USB Starter Kit II

Hardware Information:



SW1 - Application switch. Tied to RD6.

SW2 - Application switch. Tied to RD7.

SW3 - Application switch. Tied to RD13.

LED1 - Application LED. Tied to RD0.

LED2 - Application LED. Tied to RD1.

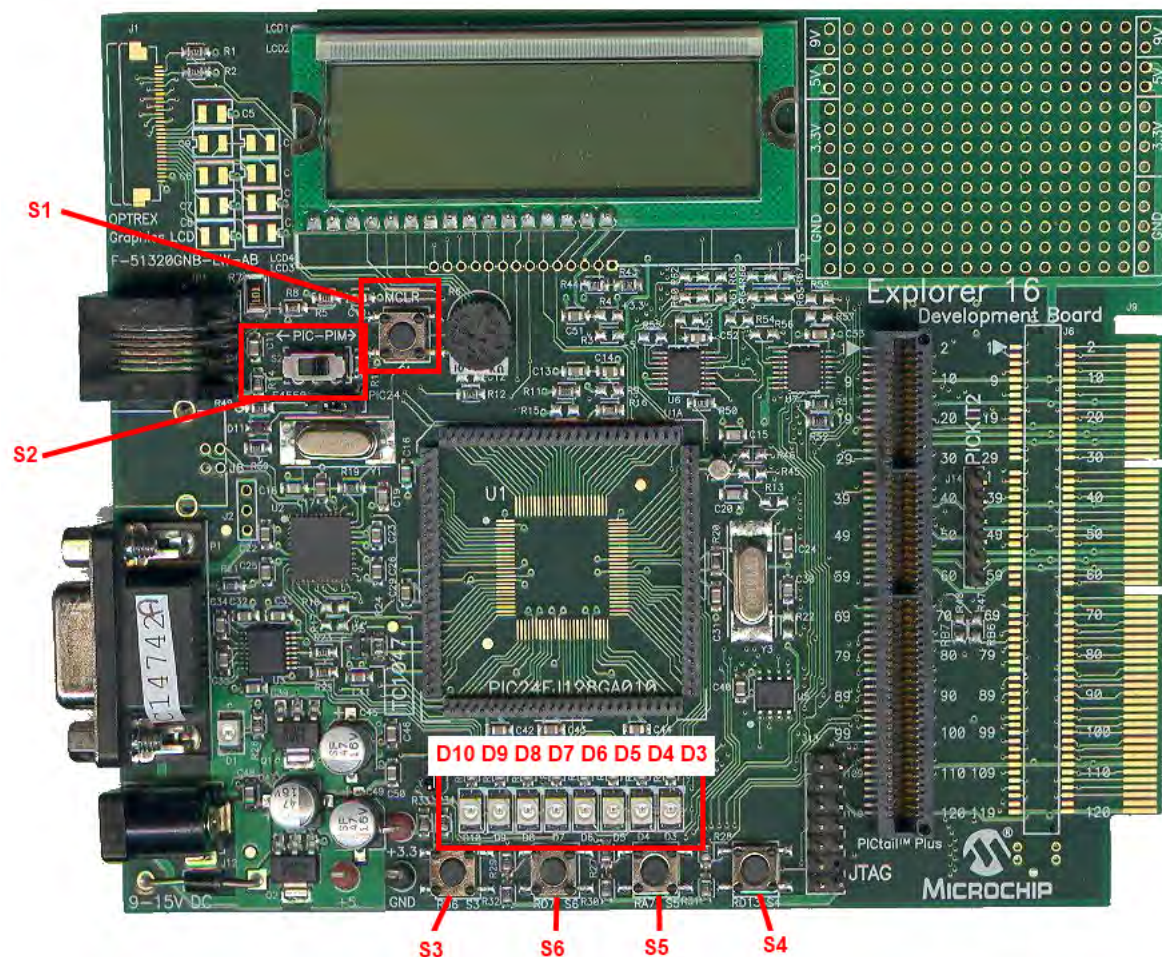
LED3 - Application LED. Tied to RD2.

More Information

[Product webpage](#)

3.2.5.2 Explorer 16 Development Board

Hardware Information:



S1 - Reset button (MCLR)

S2 - Processor switch. This switch determines which processor is running, the processor on the board or the processor on the Plug-In-Module (PIM).

S3, S4, S5, S6 - Application switches. For information about what pin is connected to this switch, please refer to the information for the PIM in use.

D3 through D10 - Application LEDs. For information about what pin is connected to this LED, please refer to the information for the PIM in use.

More Information:

[Product webpage](#)

3.2.5.3 PIC32MX795F512 Plug-In-Module (PIM)

More Information

[Product webpage](#)

3.2.5.4 PICtail Daughter Board for SD and MMC

More Information:

[Product webpage.](#)

3.2.5.5 PIC32MZ Embedded Connectivity (EC) Starter Kit

Provides details information on the PIC32MZ Embedded Connectivity (EC) Starter Kit.

Description

Microchip Part Number: DM320006 (without Crypto) or DM320006-C (with Crypto)

The top assembly of the board includes these key features, as indicated in Figure 1:

1. PIC32MZ2048ECH144 32-bit microcontroller.
2. Green power indicator LED.
3. On-board crystal or oscillator for precision microcontroller clocking (12 MHz).
4. USB connectivity for on-board debugger communications.
5. Green debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 host-based applications.
9. HOST mode power jumper.
10. Daughter board connectors for flexible Ethernet PHY options.
11. 32 kHz oscillator for RTCC and Timer1 (optional).
12. External 2 GB SQI memory for expanded memory application.
13. Jumper for using or disconnecting the on-board debugger.

NOTE: When running self-powered USB device applications, open the jumper JP1 to prevent possibly back-feeding voltage onto the Vbus from one port on the host to another (or from one host to another).

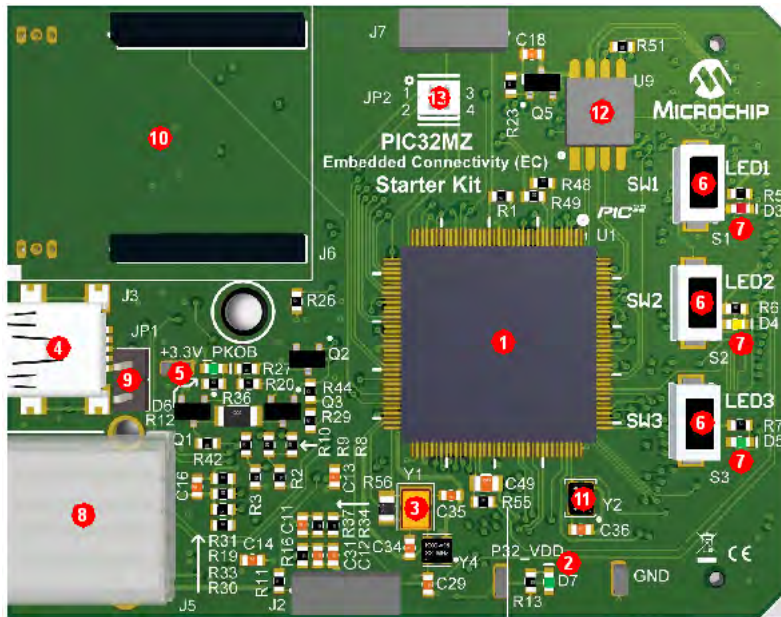


Figure 1: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Front)

The bottom assembly of the board includes these key features, as indicated in Figure 2:

1. PIC24FJ256GB106 USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Type micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.
5. 50 MHz Ethernet PHY oscillator.
6. USB Host and OTG power supply for powering PIC32 USB applications.

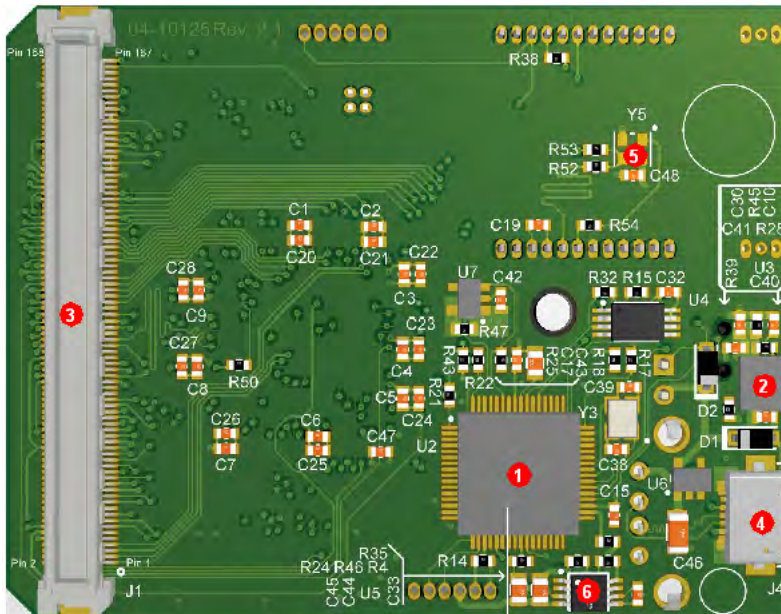


Figure 2: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Back)

3.2.5.6 Multimedia Expansion Board II (MEB II)

Provides details information on the Multimedia Expansion Board II (MEB) II.

Description

Microchip Part Number: DM320005-2

[Product Page](#)

The front side of the MEB II includes these key features, as shown in Figure 1:

1. Display daughter board connector (60-pin Hirose board-to-board connector).
2. mTouch™ buttons.
3. Push Button.
4. Power LED.
5. User LEDs.
6. VGA Camera (OVM7690).
7. PICtail™ Connector.
8. Analog temperature sensor.

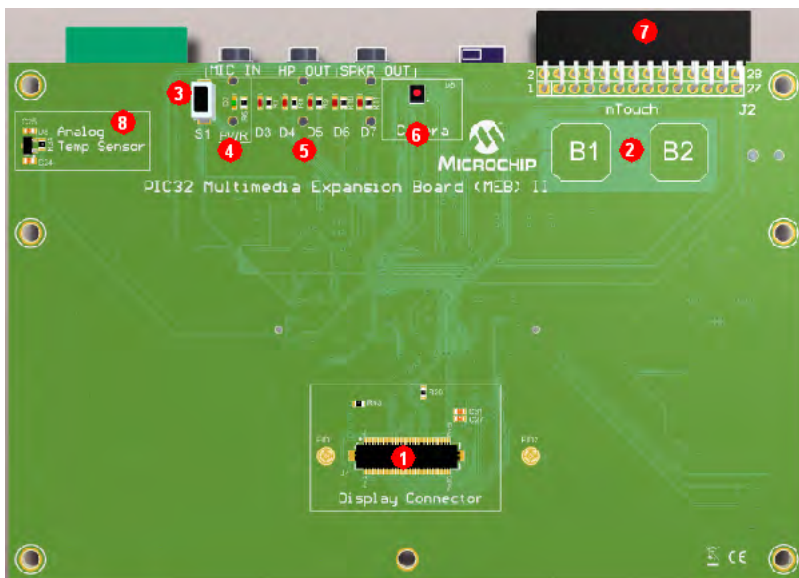


Figure 1: Multimedia Expansion Board II Layout (Front)

The underside of the MEB II includes these key features, as shown in Figure 2:

1. Regulated 5V and 3.3V power supply for powering the board via a 9-15V DC Adapter.
2. PIC32 Starter Kit connector (168-pin Hirose board-to-board connector).
3. 24-bit stereo audio codec (AK4953A).
4. Integrated 802.11bg wireless module (MRF24WG0MA).
5. Low-cost Bluetooth® HCI transceiver (BTM805).
6. EBI SRAM memory (IS61WV102416BLL).
7. microSD slot.

8. Analog accelerometer (ADXL325).

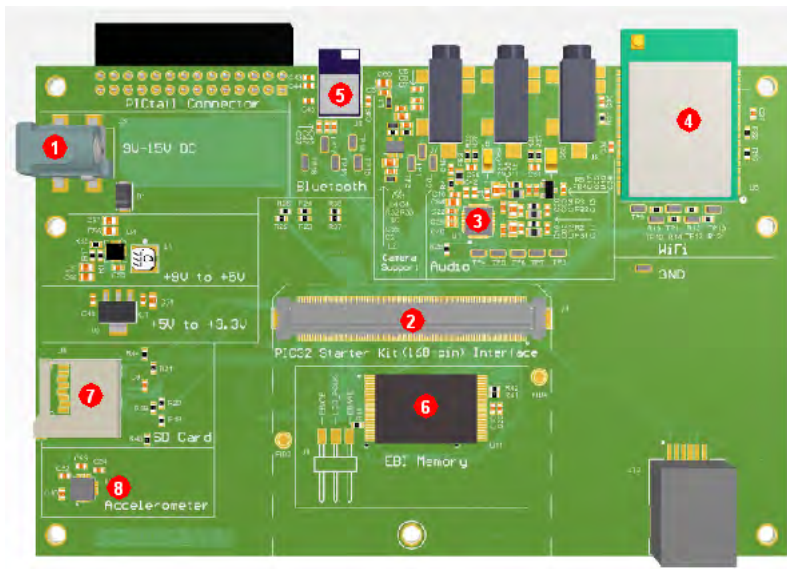


Figure 2: Multimedia Expansion Board II Layout (Back)

3.3 Graphics Demonstrations

3.3.1 Introduction

Graphics Library Demonstrations Applications Help

Description

This distribution package contains a variety of Graphics-related firmware projects that demonstrate the capabilities of the MPLAB Harmony Graphics library. This help file describes the hardware requirement and procedures to run these firmware projects on Microchip graphics boards.

To know more about MPLAB Harmony Graphics, configuring the library and the APIs provided; refer to the [Graphics Library](#) documentation.

Graphic Demonstration	Description
Primitive	Cyclically demonstrates various types of shapes and images, which include circles, lines, and rectangles that can be created from the Graphics Library.
Object	Touch interactive demonstration of various pages of objects that can be created from the Graphics Library.
ssd1926	Touch interactive demonstration of JPEG decoded images from an SD card utilizing the JPEG decoder of the Solomon Systech SSD1926 graphics controller.
lcc	Demonstrates the advanced capabilities of the Graphics Library utilizing the software graphics controller.

3.3.2 Release Notes

MPLAB Harmony Version: v0.70b **Graphics Demonstrations Version :** 4.00b **Release Date:** 18Nov2013

New This Release:

Nothing to report for this release.

Known Issues:

LCC Demonstration: For the PIC32MZ Embedded Connectivity (EC) Starter Kit, after programming, the application may not display the UI folders as expected. If this occurs, to activate the UI display, touch the screen in the folder icons area. This issue will be addressed in the next release of MPLAB Harmony.

3.3.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

3.3.4 Demonstrations

This topic provides information on how to run the Graphics Library demonstration applications included in this release.

Description

All demonstrations provided should be built using the MPLAB X IDE development environment tool supplied by Microchip.

The information that follows, communicate the general steps to building the graphics demos.

- Using MPLAB X IDE, click the File drop-down, select "Open Project...". Navigate to your Harmony installation folder. The default location on Windows will be: C:\Microchip\Harmony\<version>\apps\gfx.
- In the \app\gfx folder there will be the object, primitive, lcc, and ssd1926 demonstration applications.
- Within each demonstration folder, there will be a *firmware* folder. In the firmware folder will exist the MPLAB X IDE project specific to each demonstration.
- Select the project and click "Open Project"
- To configure the project specific to a preset board configuration, right click on the project and select "Set Configuration". This will set the build configuration to match the supported board configuration.

3.3.4.1 lcc_demo

Demonstrates the advanced capabilities of the Graphics Library utilizing the software graphics controller.

Description

This demonstration provides the ability to display alpha blend and gradient colors through the software graphics controller. The demonstration renders four folders for alpha blend, gradient, picture-in-picture (PIP), and performance for all PIC32 devices.

For PIC32MX devices, the demonstration runs on the PICtail Plus LCC Daughter Board, and on PIC32MZ devices, it runs on the Multimedia Expansion Board II (MEB II).

3.3.4.1.1 Supported Demonstration Boards

Lists supported demo and development hardware boards.

Description

Demo Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
Graphics PICTail LCC Board AC164144	-
WQVGA 480x272 Board AC164127-6	-
Multimedia Expansion Board II (MEB II)	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.3.4.1.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[Multimedia Expansion Board II \(MEB II\)](#)

The MEB II has two memory options: External or Internal. This demonstration requires External EBI memory, which is configured using the following J9 jumper settings:

- EBIOE and LCD_PCLK (jumper is closed)
- EBIWE (jumper is open)

3.3.4.1.3 Running the Demonstration

Provides instructions on how to build and run the LCC demo.

Description

Select one of the following build configurations to build the LCC demo:

- pic32mxusbsk_pictailcc_wqvga -- For PIC32MX devices
- pic32mzsk_meb2_wqvga -- For PIC32MZ devices

Build the selected configuration in the MPLAB X IDE project and program the demonstration board. The demonstration renders four folders for alpha blend, gradient, picture-in-picture (PIP), and performance. Touch one of the folders to activate the demonstration.

Short Name	Long Name
pic32mxusbsk	PIC32 USB Starter Kit II
pic32mzsk	PIC32MZ Embedded Connectivity (EC) Starter Kit
meb2	Multimedia Expansion Board II (MEB II)

wqvga	Graphics Display Powertip 4.3" 480 × 272 Board (AC164127-6)

3.3.4.2 primitive_demo

An example of primitive drawing capabilities of the Graphics Library.

Description

The Primitive example demonstrates the various types of shapes and images which include: circles, lines, rectangles that exist in the Graphics Library.

This example shows the following primitives:

- [GFX_LineDraw](#)
- [GFX_CircleDraw](#)
- [GFX_CircleFillDraw](#)
- [GFX_RectangleRoundDraw](#)
- [GFX_RectangleRoundFillDraw](#)
- [GFX_RectangleFillDraw](#)
- [GFX_ArcDraw](#)
- [GFX_PolygonDraw](#)
- [GFX_FontSet](#)
- [GFX_TextStringXYPut](#)
- [GFX_ImageDraw](#)
- [GFX_ScreenClear](#)

3.3.4.2.1 Supported Demonstration Boards

Lists supported demo and development hardware boards.

Description

Following boards are supported for PIC32MX devices:

Demo Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
Graphics LCD Controller PICtail Plus SSD1926 Board AC164127-5	-
QVGA 320x240 Board AC164127-4	-

Following board is supported for PIC32MZ devices:

Demo Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-
Multimedia Expansion Board II (MEB II)	-

3.3.4.2.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

3.3.4.2.3 Running the Demonstration

Provides instructions on how to build and run the primitive demo.

Description

Navigate to the apps/gfx/primitive/firmware folder and open the primitive project, and then select one of the following build configurations to build the Primitive demo:

- pic32mxusbsk_pictails1d_wqvga -- For PIC32MX devices
- pic32mxusbsk_pictailssd_qvga -- For PIC32MX devices
- pic32mzsk_meb2_wqvga -- For PIC32MZ devices

Once the project is loaded and executed on the target board, a continuous display of primitive graphics will appear on the LCD.

Short Name	Long Name
pic32mxusbsk	PIC32 USB Starter Kit II
pic32mzsk	PIC32MZ Embedded Connectivity (EC) Starter Kit
pictails1d	Graphics LCD Controller PICTail Plus S1D1357 Board
pictailssd	Graphics LCD Controller PICTail Plus SSD1926 Board
meb2	Multimedia Expansion Board II (MEB II)
qvga	Graphics Display Truly 3.2" 240 × 320 Board (AC164127-4)
wqvga	Graphics Display Powertip 4.3" 480 × 272 Board (AC164127-6)

3.3.4.3 object_demo

An example of standard user-interface widgets of the Graphics Library.

Description

The Object example demonstrates the standard types of objects/widgets that exist in the Graphics Library Graphics Object Layer (GOL). The demonstration is composed of several pages of examples.

The demonstration requires that the user touch the screen to begin the demo. The user is expected to navigate to the next or previous page by selecting the buttons on the far right and left locations of the screen.

This example shows the following widgets:

- [GFX_GOL_ButtonCreate](#)

- [GFX_GOL_ButtonTextSet](#)
- [GFX_GOL_CheckboxCreate](#)
- [GFX_GOL_GroupboxCreate](#)
- [GFX_GOL_RadioButtonCreate](#)
- [GFX_GOL_GroupboxCreate](#)
- [GFX_GOL_StaticTextCreate](#)
- [GFX_GOL_ScrollBarCreate](#)
- [GFX_GOL_ProgressBarCreate](#)
- [GFX_GOL_ListboxCreate](#)
- [GFX_GOL_ScrollBarCreate](#)
- [GFX_GOL_Editbox_Create](#)
- [GFX_GOL_MeterCreate](#)
- [GFX_GOL_DigitalMeterCreate](#)
- [GFX_GOL_RoundDialCreate](#)
- [GFX_GOL_PictureCreate](#)
- [GFX_GOL_CustomControlCreate](#)

3.3.4.3.1 Supported Demonstration Boards

Lists supported demo and development hardware boards.

Description

Following boards are supported for PIC32MX devices:

Demo Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
Graphics LCD Controller PICtail Plus SSD1926 Board AC164127-5	-
QVGA 320x240 Board AC164127-4	-

Following board is supported for PIC32MZ devices:

Demo Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-
Multimedia Expansion Board II (MEB II)	-

3.3.4.3.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

3.3.4.3.3 Running the Demonstration

Provides instructions on how to build and run the object demo.

Description

Navigate to the apps/gfx/object/firmware and open the object project, and then select one of the following build configurations to build the Object demo:

- pic32mxusbsk_pictailssd_qvga -- For PIC32MX devices
- pic32mzsk_meb2_wqvga -- For PIC32MZ devices

Once the project is loaded and executed on the target board, various objects are displayed. Touch the forward or back arrow to view additional objects.

Short Name	Long Name
pic32mxusbsk	PIC32 USB Starter Kit II
pic32mzsk	PIC32MZ Embedded Connectivity (EC) Starter Kit
pictailssd	Graphics LCD Controller PICtail Plus SSD1926 Board
meb2	Multimedia Expansion Board II (MEB II)
qvga	Graphics Display Truly 3.2" 240 × 320 Board (AC164127-4)

3.3.4.4 ssd_demo

Shows how to display JPEG images using the Solomon Systech (SSD1926) graphics controller that resides on the PICtail™ Plus SSD1926 Board.

Description

The SSD1926 demo shows how to decode of JPEG images from an SD card utilizing the JPEG decoder capabilities of the Solmon Systech SSD1926 graphics controller.

3.3.4.4.1 Supported Demonstration Boards

Lists supported demo and development hardware boards.

Description

Following board is supported for PIC32MX devices:

Demo Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
Graphics LCD Controller PICtail Plus SSD1926 Board AC164127-5	-
QVGA 320x240 Board AC164127-4	-

3.3.4.4.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

3.3.4.4.3 Running the Demonstration

Provides instructions on how to build and run the SSD1926 demo.

Description

Navigate to the apps/gfx/ssd1926/firmware and open the ssd1926 project, and then select one of the following build configurations to build the SSD1926 demo:

- pic32mxusbsk_pictailssd_qvga -- For PIC32MX devices

Once the project is loaded and executed on the target board, various JPEG images, which are stored on a SD card are displayed.

Short Name	Long Name
pic32mxusbsk	PIC32 USB Starter Kit II
pictailssd	Graphics LCD Controller PICtail Plus SSD13517 Board
qvga	Graphics Display Truly 3.2" 240 × 320 Board (AC164127-4)

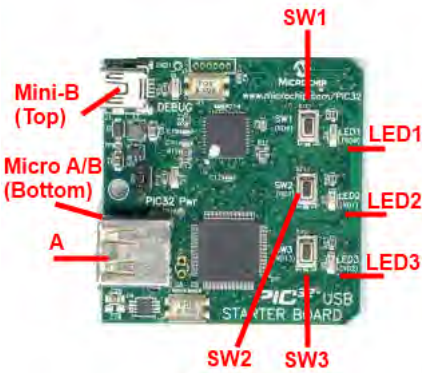
3.3.5 Demonstration Board Information

This topic provides specific information about the development boards available for the demonstration applications.

3.3.5.1 PIC32 USB Starter Kit II

Provides information on the PIC32 USB Starter II.

Description



SW1 - Application switch. Tied to RD6.

SW2 - Application switch. Tied to RD7.

SW3 - Application switch. Tied to RD13.

LED1 - Application LED. Tied to RD0.

LED2 - Application LED. Tied to RD1.

LED3 - Application LED. Tied to RD2.

This board supports the following USB Device demonstrations:

- CDC Single COM port ([cdc_com_port_single](#)).
- CDC Dual COM port ([cdc_com_port_dual](#))
- HID Basic ([hid_basic](#))
- HID Mouse ([hid_mouse](#))
- HID Keyboard ([hid_keyboard](#))
- HID Joystick ([hid_joystick](#))
- MSD Flash Drive ([msd_basic](#))
- Generic USB Device ([generic_device](#))

More Information

Microchip Part Number : DM320003-2

3.3.5.2 PIC32MZ Embedded Connectivity (EC) Starter Kit

Provides details information on the PIC32MZ Embedded Connectivity (EC) Starter Kit.

Description

Microchip Part Number: DM320006 (without Crypto) or DM320006-C (with Crypto)

The top assembly of the board includes these key features, as indicated in Figure 1:

1. PIC32MZ2048ECH144 32-bit microcontroller.
2. Green power indicator LED.
3. On-board crystal or oscillator for precision microcontroller clocking (12 MHz).
4. USB connectivity for on-board debugger communications.
5. Green debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 host-based applications.
9. HOST mode power jumper.
10. Daughter board connectors for flexible Ethernet PHY options.
11. 32 kHz oscillator for RTCC and Timer1 (optional).
12. External 2 GB SQI memory for expanded memory application.
13. Jumper for using or disconnecting the on-board debugger.

NOTE: When running self-powered USB device applications, open the jumper JP1 to prevent possibly back-feeding voltage onto the Vbus from one port on the host to another (or from one host to another).

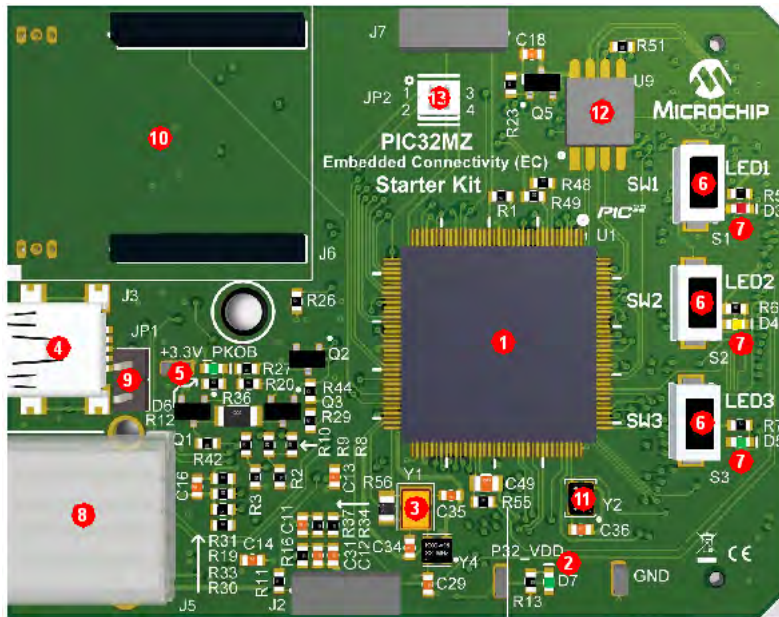


Figure 1: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Front)

The bottom assembly of the board includes these key features, as indicated in Figure 2:

1. PIC24FJ256GB106 USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Type micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.
5. 50 MHz Ethernet PHY oscillator.
6. USB Host and OTG power supply for powering PIC32 USB applications.

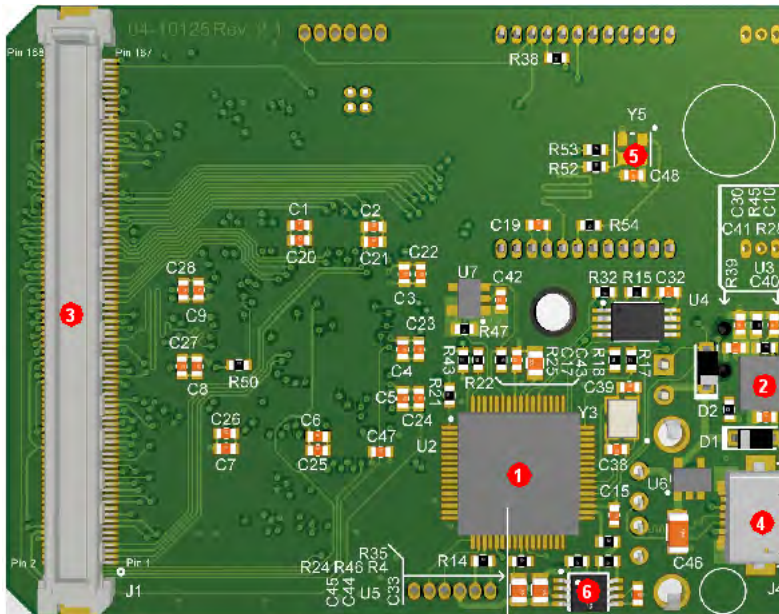


Figure 2: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Back)

Microchip Part Number: DM320006 (without Crypto) or DM320006-C (with Crypto)

The top assembly of the board includes these key features, as indicated in Figure 1:

1. PIC32MZ2048ECH144 32-bit microcontroller.
2. Green power indicator LED.
3. On-board crystal or oscillator for precision microcontroller clocking (12 MHz).
4. USB connectivity for on-board debugger communications.
5. Green debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 host-based applications.
9. HOST mode power jumper.
10. Daughter board connectors for flexible Ethernet PHY options.
11. 32 kHz oscillator for RTCC and Timer1 (optional).
12. External 2 GB SQI memory for expanded memory application.
13. Jumper for using or disconnecting the on-board debugger.

NOTE: When running self-powered USB device applications, open the jumper JP1 to prevent possibly back-feeding voltage onto the Vbus from one port on the host to another (or from one host to another).

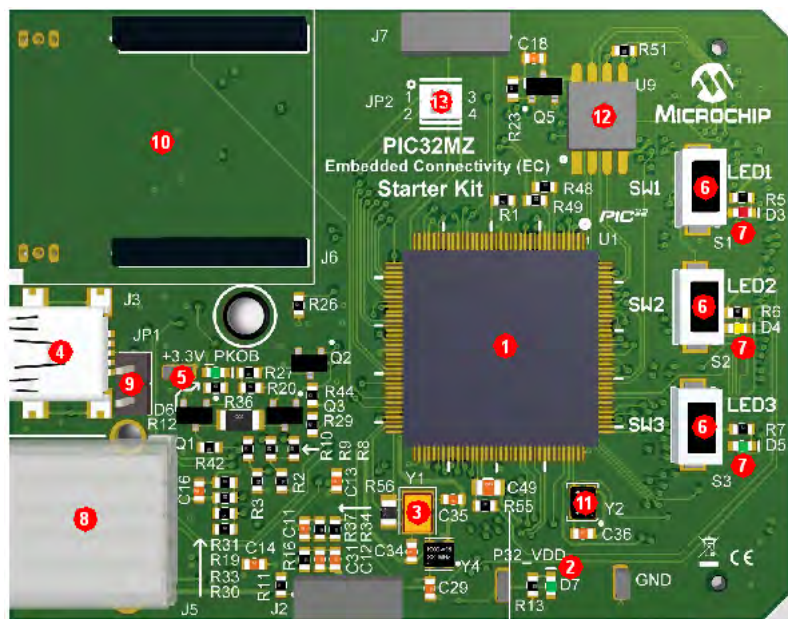


Figure 1: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Front)

The bottom assembly of the board includes these key features, as indicated in Figure 2:

1. PIC24FJ256GB106 USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Type micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.
5. 50 MHz Ethernet PHY oscillator.

6. USB Host and OTG power supply for powering PIC32 USB applications.

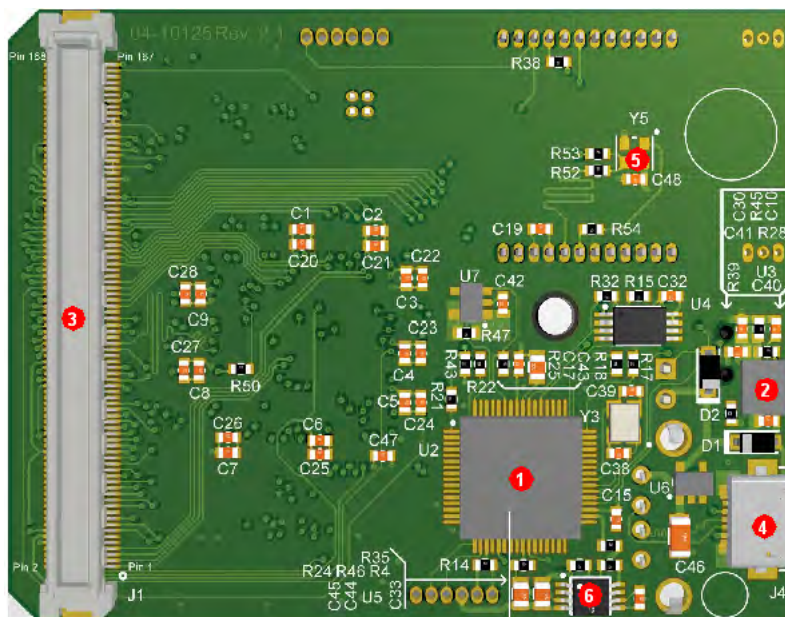


Figure 2: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Back)

3.3.5.3 PICTail Plus SSD1926 Daughter Board

This topic provides information about the Graphics LCD Controller PICTail™ Plus SSD1926 Board.

Description

More Information

Microchip Part Number: AC164127-5

[Microchip Product Page](#)

3.3.5.4 PICTail Plus S1D13517 Daughter Board

This topic provides information about the Graphics LCD Controller PICTail™ Plus SSD1926 Daughter Board.

Description

More Information

Microchip Part Number: AC164127-7

[Microchip Product Page](#)

3.3.5.5 PICTail Plus LCC Daughter Board

This topic provides information about the Graphics LCD Controller PICTail™ Plus LCC Daughter Board.

Description

More Information

Microchip Part Number: AC164144

[Microchip Product Page](#)

3.3.5.6 PIC32MX795F512L Plug-In-Module (PIM)

Provides information on the PIC32MX795F512L Plug-In-Module (PIM).

Description

This PIM is required while using the [Explorer 16 Development Board](#).

More Information

Microchip Part Number: MA320003

[microchipDIRECT Product Page](#)

3.3.5.7 Multimedia Expansion Board (MEB)

This topic provides information about the Multimedia Expansion Board (MEB).

3.3.5.8 Multimedia Expansion Board II (MEB II)

Provides details information on the Multimedia Expansion Board II (MEB) II.

Description

Description

Microchip Part Number: DM320005-2

[Product Page](#)

The front side of the MEB II includes these key features, as shown in Figure 1:

1. Display daughter board connector (60-pin Hirose board-to-board connector).
2. mTouch™ buttons.
3. Push Button.
4. Power LED.
5. User LEDs.
6. VGA Camera (OVM7690).
7. PICTail™ Connector.
8. Analog temperature sensor.

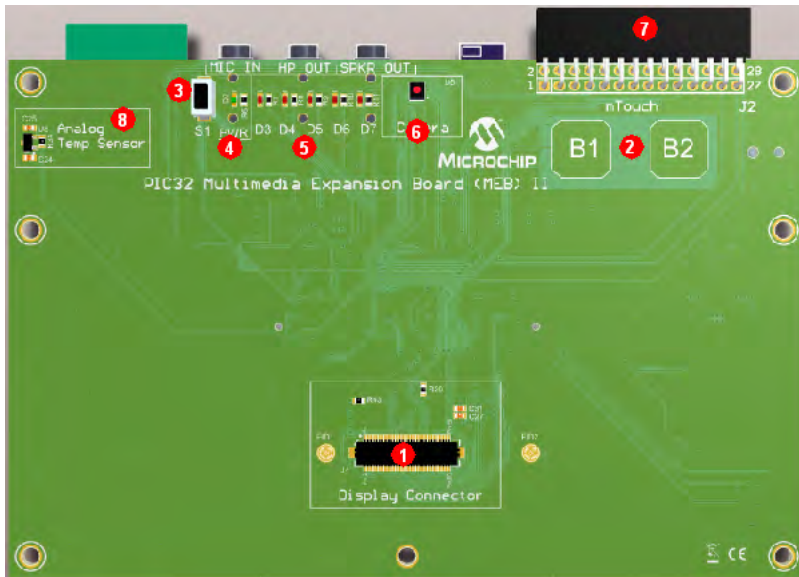


Figure 1: Multimedia Expansion Board II Layout (Front)

The underside of the MEB II includes these key features, as shown in Figure 2:

1. Regulated 5V and 3.3V power supply for powering the board via a 9-15V DC Adapter.
2. PIC32 Starter Kit connector (168-pin Hirose board-to-board connector).
3. 24-bit stereo audio codec (AK4953A).
4. Integrated 802.11bg wireless module (MRF24WG0MA).
5. Low-cost Bluetooth® HCI transceiver (BTM805).
6. EBI SRAM memory (IS61WV102416BLL).
7. microSD slot.
8. Analog accelerometer (ADXL325).

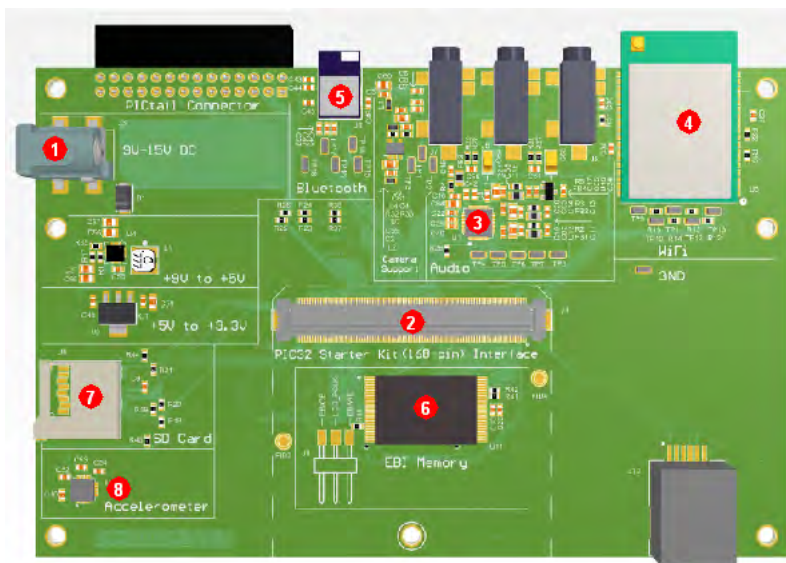


Figure 2: Multimedia Expansion Board II Layout (Back)

3.4 Peripheral Library Example Applications

This topic describes the peripheral library example applications.

3.4.1 Introduction

Peripheral Library Example Applications for MPLAB Harmony

The example applications for MPLAB peripheral libraries (PLIBs) provide very simple single-purpose examples of how to use MPLAB Harmony peripheral libraries.

Description

Peripheral libraries (PLIBs) are the lowest level libraries provided with MPLAB Harmony. They provide a functional abstraction of the peripherals available on Microchip microcontrollers to hide differences in register details that can exist from part to part. However, they do not maintain any state data (at least not any that isn't stored in the hardware registers) from call to call and they do not provide any protection of the peripheral's resources. As such, any code that calls the PLIB for a peripheral must take responsibility for ownership of that peripheral and is responsible for managing the behavior of that peripheral.

As such, PLIBs are normally only used by MPLAB Harmony device drivers or system services. However, there are some times when it is necessary and appropriate to interact with a PLIB directly from an application. So, simple examples are provided to show how to use the interfaces the peripheral libraries. These examples are available in the MPLAB Harmony installation in the following location.

Location of Example Applications

<installation-root>/apps/examples/peripheral

Examples are divided into two different groups:

- Blocking Examples
- State-Driven Examples

Most MPLAB Harmony applications use a state-machine driven structure. So, examples are provided that follow this structure in the "state_driven" subdirectory. However, it is sometimes easier to understand a simple blocking example (one that sits in a loop for ever or until its task is done). So, simple blocking examples are also provided in the "blocking" subdirectory.

Blocking examples are divided into PIC32MX and PIC32MZ variants. PIC32MX examples are written for the Explorer16 Development Board with a PIC32MX795F512L PIM. PIC32MZ examples are written for the PIC32MZ Embedded Connectivity (EC) Starter Kit. The state-driven examples are tested and working on the Explorer 16 Development Board, the PIC32 USB Starter Kit II, and the PIC32MZ Embedded Connectivity (EC) Starter Kit, and the appropriate board configuration for each project can be selected in MPLAB X IDE.

3.4.2 Release Notes

MPLAB Harmony Version: v0.70b **Peripheral Library Example Applications Version:** 0.01a **Release Date:** 18Nov2013

The interface can change in the beta and/or 1.0 release.

New This Release:

Nothing to report for this release.

Known Issues:

Nothing to report for this release.

3.4.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

3.4.4 Blocking Applications

This topic contains a list of the MPLAB Harmony blocking peripheral library (PLIB) example applications.

Description

The tables below list and summarizes the simple blocking peripheral library example applications.

Location

<installation-root>/apps/examples/peripheral/blocking/pic32mx

PIC32MX PLIB-based Applications

Name	Subdirectory	Description
ADC Potentiometer	adc/adc_pot	Repeatedly reads the Explorer 16 Development Board potentiometer and outputs a pattern on the LEDs corresponding to the potentiometer's position.
Memory Partition	bmx/mem_partition	Partitions flash memory and ram. Explorer 16 Development Board LEDs are turned on if the RAM partition sizes are correct and the partitioning completed successfully.
Simple Comparator	cmp/simple_comparator	Toggles the LEDs on the Explorer 16 Development Board, depending on which input of the comparator is at a higher voltage.
Triangle Wave	cvref/triangle_wave	This example will generate a triangular wave on the CVREF output pin.
LED Pattern	dma/led_pattern	Blinks LEDs on the Explorer 16 Development Board using a pattern stored in Flash memory. It uses the DMA controller to transfer data from flash to the I/O port controlling the LEDs.
Master-Slave	i2c/master_slave	The I2C1 module is setup to operate as the master, while the I2C2 module operates as the slave. The master sends a string to the slave, which stores the data into an array.
Simple Input-Capture	ic/simple_input_capture	When a rising edge is output to the Input Capture 1 pin (RD8), it triggers an interrupt, which then stores the captured timer value into the 'CaptureTime' variable.
Flash Modify	nvm/flash_modify	Reads, writes and erases data in the program Flash memory.
PWM Example	oc/oc_pwm	Generates a 40 kHz PWM with a 25% duty cycle on the OC1 output pin.
Oscillator Configuration	osc/osc_config	Demonstrates how to change the system clock source and PLL values during run-time.
LCD Example	pmp/pmp_lcd	Writes two lines of text to the LCD screen on the Explorer 16 Development Board.
Blinky Leds	ports/blinky_leds	Toggles the Explorer 16 Development Board LEDs in an infinite loop.
Change Notice	ports/cn_interrupt	When SW4 (the right most button on Explorer 16 Development Board) is pressed, it triggers a change notice interrupt, which then toggles the Explorer 16 Development Board LEDs.
Sleep Mode	power/sleep_mode	This example demonstrates how to put the device into Sleep mode and then awake the device using the Watchdog Timer.
Reset Handler	reset/reset_handler	This example does checks on various reset flags, assigning each one to an Explorer 16 Development Board LED. If the flag is set, the LED is turned on and the reset flag is cleared.
RTCC Alarm	rtcc/rtcc_alarm	Sets up the RTCC module to function as an alarm, which goes off at 6:00am every morning.
SPI Loopback	spi/spi_loopback	This example assumes that the SPI SDO (output) is connected to the SDI (input). Data is then transferred directly from the output to the input and the Explorer 16 Development Board LEDs are turned on if the data transfer is successful.
Timer Example	tmr/timer2_interrupt	Uses Timer2 in 32-bit mode to generate an interrupt every one second, blinking the LEDs on the Explorer 16 Development Board.
Simple UART	uart/simple_uart	When the Explorer 16 Development Board is connected to a PC with an RS-232 cable, the device will echo what characters the user types into a terminal program (e.g., Putty) and write the corresponding 8-bit character pattern to the Explorer 16 Development Board LEDs.
Simple WDT	wdt/simple_wdt	Demonstrates how to setup the Watchdog Timer and detect whether or not a time-out reset has occurred.

Location

<installation-root>/apps/examples/peripheral/blocking/pic32mz

PIC32MZ PLIB-based Applications

Name	Subdirectory	Description
Triangle Wave	cvref/triangle_wave	This example will generate a triangular wave on the CVref output pin.
LED Pattern	dma/led_pattern	Blinks LEDs on the PIC32MZ Starter-Kit using a pattern stored in flash memory. It uses the DMA controller to transfer data from flash to the I/O port controlling the LEDs.
Master-Slave	i2c/master_slave	The I2C1 module is setup to operate as the master, while the I2C2 module operates as the slave. The master sends a string to the slave, which stores the data into an array.
Flash Modify	nvm/flash_modify	Reads, writes and erases data in the program flash memory.
PWM Example	oc/oc_pwm	Generates a 40KHz PWM with a 25% duty cycle on the OC1 output pin.
Oscillator Configuration	osc/osc_config	Demonstrates how to change the system clock source and PLL values during run-time.
Blinky Leds	ports/blinky_leds	Toggles the PIC32MZ Starter-Kit LEDs in an infinite loop.
Change Notice	ports/cn_interrupt	When SW1 is pressed, it triggers a change notice interrupt, which then toggles the PIC32MZ Starter-Kit LEDs.
Sleep Mode	power/sleep_mode	This example demonstrates how to put the device into sleep mode and then awake the device using the watchdog timer.
Reset Handler	reset/reset_handler	This example does checks on various reset flags, assigning each one to an LED. If the flag is set, the LED is turned on and the reset flag is cleared.
RTCC Alarm	rtcc/rtcc_alarm	Sets up the RTCC module to function as an alarm, which goes off at 6:00am every morning.
SPI Loopback	spi/spi_loopback	This example assumes that the SPI SDO (output) is connected to the SDI (input). Data is then transferred directly from the output to the input and the LEDs are turned on if the data transfer is successful.
Timer Example	tmr/timer2_interrupt	Uses Timer 2 in 32-bit mode to generate an interrupt every one second, blinking the LEDs on the PIC32MZ Starter-Kit.
Simple UART	uart/simple_uart	When the PIC32MZ Starter-Kit is connected to a PC with an RS-232 cable, the device will echo what characters the user types into a terminal program (e.g. Putty).
Simple WDT	wdt/simple_wdt	Demonstrates how to setup the watchdog timer and detect whether or not a timeout reset has occurred.

3.4.5 State-Driven Applications

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

3.5 NVM Driver Demonstration

3.5.1 Introduction

This help file contains instructions and associated information about MPLAB Harmony NVM driver application demonstrations, which are contained in the MPLAB Harmony Library distribution.

Description

This application demonstrates the capabilities of the MPLAB Harmony NVM driver. This help file describes the hardware requirement and procedures to build, run the demo project on Microchip development tools. In this demo application, NVM driver is used to access the internal flash memory of PIC32 MX and PIC32 MZ Devices to perform Write and Read operations and indicates the result by LED display.

To know more about MPLAB Harmony NVM driver, configuring the NVM driver and API's provided by the NVM driver, refer to MPLAB Harmony [NVM Driver Library](#) documentation.

3.5.2 Release Notes

MPLAB Harmony Version: v0.70b **NVM Driver Demonstration Version:** 0.01a **Release Date:** 18Nov2013

New This release:

This is the first release of this NVM Driver demonstration application. This demonstration has been developed to support PIC32MX and PIC32MZ devices.

Known Issues:

- This Demo application is tested only up to 150 bytes in interrupt mode in PIC32MZ devices
- Changes in NVM driver interfaces will have impact on this demo application as well in subsequent releases

3.5.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO

YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

3.5.4 Hardware

This section describes how to select and use an appropriate development board and steps to configure the hardware to run this NVM driver demo.

3.5.4.1 Required Hardware

This section describes the overview of required hardware family and supported demonstration boards for running this NVM driver demo application.

Description

To run this demonstration project, you will require one of the following sets of hardware.

3.5.4.1.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Following board is supported for PIC32MX devices: -

Demo Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
Explorer 16 Development Board	-
PIC32MX795F512L Plug-in Module (PIM)	-

Following board is supported for PIC32MZ devices: -

Demo Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.5.4.1.2 Hardware Information

Information on hardware used by the NVM driver demo application.

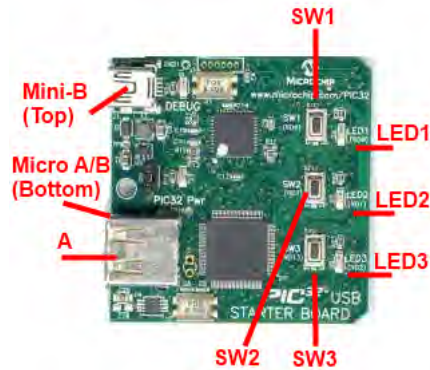
Description

The NVM driver demo application can be tried on different hardware boards. This section provides details on the different hardware boards that this demo application supports.

3.5.4.1.2.1 PIC32 USB Starter Kit II

Provides information on the PIC32 USB Starter Kit II.

Description



SW1 - Application switch. Tied to RD6.

SW2 - Application switch. Tied to RD7.

SW3 - Application switch. Tied to RD13.

LED1 - Application LED. Tied to RD0.

LED2 - Application LED. Tied to RD1.

LED3 - Application LED. Tied to RD2.

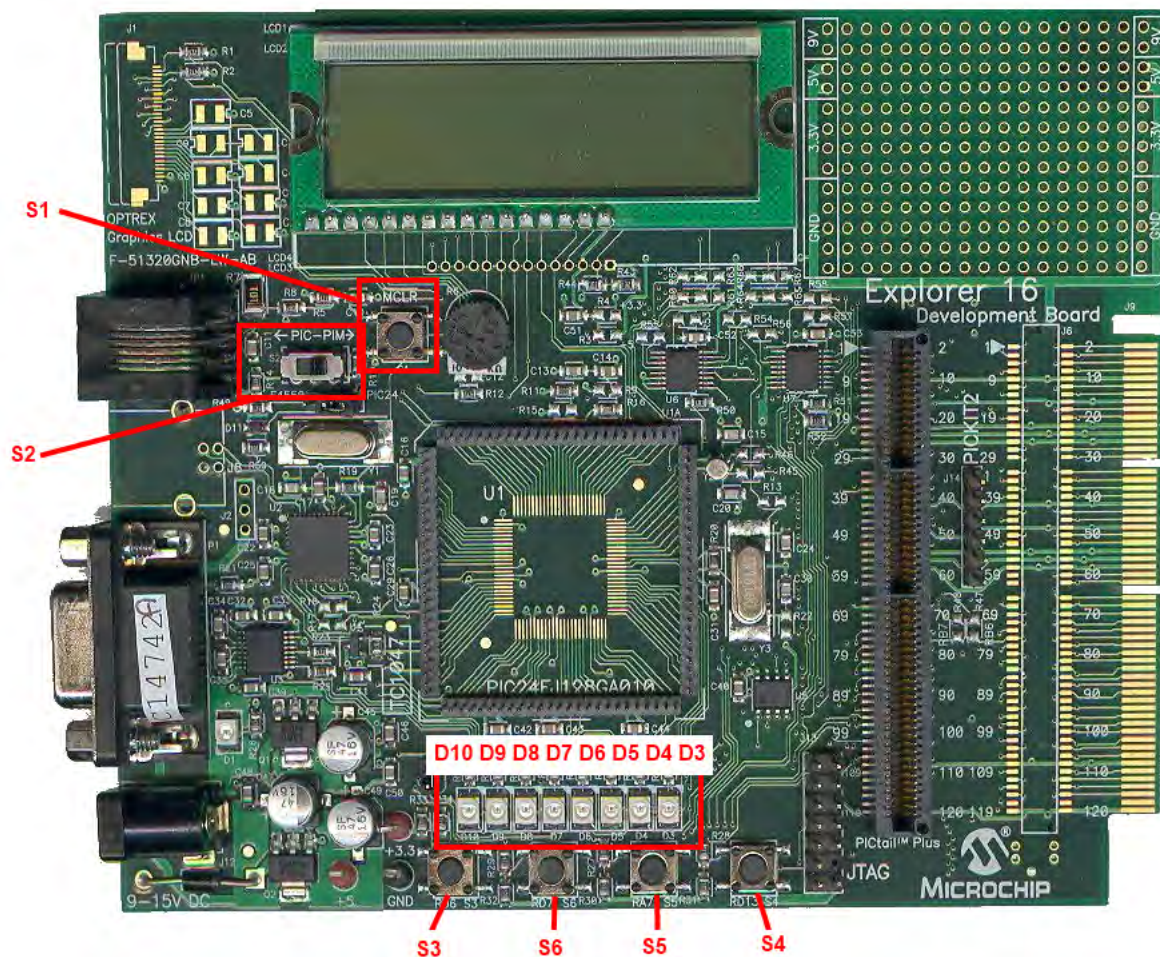
More Information

Microchip Part Number : DM320003-2

3.5.4.1.2.2 Explorer 16 Development Board

Provides information on the Explorer 16 Development Board.

Description



S1 - Reset button (MCLR)

S2 - Processor switch. This switch determines which processor is running, the processor on the board or the processor on the Plug-In-Module (PIM).

S3, S4, S5, S6 - Application switches. For information about what pin is connected to this switch, please refer to the information for the PIM in use.

D3 through D10 - Application LEDs. For information about what pin is connected to this LED, please refer to the information for the PIM in use.

More Information:

Microchip Part Number: DM240001

3.5.4.1.2.3 PIC32MX795F512L Plug-in Module (PIM)

Provides information on the PIC32MX795F512L Plug-In-Module (PIM).

Description

This PIM is required while using the [Explorer 16 Development Board](#).

More Information

Microchip Part Number: MA320003

[microchipDIRECT Product Page](#)

3.5.4.1.2.4 PIC32MZ Embedded Connectivity (EC) Starter Kit

Provides information on the PIC32MZ Embedded Connectivity (EC) Starter Kit.

Description

Microchip Part Number: DM320006 (without Crypto) or DM320006-C (with Crypto)

The top assembly of the board includes these key features, as indicated in Figure 1:

1. PIC32MZ2048ECH144 32-bit microcontroller.
2. Green power indicator LED.
3. On-board crystal or oscillator for precision microcontroller clocking (12 MHz).
4. USB connectivity for on-board debugger communications.
5. Green debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 host-based applications.
9. HOST mode power jumper.
10. Daughter board connectors for flexible Ethernet PHY options.
11. 32 kHz oscillator for RTCC and Timer1 (optional).
12. External 2 GB SQI memory for expanded memory application.
13. Jumper for using or disconnecting the on-board debugger.

NOTE: When running self-powered USB device applications, open the jumper JP1 to prevent possibly back-feeding voltage onto the Vbus from one port on the host to another (or from one host to another).

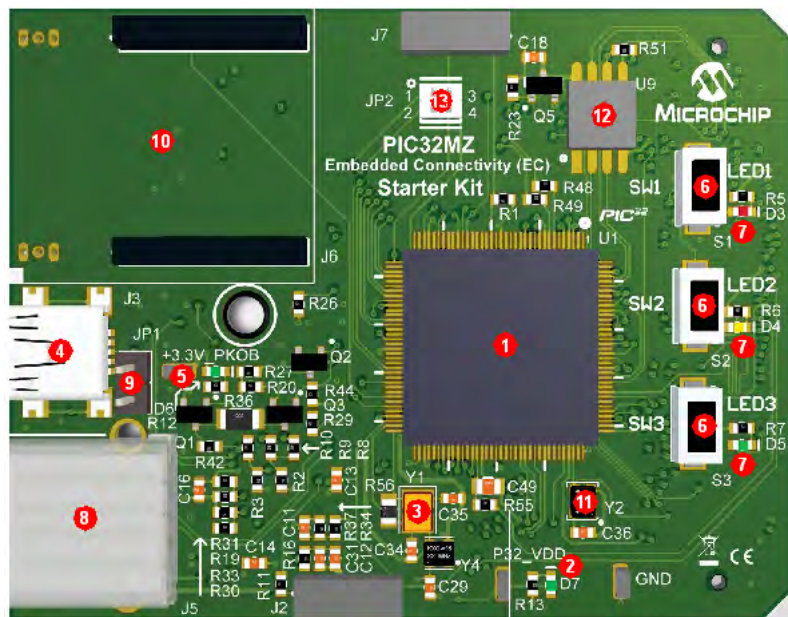
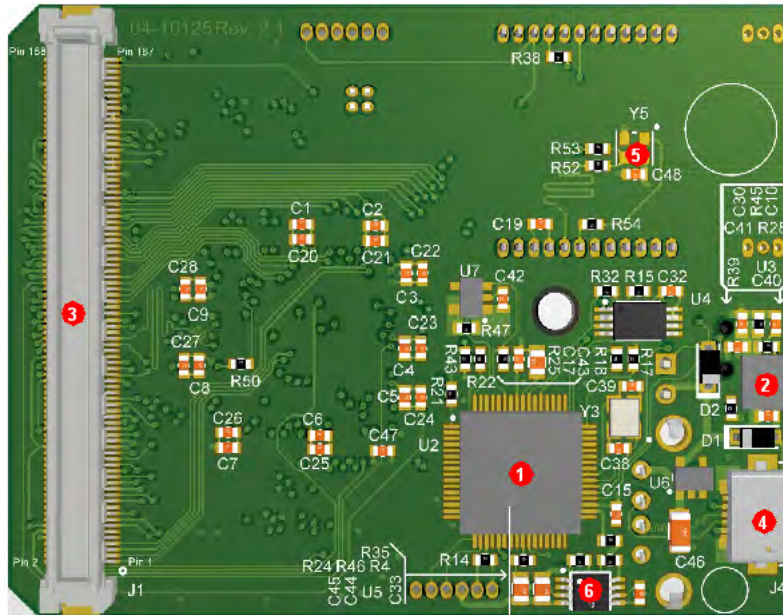


Figure 1: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Front)

The bottom assembly of the board includes these key features, as indicated in Figure 2:

1. PIC24FJ256GB106 USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Type micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.
5. 50 MHz Ethernet PHY oscillator.
6. USB Host and OTG power supply for powering PIC32 USB applications.

**Figure 2: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Back)**

3.5.4.1.2.5 Multimedia Expansion Board II (MEB II)

Provides information on the Multimedia Expansion Board II (MEB II).

Description

Microchip Part Number: DM320005-2

[Product Page](#)

The front side of the MEB II includes these key features, as shown in Figure 1:

1. Display daughter board connector (60-pin Hirose board-to-board connector).
2. mTouch™ buttons.
3. Push Button.
4. Power LED.
5. User LEDs.
6. VGA Camera (OVM7690).
7. PICtail™ Connector.
8. Analog temperature sensor.

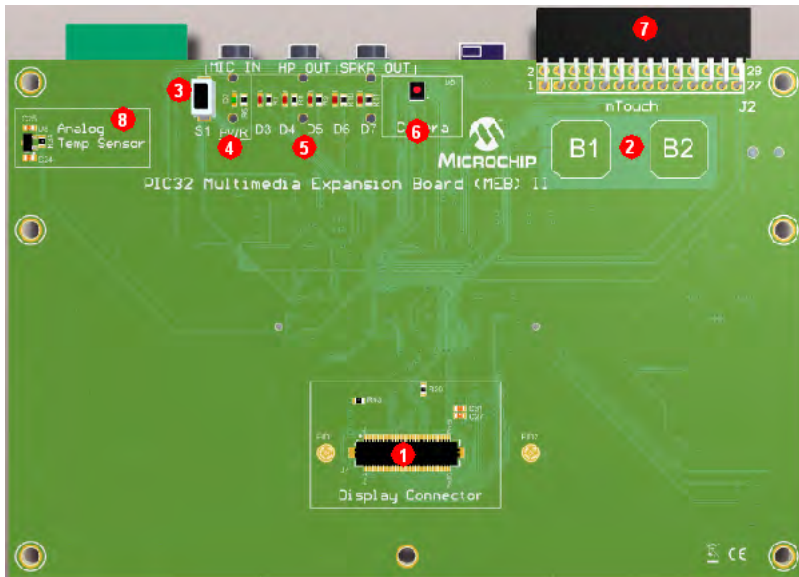


Figure 1: Multimedia Expansion Board II Layout (Front)

The underside of the MEB II includes these key features, as shown in Figure 2:

1. Regulated 5V and 3.3V power supply for powering the board via a 9-15V DC Adapter.
2. PIC32 Starter Kit connector (168-pin Hirose board-to-board connector).
3. 24-bit stereo audio codec (AK4953A).
4. Integrated 802.11bg wireless module (MRF24WG0MA).
5. Low-cost Bluetooth® HCI transceiver (BTM805).
6. EBI SRAM memory (IS61WV102416BLL).
7. microSD slot.
8. Analog accelerometer (ADXL325).

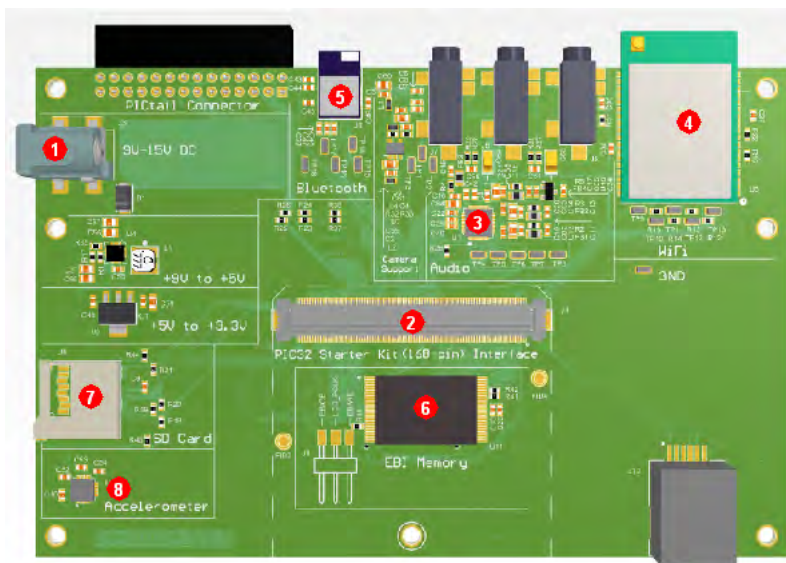


Figure 2: Multimedia Expansion Board II Layout (Back)

3.5.4.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

Explorer 16 Development Board

This demonstration application is demonstrated using Explorer 16 Development Board, PIC32 USB Starter Kit II and PIC32MZ Embedded Connectivity (EC) Starter Kit. The following hardware configuration should be set up before running this application.

1. Before attaching the PIC32MX795F512L PIM to the Explorer 16 Development Board, ensure that the processor select switch (S2) is in the PIM Position.
2. Short JP2 on the Explorer 16 Development Board to enable the LEDs.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Connectivity (EC) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

3.5.5 Building the Application

This section describes how to build this demonstration project.

Description

Once the hardware is configured, you are ready to build this application.

Project Setup :

From the MPLAB X IDE, Select "open project" and browse to this demo location. This demo application is available in "<Install Path>/apps/driver/nvm/nvm_read_write/firmware/nvm_read_write.X". This directory contains all the application related source files, header files as well as MPLAB X IDE project related workspace files.

How to build this demonstration application:

To build this demo in debug mode, Go to "Run" menu in MPLAB X IDE and select "Build Main Project" or select "Build Main Project" in toolbar. Once the project gets compiled successfully, you can run this demonstration application.

For more information on how to compile and program projects, refer to the MPLAB X IDE help available through the help menu in MPLAB X IDE.

Note:The appropriate project configurations should be selected before building the project for the configured hardware.

3.5.5.1 Configuring the Application

This section describes the list of configuration options, build options, and how they are configured for this demonstration application.

Build options:

This application supports both PIC32MX and PIC32MZ devices. From project properties of MPLAB X IDE, select "pic32mx_usb_starterkit_II" configuration if the target hardware is a PIC32MX device, or select "pic32mz_sk" configuration if the

target hardware is a PIC32MZ device.

The default target device is "PIC32MX795F512L" for "pic32mx_usb_starterkit_II" configuration and "PIC32MZ2048ECH144" for "pic32mz_sk" configuration with optimization level as "0". The various optimization levels can be selected from the list to run this demonstration.

Note: If the target device is changed, the appropriate PIM needs to be used to run this demonstration application.

Configuration options:

All of the demonstration application related configuration options are available in "system_config.h" file. This file is available in the "<Install Path>apps/driver/nvm/nvm_read_write/firmware/src/system_config/pic32mx_usb_starterkit_II" or "<Install Path>apps/driver/nvm/nvm_read_write/firmware/src/system_config/pic32mz_sk" directory based on the configuration selected. This file has the following list of configurations:

- General System Clock Services
- NVM Driver
- Interrupt services

Note: Based on the configurations, the demo application behavior will be changed.

3.5.5.2 Configuring the Libraries

This section describes the configuration options provided by MPLAB Harmony Libraries used by the application.

Macro Name	Description
#define DRV_NVM_INSTANCES_NUMBER	This definition selects the maximum number of hardware instances that can be supported by the dynamic NVM driver
#define DRV_NVM_CLIENTS_NUMBER	This definition selects the maximum number of clients that the NVM driver can support at run time
#define DRV_NVM_INTERRUPT_MODE	This macro controls the operation of the driver in the interrupt mode of operation
#define DRV_NVM_ROW_SIZE	This macro defines the row size of target Flash
#define DRV_NVM_PAGE_SIZE	This macro selects the NVM page size size
#define NVM_BASE_ADDRESS	This macro defines the address to which the data is written and read

3.5.5.3 Customizing the Application

This section explains the steps involved in customizing the existing demonstration application.

Customizing Existing Demonstration Application:

In this demo application, the NVM driver operates in interrupt mode. Following are the steps to be used to customize this application in polling mode.

1. Change #define [DRV_NVM_INTERRUPT_MODE](#) as false in "system_config.h" file
2. Remove the NVM ISR routine from "system_int.c" file
3. Call the [DRV_NVM_Tasks](#) function in "system_task.c" file

After changing the above configurations, compile the customized application and program the target device.

3.5.6 Running the Application

This section demonstrates how to run the NVM Driver demonstration.

Description

This is a simple demonstration to show how to configure and make use of NVM driver API's to implement and access on-board Flash memory of PIC32MX and PIC32MZ devices.

How to run this demonstration application:

Once the demonstration application is compiled successfully, you are ready to program the firmware in the target device.

To run the demonstration in debug mode, below are the steps to be followed.

1. Select appropriate configuration from project properties based on the target hardware.
2. Select your device programmer from the "hardware tool" in MPLAB X IDE.
3. Select the "Debug Main Project" in "Debug" menu or select the "Debug Main Project" in toolbar.

Build the selected configuration in the MPLAB X IDE project and program the demonstration board. The execution status (pass/fail) of the demonstration is indicated by LEDs on the demonstration board.

Demonstration Board	Successful Indication	Failure Indication
PIC32 USB Starter Kit II	LED2	LED1
PIC32MZ Embedded Connectivity (EC) Starter Kit	Green LED	Red LED

This demonstration first reads the data from specific NVM address define in system_config.h header file and erases the entire sector, write 150 bytes of data and reads back. The Pass/Fail criteria is indicated using LEDs as defined above.

3.5.7 Files

This section provides a complete list of source files required by the application and describes the purpose of adding each files.

MPLAB Harmony installation follows a standard directory structure for all application solutions. In this directory structure, the driver, plib, system services related source files, header files for this demonstration application are available in "<Installed Path>/framework/". Below are the list of files added in this demonstration application.

Files	Description
driver.h	Driver Library Interface Header
drv_common.h	This file defines all common macros and definitions used by the driver
drv_nvm.h	NVM device driver interface file.
system.h	System Service Interface Headers
sys_common.h	This file contains system Common definitions and declarations
sys_module.h	This file contains system module definitions
system_config.h	This is file contains all the system configurations
system_definitions.h	This contains system prototypes and definitions

app.h	This file contains application related definitions and prototypes
config_performance.c	BSP file for Explorer 16 Development Board
drv_nvm_dynamic.c	This file contains NVM device driver dynamic implementation
plib_int_pic32.c	Interrupt Source implementation for PIC32 devices
sys_int_pic32.c	This file contains functions related to the Interrupt System Service for PIC32 devices
app.c	This file contains application related implementations
main.c	This file contains application's main entry point
system_init.c	This file contains source code necessary to initialize the system
system_int.c	This file contains a definitions of the raw ISRs required to support the interrupt sub-system
system_tasks.c	This file will contain any source code necessary to maintain various tasks in the system

3.6 RTOS Demonstrations

3.6.1 Introduction

RTOS Demonstration Applications Help

Description

This distribution package contains a variety of RTOS based firmware projects that demonstrate the capabilities of the MPLAB Harmony services and stacks integrated with RTOS running on PIC32 devices. This help file describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related Framework help documentation.

3.6.2 Release Notes

MPLAB Harmony Version: v0.70b **RTOS Demonstrations Version:** 0.01a **Release Date:** 18Nov2013

New This Release:

This release contains the FreeRTOS basic project.

Known Issues:

Nothing to report for this release.

3.6.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

3.6.4 Demonstrations

This topic provides information on how to run the RTOS demonstration applications included in this release.

Description

All demonstrations provided should be built using the MPLAB X IDE development environment tool supplied by Microchip.

The following information communicates the general steps to building the demonstrations.

- Using MPLAB X IDE, click the File drop-down, select "Open Project...". Navigate to your Harmony installation folder. The default location on Windows will be: C:\Microchip\Harmony\<version>\apps\rtos
- In the \app\rtos folder there will be the demonstration applications.
- Within each demonstration folder, there will be a MPLAB X IDE project specific to each demonstration
- Select the project and click "Open Project"
- To configure the project specific to a preset board configuration, right click on the project and select "Set Configuration"

This will set the build configuration to match the supported board configuration.

3.6.4.1 FreeRTOS_basic_demo

3.6.4.1.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.6.4.1.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Connectivity (EC) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

3.6.4.1.3 Running the Demonstration

Provides instructions on how to build and run the FreeRTOS basic demonstration.

Description

Please use the following procedure to run the demonstration:

1. Load demonstration project into MPLAB X IDE.
2. Connect the mini-B debugger port on-board the PIC32MX or PIC32MZ Starter board to an USB port on the development computer using the USB cable provided in the kit.
3. Build, Download, and Run demonstration project on the target board.

The demonstration application features the following:

- Application creates one queue and four tasks. One task that sends the data using FreeRTOS queue to the two tasks that waits for the data in the queue. (QueueReceiveTask2 priority is higher than the QueueReceiveTask1 priority.)
- QueueReceiveTask2 receives the data first, toggles the LED, and then sleeps for the specified time
- QueueReceiveTask1 receives the next data since QueueReceiveTask2 is not in running state
- QueueReceiveTask1 receives the data, toggles the LED and waits for the data arrival

3.6.4.2 uC_OS_III_basic_demo

3.6.4.2.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.6.4.2.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Connectivity (EC) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

3.6.4.2.3 Running the Demonstration

Provides instructions on how to build and run the uC_OS_III basic demonstration.

Description

Demonstration applications are available in the following folders:

- \app\rtos\freeRTOS\gfx
- \app\rtos\uC_OS_III\gfx
- \app\rtos\freeRTOS\basic
- \app\rtos\uC_OS_III\basic

Within each demonstration folder, there will be a MPLAB X IDE project specific to each demonstration

Select the project and click "Open Project"

These demos will show the following:

- <gfx> The selected RTOS integrated in with the Harmony compliant Graphics stack.
- <basic> The basic blink LED demo showing the selected RTOS running with the hardware.

Once the demo is up running an LED will toggle every 500ms. This demo will show the RTOS running with the selected hardware.

3.6.4.3 FreeRTOS RTOS with Graphics

3.6.4.3.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.6.4.3.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description**PIC32 USB Starter Kit II**

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Connectivity (EC) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

3.6.4.3.3 Running the Demonstration

Provides instructions on how to build and run the FreeRTOS RTOS with Graphics demonstration.

Description

Demonstration applications are available in the following folders:

- \apps\rtos\freeRTOS\gfx
- \apps\rtos\uC_OS_III\gfx
- \apps\rtos\freeRTOS\basic
- \apps\rtos\uC_OS_III\basic

Within each demonstration folder, there will be a MPLAB X IDE project specific to each demonstration

Select the project and click "Open Project"

These demos will show the following:

- <gfx> The selected RTOS integrated in with the Harmony compliant Graphics stack.
- <basic> The basic blink LED demo showing the selected RTOS running with the hardware.

Once the demo is up running the screen color on the graphics display will change once every 500ms. A LED will toggle every 500ms.

The projects have been setup to use relative paths from the project folder and must remain under the apps\rtos\ root folders in order to build correctly.

3.6.4.4 Micrium uC_OS_III with Graphics

3.6.4.4.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.6.4.4.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description**PIC32 USB Starter Kit II**

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Connectivity (EC) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

3.6.4.4.3 Running the Demonstration

Provides instructions on how to build and run the Micrium uC_OS_III with Graphics demonstration.

Description

Demonstration applications are available in the following folders:

- \app\rtos\freeRTOS\gfx
- \app\rtos\uC_OS_III\gfx
- \app\rtos\freeRTOS\basic
- \app\rtos\uC_OS_III\basic

Within each demonstration folder, there will be a MPLAB X IDE project specific to each demonstration

Select the project and click "Open Project"

These demos will show the following:

- <gfx> The selected RTOS integrated in with the Harmony compliant Graphics stack.
- <basic> The basic blink LED demo showing the selected RTOS running with the hardware.

Once the demo is up running the screen color on the graphics display will change once every 500ms. A LED will toggle every 500ms.

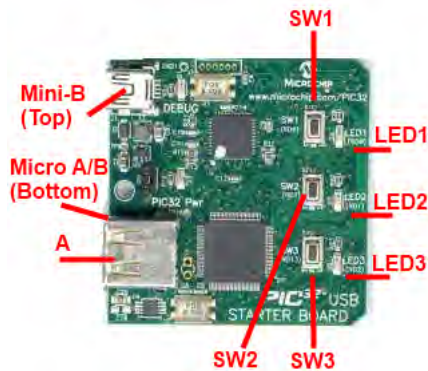
The projects have been setup to use relative paths from the project folder and must remain under the apps\rtos\ root folders in order to build correctly.

Under the uC_OS_III\ folder, an extra step must be taken to build the project successfully. The RTOS source code and appropriate port for the selected hardware must be download from Micrium and installed into the follow folder, <harmony_install_path>\third_party\rtos\.

3.6.5 Demonstration Board Information

3.6.5.1 PIC32 USB Starter Kit II

Provides information on the PIC32 USB Starter II.

Description

SW1 - Application switch. Tied to RD6.

SW2 - Application switch. Tied to RD7.

SW3 - Application switch. Tied to RD13.

LED1 - Application LED. Tied to RD0.

LED2 - Application LED. Tied to RD1.

LED3 - Application LED. Tied to RD2.

More Information

Microchip Part Number : DM320003-2

3.6.5.2 PIC32MZ Embedded Connectivity (EC) Starter Kit

Provides details information on the PIC32MZ Embedded Connectivity (EC) Starter Kit.

Description

Microchip Part Number: DM320006 (without Crypto) or DM320006-C (with Crypto)

The top assembly of the board includes these key features, as indicated in Figure 1:

1. PIC32MZ2048ECH144 32-bit microcontroller.
2. Green power indicator LED.
3. On-board crystal or oscillator for precision microcontroller clocking (12 MHz).
4. USB connectivity for on-board debugger communications.
5. Green debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 host-based applications.
9. HOST mode power jumper.
10. Daughter board connectors for flexible Ethernet PHY options.
11. 32 kHz oscillator for RTCC and Timer1 (optional).
12. External 2 GB SQI memory for expanded memory application.

13. Jumper for using or disconnecting the on-board debugger.

NOTE: When running self-powered USB device applications, open the jumper JP1 to prevent possibly back-feeding voltage onto the Vbus from one port on the host to another (or from one host to another).

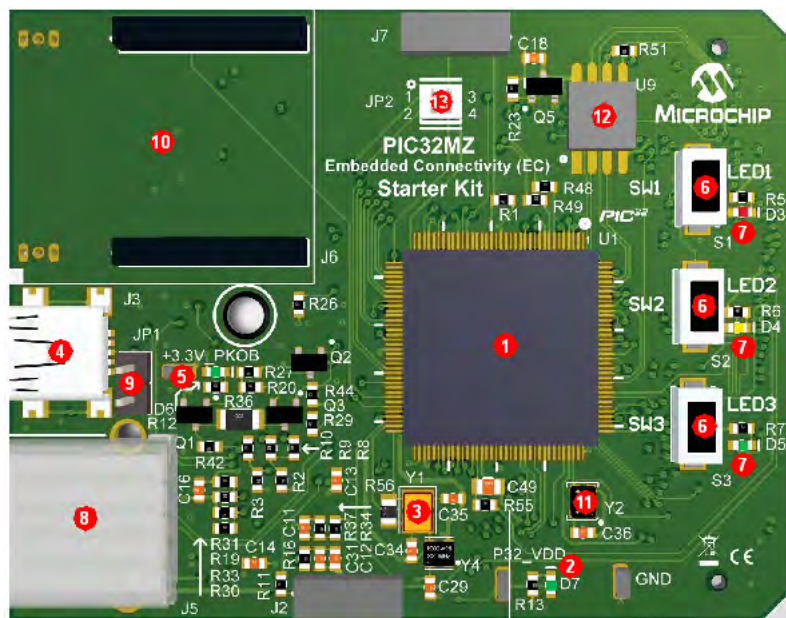


Figure 1: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Front)

The bottom assembly of the board includes these key features, as indicated in Figure 2:

1. PIC24FJ256GB106 USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Type micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.
5. 50 MHz Ethernet PHY oscillator.
6. USB Host and OTG power supply for powering PIC32 USB applications.

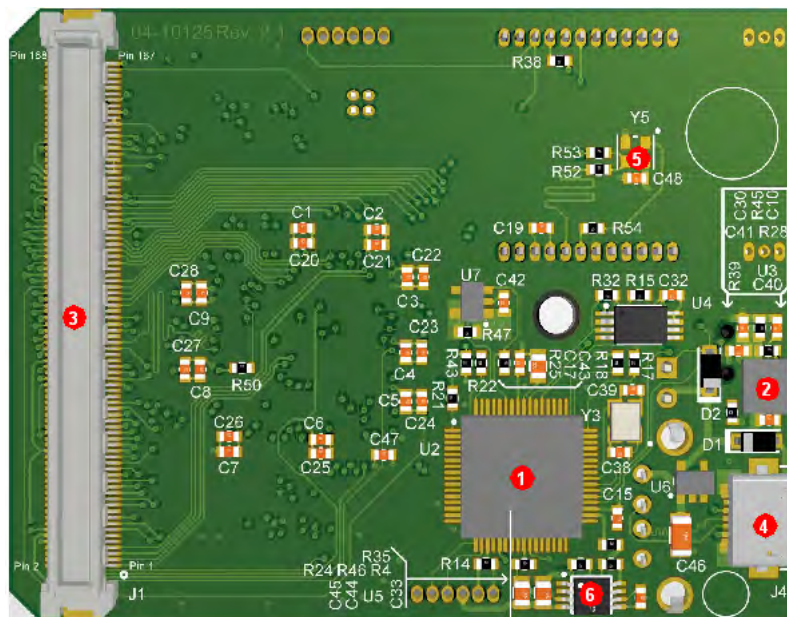


Figure 2: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Back)

Microchip Part Number: DM320006 (without Crypto) or DM320006-C (with Crypto)

The top assembly of the board includes these key features, as indicated in Figure 1:

1. PIC32MZ2048ECH144 32-bit microcontroller.
2. Green power indicator LED.
3. On-board crystal or oscillator for precision microcontroller clocking (12 MHz).
4. USB connectivity for on-board debugger communications.
5. Green debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 host-based applications.
9. HOST mode power jumper.
10. Daughter board connectors for flexible Ethernet PHY options.
11. 32 kHz oscillator for RTCC and Timer1 (optional).
12. External 2 GB SQI memory for expanded memory application.
13. Jumper for using or disconnecting the on-board debugger.

NOTE: When running self-powered USB device applications, open the jumper JP1 to prevent possibly back-feeding voltage onto the Vbus from one port on the host to another (or from one host to another).

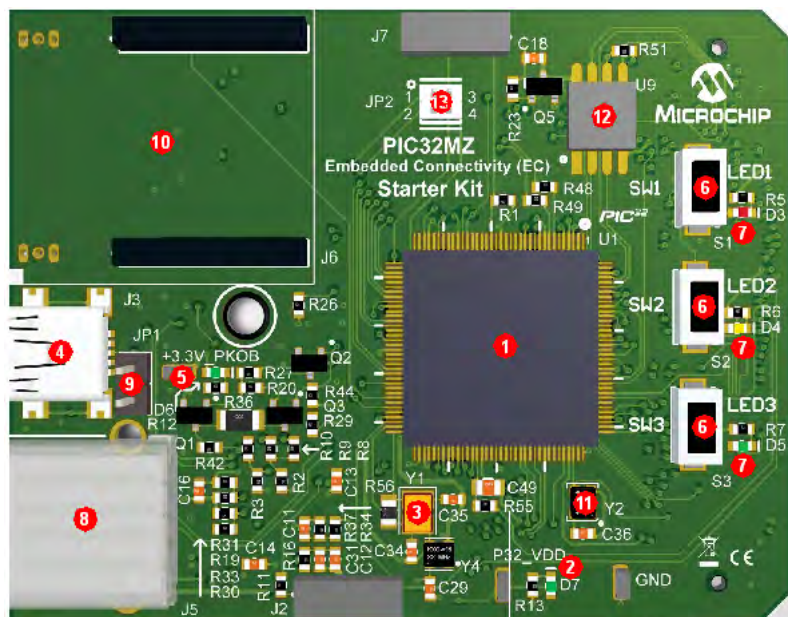


Figure 1: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Front)

The bottom assembly of the board includes these key features, as indicated in Figure 2:

1. PIC24FJ256GB106 USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Type micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.
5. 50 MHz Ethernet PHY oscillator.
6. USB Host and OTG power supply for powering PIC32 USB applications.

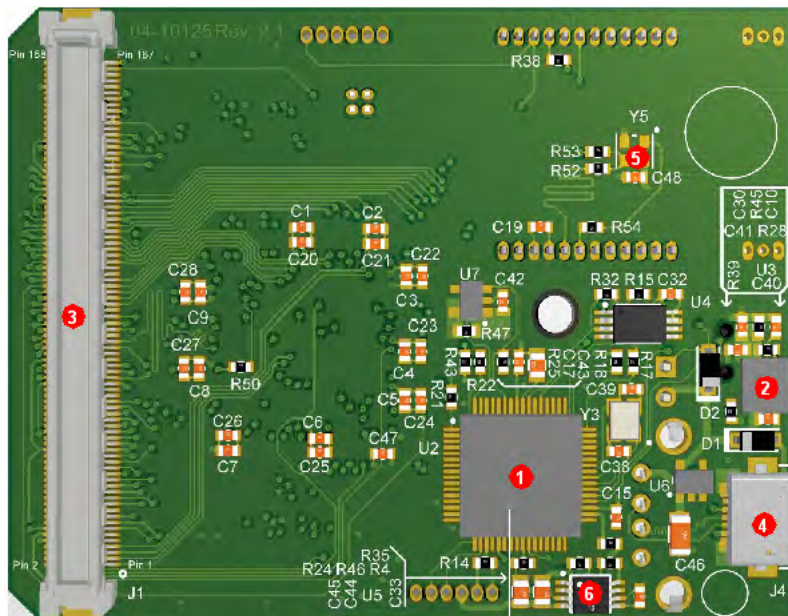


Figure 2: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Back)

3.7 SPI Driver Demonstrations

3.7.1 Introduction

This help file contains instructions and associated information about MPLAB Harmony SPI driver application demonstrations, which are contained in the MPLAB Harmony Library distribution.

Description

This application demonstrates the capabilities of the MPLAB Harmony SPI driver. This help file describes the hardware requirement and procedures to build, run the demo project on Microchip development tools. In this demo application, SPI driver is used to access the external EEPROM in the Explorer 16 Development Board to perform Write and Read operations and indicates the result by LED display.

To know more about MPLAB Harmony SPI driver, configuring the SPI driver and API's provided by the SPI driver, refer to the [SPI Driver Library](#) documentation.

3.7.2 Release Notes

MPLAB Harmony Version: v0.70b **SPI Driver Demonstrations Version:** 0.01a **Release Date:** 18Nov2013

New This release:

This is the first release of this SPI driver demonstration application. This demo has been developed to support PIC32MX devices only. The following demonstration shows the usage of the SPI driver:

- serial_eeprom

Limitations and Known Issues:

- Demo application is only developed and tested in Interrupt mode
- Changes in SPI driver interfaces will have impact on this demo application as well in subsequent releases
- This demo will support to run only on Explorer 16 Development Board

3.7.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

3.7.4 Hardware

This section describes how to select and use an appropriate development board and steps to configure the hardware to run this SPI driver demo.

3.7.4.1 Required Hardware

This section describes the overview of required hardware family and supported demonstration boards for running this SPI driver demo application.

Description

To run this demonstration project, you will require one of the following sets of hardware.

3.7.4.1.1 Supported Demonstration Boards

The following demo boards are supported in this release.

Description

Demonstration Board	Notes
Explorer 16 Development board	-

3.7.4.1.2 Hardware Information

Information on hardware used by the SPI driver demo application.

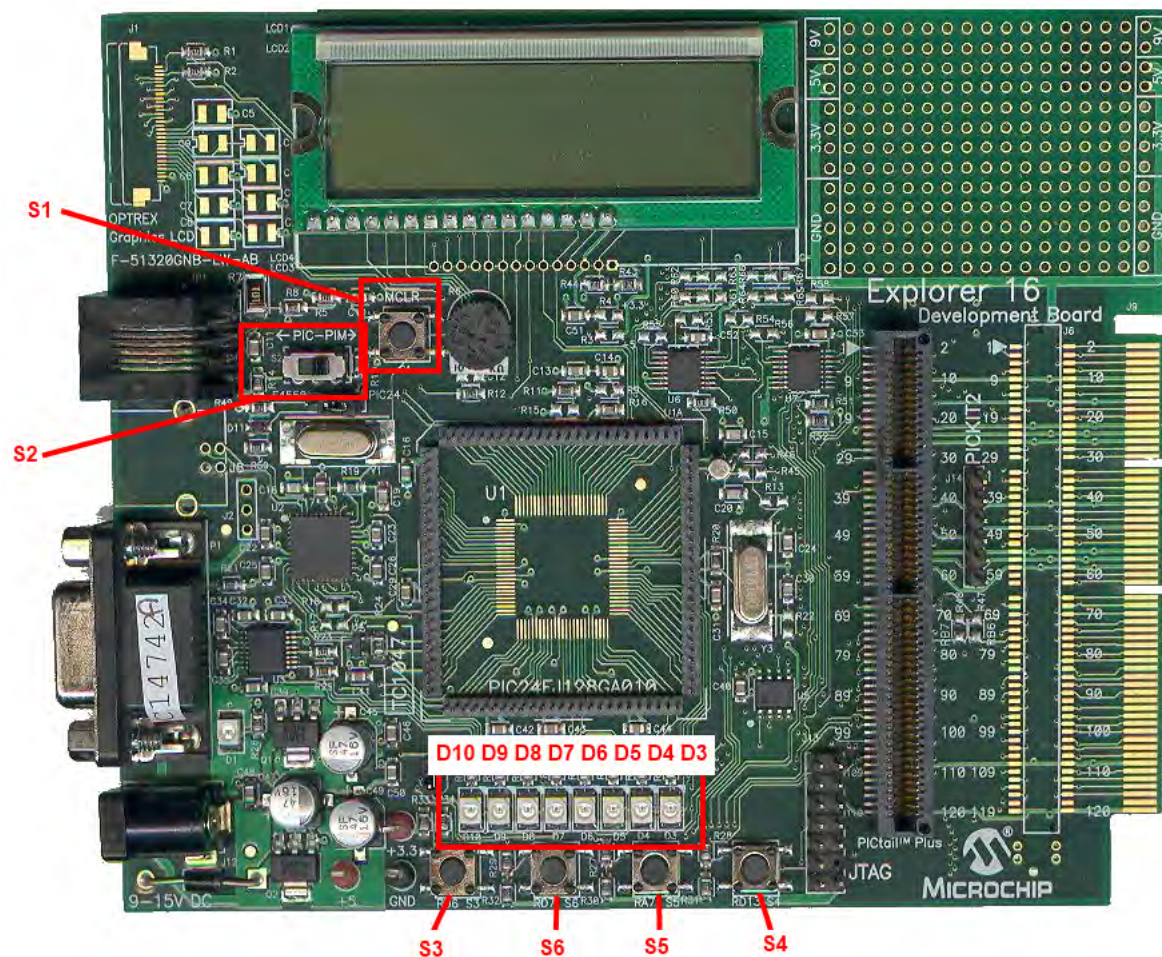
Description

The SPI driver demo application can be tried across on hardware boards. This section provide details on the different hardware boards that are used by the demo application.

3.7.4.1.2.1 Explorer 16 Development Board

Provides information on the Explorer 16 Development Board .

Description



S1 - Reset button (MCLR)

S2 - Processor switch. This switch determines which processor is running, the processor on the board or the processor on the Plug-In-Module (PIM).

S3, S4, S5, S6 - Application switches. For information about what pin is connected to this switch, please refer to the information for the PIM in use.

D3 through D10 - Application LEDs. For information about what pin is connected to this LED, please refer to the information for the PIM in use.

More Information:

Microchip Part Number: DM240001

3.7.4.1.2.2 PIC32MX795F512L Plug-in Module (PIM)

Provides information on the PIC32MX795F512L Plug-In-Module (PIM).

Description

This PIM is required while using the [Explorer 16 Development Board](#).

More Information

Microchip Part Number: MA320003

[Information Sheet](#)

3.7.4.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description**serial_eeprom demo**

This application is demonstrated using Explorer 16 Development Board. the following hardware configuration should be done before running this application.

1. Before attaching the PIC32MX795F512L PIM to the Explorer 16 Development Board, ensure that the processor select switch (S2) is in the PIM Position.
2. Short JP2 on the Explorer 16 Development Board to enable the LEDs.

Complete the previously described hardware configuration setup to to run this demo.

3.7.5 Building the Application

This section describes how to build this demo project.

Description

Once the hardware is configured, you are ready to build this application.

Project Setup :

From MPLAB X IDE, Select "open project" and browse to this demo location. This demo application is available in "<Install Path>/apps/driver/spi/serial_eeprom.X". This directory contains all the application related source files, header files as well as MPLAB X IDE project related workspace files.

How to build this demo application:

To build this demo in debug mode, Go to "Run" menu in MPLAB X IDE and select "Build Main Project" or select "Build Main Project" in toolbar. Once the project gets compiled successfully, you can run this demo application.

For more information on how to compile and program projects, refer to the MPLAB X IDE help available through the help menu in MPLAB X IDE.

Note: The appropriate project configurations should be selected before building the project for the configured hardware.

3.7.5.1 Customizing the Application

This section explains the steps involved in customizing the existing demo application.

Customizing Existing Demo Application:

In this demo application, the SPI driver is operates in interrupt mode. Below are the steps to be followed to customize this application in polling mode.

1. Change #define `DRV_SPI_INTERRUPT_MODE` as false in "system_config.h" file
2. Remove the SPI interrupt source Priority and sub priority set functions calls from "system_init.c" file
3. Remove the SPI ISR routine from "system_int.c" file
4. Call the `DRV_SPI_Tasks` function in "system_task.c" file

After changing the above configurations, compile the customized application and program the target device.

3.7.5.2 Configuring the Libraries

This section describes the configuration options provided by MPLAB Harmony Libraries used by the application.

#define <code>DRV_SPI_INSTANCES_NUMBER</code>	This definition selects the maximum number of hardware instances that can be supported by the dynamic SPI driver
#define <code>DRV_SPI_CLIENTS_NUMBER</code>	This definition selects the maximum number of clients that the SPI driver can support at run time
#define <code>DRV_SPI_INTERRUPT_MODE</code>	This macro controls the operation of the driver in the interrupt mode of operation
#define <code>DRV_SPI_PORTS_REMAP_USAGE</code>	This macro selects the SPI pins remap
#define <code>DRV_SPI_BUFFER_SIZE</code>	This macro selects the SPI buffer size
#define <code>DRV_SPI_FRAME_SYNC_PULSE_DIRECTION</code>	This macro selects the SPI frame sync pulse direction
#define <code>DRV_SPI_FRAME_SYNC_PULSE_POLARITY</code>	This macro selects the SPI frame sync pulse polarity
#define <code>DRV_SPI_FRAME_SYNC_PULSE_EDGE</code>	This macro selects the SPI frame sync pulse edge

3.7.5.3 Configuring the Application

This section describes the list of configuration options , build options and how they are configured for this demo application.

Build options:

This application supports only to demonstrate the SPI driver in master mode. This option "spi_master_explorer16" can be selected from the project properties in MPLAB X IDE or from the configuration drop down in toolbar.

The default target device is selected as "PIC32MX795F512L" with optimization level as "0". The various optimization levels can be selected from the list to run this demo.

Note: If the target device is changed, then appropriate PIM needs to be used to run this demo application

Configuration options:

All the demo application related configuration options are available in "system_config.h" file. This file is available in the "<Install Path>/apps/driver/spi/src/system_config/" directory based on the configured hardware. This file has the following configurations,

- General System Clock Services
- SPI Driver

Note: Based on the configurations, the demo application behavior will be changed.

3.7.6 Running the Application

This section demonstrates how to run the SPI driver demo.

Description

This is a simple demo to show how to configure and make use of SPI driver API's to implement and access on board EEPROM chip in the Explorer 16 Development Board.

How to run this demo application

Once the demo application is compiled successfully, you are ready to program the firmware in the target device.

To run the demo in debug mode, below are the steps to be followed.

1. Select your device programmer from the "hardware tool" in MPLAB X IDE.
2. Select the "Debug Main Project" in "Debug" menu or select the "Debug Main Project" in toolbar.

Once the device is successfully programmed, you can observe LED in the Explorer 16 Development Board will be turned ON. This shows the demo project has ran successfully and based on the LED "D4" or "D3" the demo application indicates the PASS/FAIL.

Notes:

If LED "D3" is turned ON, it indicates the EEPROM W/R functionality has failed.

If LED "D4" is turned ON, it indicates the EEPROM W/R functionality has passed.

3.7.7 Files

This section provides a complete list of source files required by the application and describes the purpose of adding each files.

MPLAB Harmony installation follows a standard directory structure for all application solutions. In this directory structure, the driver, plib, system services related source files, header files for this demo application are available in "<Installed Path>/framework/". Below are the list of files added in this demo application.

Files	Description
driver.h	Driver Library Interface Header
drv_common.h	This file defines all common macros and definitions used by the driver
drv_spi.h	SPI device driver interface file.
system.h	System Service Interface Headers
sys_common.h	This file contains system Common definitions and declarations
sys_module.h	This file contains system module definitions
system_config.h	This is file contains all the system configurations
system_definitions.h	This contains system prototypes and definitions
app.h	This file contains application related definitions and prototypes
config_performance.c	BSP file for Explorer 16 Development Board
drv_spi_dynamic.c	This file contains SPI device driver dynamic implementation

plib_int_pic32.c	Interrupt Source implementation for PIC32 devices
sys_clk.c	This file contains clock system service implementation
sys_clk_pic32.c	This file contains clock system service implementation specific to PIC32 devices
sys_queue.c	This file contains system queue services
sys_int_pic32.c	This file contains functions related to the Interrupt System Service for PIC32 devices
sys_ports.c	This file contains Ports System Service interface implementation.
app.c	This file contains application related implementations
main.c	This file contains application's main entry point
system_init.c	This file contains source code necessary to initialize the system
system_int.c	This file contains a definitions of the raw ISRs required to support the interrupt sub-system
system_tasks.c	This file will contain any source code necessary to maintain various tasks in the system

3.8 TCP/IP Demonstrations

3.8.1 Introduction

TCP/IP Web Server Demonstration Applications Help

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

3.8.2 Release Notes

MPLAB Harmony Version: v0.70b **TCP/IP Demonstrations Version:** 0.01a **Release Date:** 18Nov2013

The interface can change in the beta and/or 1.0 release.

New This Release:

Nothing to report for this release.

Known Issues:

Nothing to report for this release.

3.8.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

3.8.4 Demonstrations

Description of TCP/IP Stack Library Demonstration Application

Description

This section describes Microchip's TCP/IP Demonstration projects, including information about demonstration-hardware compatibility and also provides the information about how to configure and run the demonstration.

3.8.4.1 web_server

3.8.4.1.1 pic32_ethernet_starter_kit

This section describes the steps necessary to begin using Microchip's TCP/IP Web server Demonstration Application.

Description

This section describes Microchip's TCP/IP Web Server Demonstration project, including information about demonstration-hardware compatibility and also provides the information about how to configure and run the demonstration.

3.8.4.1.1.1 Supported Hardware

Lists supported demonstration and development hardware boards.

Description

Demo Board (click link for board information)	Notes
PIC32 Ethernet Starter Kit	--
PIC32MZ Embedded Connectivity (EC) Starter Kit	--

3.8.4.1.1.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

No hardware related configuration or jumper setting changes are necessary.

3.8.4.1.1.3 Running the Demonstration

Provides instructions on how to build and run the TCP/IP Web Server demonstration.

Description

This demonstration hosts a HTTP server when connected to a network. To view the web page hosted by the demonstration application open a web browser, type `http://mchpboard` in the address bar and press the enter key.

Please use the following procedure to run the demonstration:

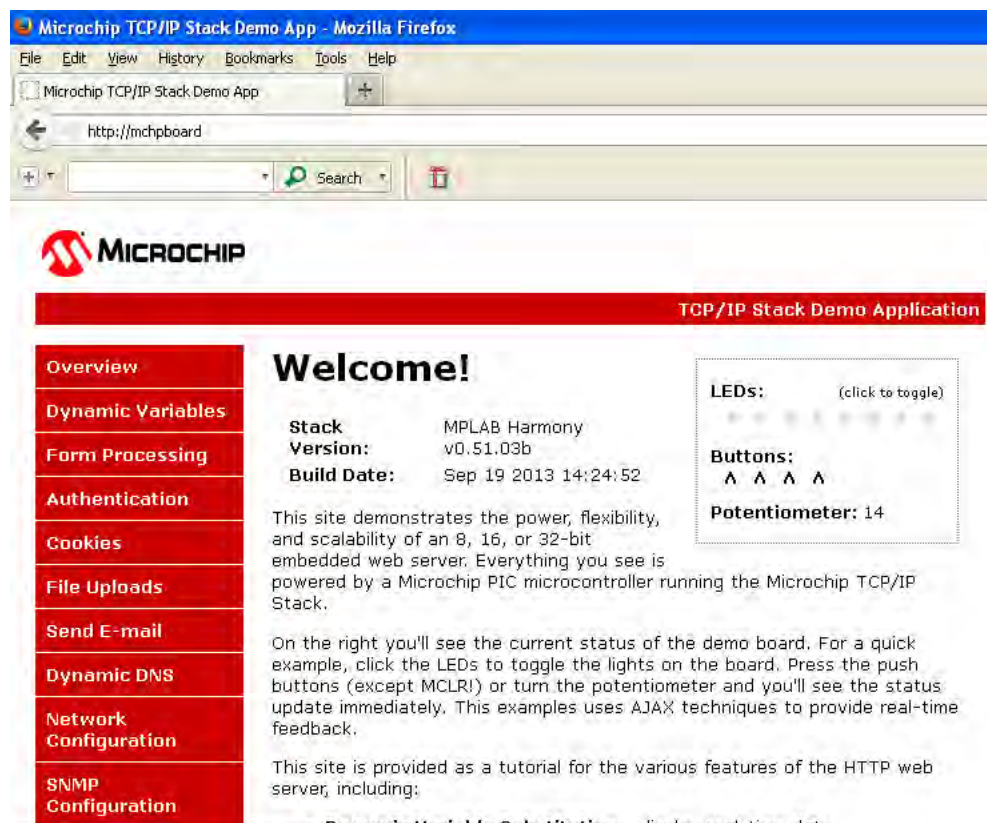
1. Load demonstration project into MPLAB X IDE.

2. Connect the mini-B debugger port on-board the PIC32MZ Embedded Connectivity (EC) Starter Kit or PIC32MZ Ethernet Starter Kit board to an USB port on the development computer using the USB cable provided in the kit.
3. Connect the RJ-45 Ethernet port on the PIC32MZ Embedded Connectivity (EC) Starter Kit or PIC32MZ Ethernet Starter Kit board to a network hub or an Ethernet port on the development computer using the Ethernet patch cord provided in the kit.
4. Build, Download, and Run demonstration project on the target board.
5. A HTTP server is hosted by the demonstration application. Open a web browser, type `http://mchpboard` in the address bar and press the enter key.

The demonstration application features following:

- a) Real-time hardware control and Dynamic Variables - On the Overview page the LEDs can be clicked to toggle LEDs on the Ethernet Starter Board. The buttons on Ethernet Starter Board can be pressed to see the Buttons on the webpage toggle. The dynamic variables can be updated in real time on the HTTP server.
- b) Form Processing - Input can be handled from the client by using GET and POST methods.
- c) Authentication - Shows an example to restricted access feature commonly used.
- d) File Uploads - Shows an example of file upload using POST method. The HTTP server can **accept** a user defined MPFS/MPFS2 image file for webpages.
- e) Board Configuration - MAC address, host name and IP address of the PIC32 Ethernet starter board can be viewed in the Network Configuration page and some configuration can be updated.

Note: LED functionality part of the demonstration is not implemented for PIC32MZ devices.



3.8.4.1.2 web_server_sdcard_fatfs

This section describes the steps necessary to begin using Microchip's TCP/IP Web server Demonstration Application.

Description

The TCP/IP SDCARD FATFS web server demo ('apps/tcpip/web_server_sdcard_fatfs/firmware/pic32_ethernet_starter_kit.X') exercises the HTTP web server running on PIC32 devices. SDCARD (Secure Digital Card) FATFS (File Allocation Table File System) web server demo has the web pages stored in external SD CARD and accessed through FATFS API. This demo currently only works with PIC32MZ using the Multimedia Expansion Board II (MEB II) Setup. PIC32MZ Testing and Fixes in progress.

3.8.4.1.2.1 Supported Hardware

Lists supported demonstration and development hardware boards.

Description

Demo Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	--
Multimedia Expansion Board II (MEB II)	--

3.8.4.1.2.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

- Plug the PIC32MZ Embedded Connectivity (EC) Starter Kit into application board connector on the MEB II
- Make sure a microSD card is formatted and loaded with the web pages provided under 'apps/tcpip/web_server_sdcard_fatfs/firmware/src/web_pages2_sdcard' directory
- Insert the microSD card with the web pages into the microSD card slot (J8) on the MEB II

3.8.4.1.2.3 Running the Demonstration

This section provides instructions on how to build and run the TCP/IP SD CARD FS Web Server demonstration.

Description

To view the web page hosted by the demonstration application open a web browser, type `http://mchpboard_e` in the address bar and press the Enter key.

Please use the following procedure to run the demonstration:

1. Load demonstration project into MPLAB X IDE.
2. Connect the mini-B debugger port on-board the PIC32MZ Embedded Connectivity (EC) Starter Kit or PIC32MZ Ethernet Starter Kit to a USB port on the Development computer using the USB cable provided in the kit.
3. Connect the RJ-45 Ethernet port on the PIC32MZ Embedded Connectivity (EC) Starter Kit or PIC32MZ Ethernet Starter Kit board to a network hub or an Ethernet port on the development computer using the Ethernet patch cord provided in the kit.
4. Make sure the microSD card with the web pages is inserted into the microSD card slot on the MEB II.
5. Build, Download, and Run demonstration project on the target board.
6. A HTTP server is hosted by the demonstration application. Open a web browser, type `http://mchpboard_e` in the address bar and press the Enter key. The demonstration application features following:

a. Real-time hardware control and Dynamic Variables - On the Overview page the LEDs can be clicked to toggle LEDs on the Ethernet Starter Board. The buttons on Ethernet Starter Board can be pressed to see the Buttons on the web page toggle. The dynamic variables can be updated in real time on the HTTP server.

Note: LED functionality part of the demonstration is somewhat limited due to errata items related to the functional multiplexing on GPIO and Ethernet pins.

b. Form Processing - Input can be handled from the client by using GET and POST methods.

c. Authentication - Shows an example to restricted access feature commonly used.

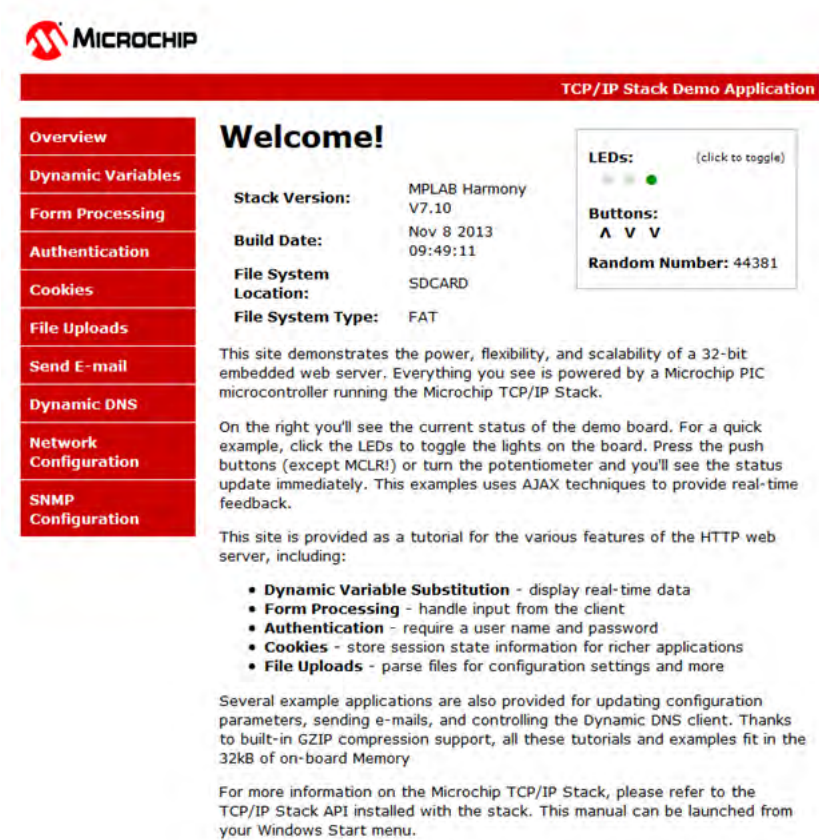
d. Cookies – Shows the example of storing small text strings on the client side.

e. File Uploads - Shows an example of file upload using POST method. The HTTP server can [accept](#) a user defined MPFS/MPFS2 image file for web pages.

f. Send E-mail – Shows a simple SMTP POST methods.

g. Dynamic DNS – Exercise Dynamic DNS capabilities.

h. Network Configuration - MAC address, host name and IP address of the PIC32 Ethernet starter board can be viewed in the Network Configuration page and some configuration can be updated.



3.8.4.1.3 web_server_nvm_mpfs

This section describes the steps necessary to begin using Microchip's TCP/IP Web server Demonstration Application.

Description

The TCP/IP NVM MPFS web server demo ('apps/tcpip/web_server_nvm_mpfs/firmware/pic32_ethernet_starter_kit.X') exercises the HTTP web server running on PIC32 devices. The Non-Volatile Memory (NVM) MPFS (Microchip Proprietary File System) web server demonstration has the web pages stored in internal Flash and accessed through the MPFS API.

3.8.4.1.3.1 Supported Hardware

Lists supported demonstration and development hardware boards.

Description

Demo Board (click link for board information)	Notes
PIC32 Ethernet Starter Kit	--
PIC32MZ Embedded Connectivity (EC) Starter Kit	--

3.8.4.1.3.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

No hardware related configuration or jumper setting changes are necessary.

3.8.4.1.3.3 Running the Demonstration

This section provides instructions on how to build and run the TCP/IP Web Server demonstration.

Description

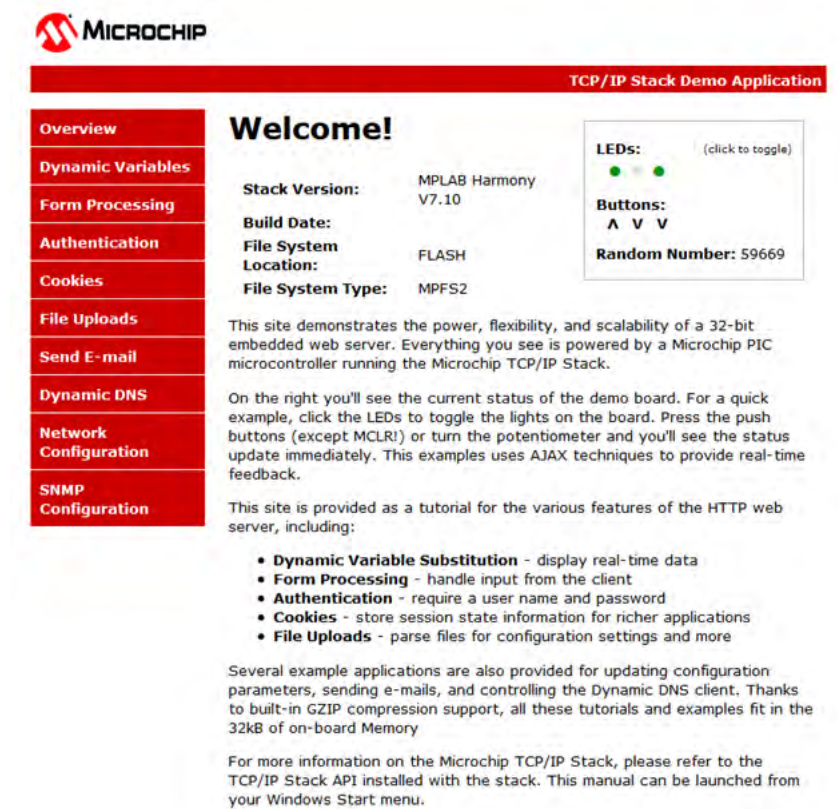
To view the web page hosted by the demonstration application open a web browser, type `http://mchpboard_e` in the address bar and press the enter key.

Please use the following procedure to run the demonstration:

1. Load demonstration project into MPLAB X IDE.
2. Connect the mini-B debugger port on-board the PIC32MZ Embedded Connectivity (EC) Starter Kit or PIC32MZ Ethernet Starter Kit board to a USB port on the Development computer using the USB cable provided in the kit.
3. Connect the RJ-45 Ethernet port on the PIC32MZ Embedded Connectivity (EC) Starter Kit or PIC32MZ Ethernet Starter Kit board to a network hub or an Ethernet port on the development computer using the Ethernet patch cord provided in the kit.
4. Build, Download, and Run demonstration project on the target board.
5. A HTTP server is hosted by the demonstration application. Open a web browser, type `http://mchpboard_e` in the address bar and press the enter key. The demonstration application features following:
 - a. Real-time hardware control and Dynamic Variables - On the Overview page the LEDs can be clicked to toggle LEDs on the Ethernet Starter Board. The buttons on Ethernet Starter Board can be pressed to see the Buttons on the web page toggle. The dynamic variables can be updated in real time on the HTTP server.

Note: LED functionality part of the demonstration is somewhat limited due to errata items related to the functional multiplexing on GPIO and Ethernet pins.
 - b. Form Processing - Input can be handled from the client by using GET and POST methods.
 - c. Authentication - Shows an example to restricted access feature commonly used.
 - d. Cookies – Shows the example of storing small text strings on the client side.

- e. File Uploads - Shows an example of file upload using POST method. The HTTP server can accept a user defined MPFS/MPFS2 image file for web pages.
- f. Send E-mail – Shows a simple SMTP POST methods.
- g. Dynamic DNS – Exercise Dynamic DNS capabilities.
- h. Network Configuration - MAC address, host name and IP address of the PIC32 Ethernet starter board can be viewed in the Network Configuration page and some configuration can be updated.



3.8.5 Demonstration Board Information

Information on Hardware used by the TCP/IP Demonstration Applications.

Description

This section provide details on the different hardware boards that are used by the demonstration applications.

3.8.5.1 PIC32 Ethernet Starter Kit

Information on PIC32 Ethernet Starter Kit Hardware.

Description

This board provides a low-cost, modular development system for Microchip's line of 32-bit Microcontrollers (MCUs)

PIC32 Ethernet Starter Kit



More Information

Microchip Part Number: DM320004

[Microchip Product Page](#)

3.8.5.2 PIC32MZ Embedded Connectivity (EC) Starter Kit

Information on the PIC32MZ Embedded Connectivity (EC) Starter Kit.

Description

Microchip Part Number: DM320006 (without Crypto) or DM320006-C (with Crypto)

The top assembly of the board includes these key features, as indicated in Figure 1:

1. PIC32MZ2048ECH144 32-bit microcontroller.
2. Green power indicator LED.
3. On-board crystal or oscillator for precision microcontroller clocking (12 MHz).
4. USB connectivity for on-board debugger communications.
5. Green debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 host-based applications.
9. HOST mode power jumper.
10. Daughter board connectors for flexible Ethernet PHY options.
11. 32 kHz oscillator for RTCC and Timer1 (optional).
12. External 2 GB SQI memory for expanded memory application.
13. Jumper for using or disconnecting the on-board debugger.

NOTE: When running self-powered USB device applications, open the jumper JP1 to prevent possibly back-feeding voltage onto the Vbus from one port on the host to another (or from one host to another).

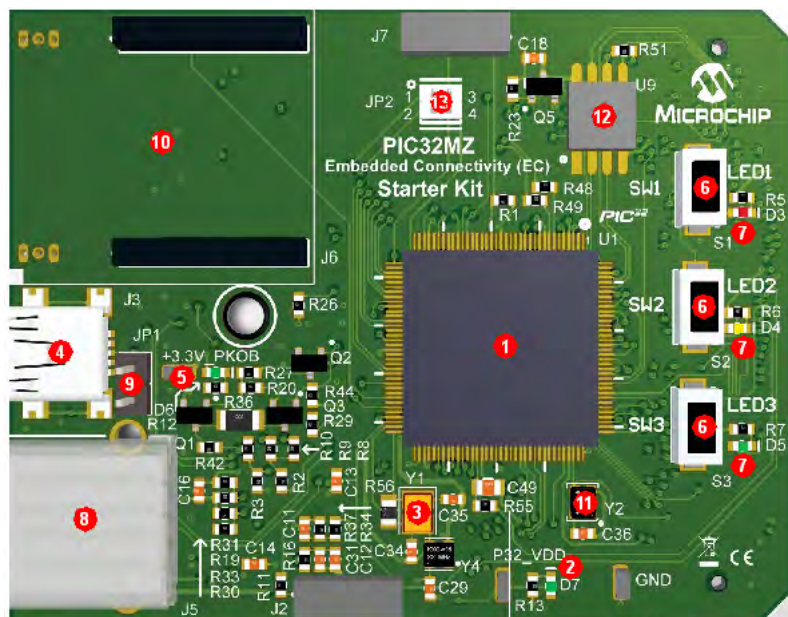


Figure 1: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Front)

The bottom assembly of the board includes these key features, as indicated in Figure 2:

1. PIC24FJ256GB106 USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Type micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.
5. 50 MHz Ethernet PHY oscillator.
6. USB Host and OTG power supply for powering PIC32 USB applications.

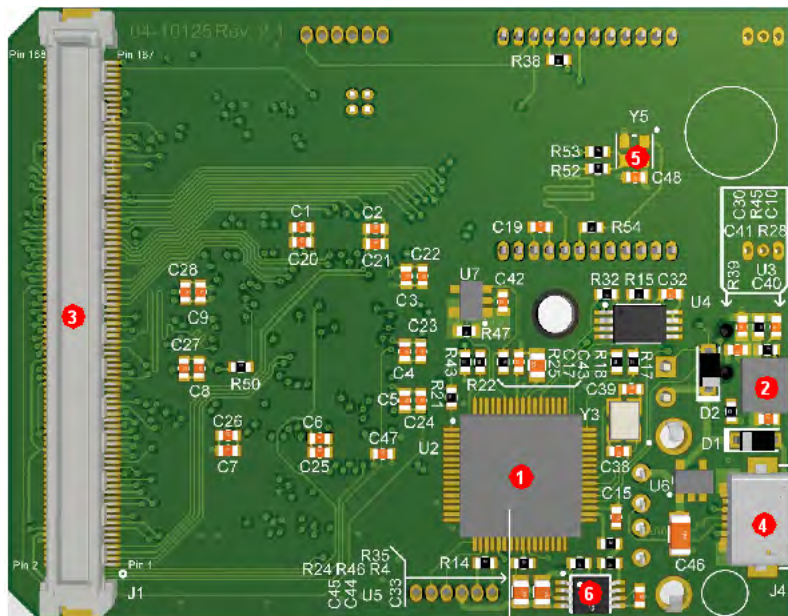


Figure 2: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Back)

Microchip Part Number: DM320006 (without Crypto) or DM320006-C (with Crypto)

The top assembly of the board includes these key features, as indicated in Figure 1:

1. PIC32MZ2048ECH144 32-bit microcontroller.
2. Green power indicator LED.
3. On-board crystal or oscillator for precision microcontroller clocking (12 MHz).
4. USB connectivity for on-board debugger communications.
5. Green debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 host-based applications.
9. HOST mode power jumper.
10. Daughter board connectors for flexible Ethernet PHY options.
11. 32 kHz oscillator for RTCC and Timer1 (optional).
12. External 2 GB SQI memory for expanded memory application.
13. Jumper for using or disconnecting the on-board debugger.

NOTE: When running self-powered USB device applications, open the jumper JP1 to prevent possibly back-feeding voltage onto the Vbus from one port on the host to another (or from one host to another).

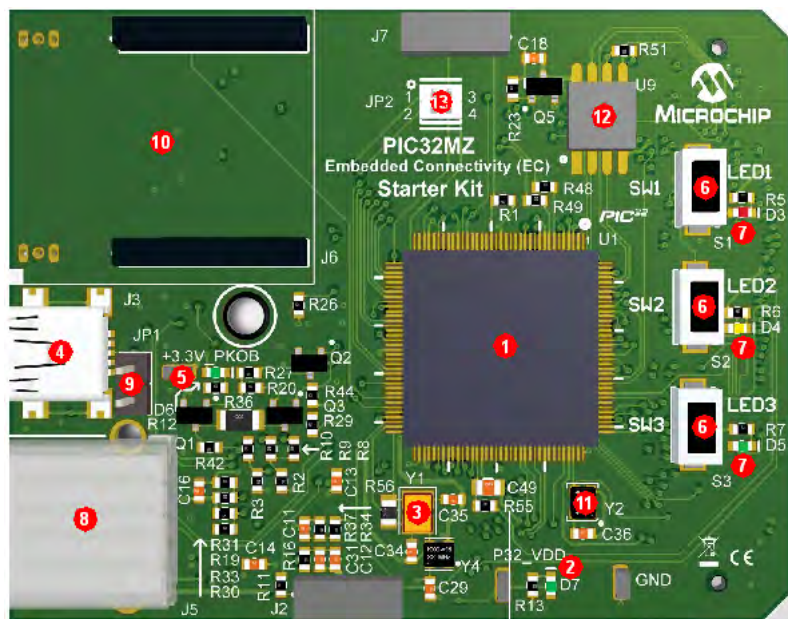


Figure 1: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Front)

The bottom assembly of the board includes these key features, as indicated in Figure 2:

1. PIC24FJ256GB106 USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Type micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.
5. 50 MHz Ethernet PHY oscillator.

6. USB Host and OTG power supply for powering PIC32 USB applications.

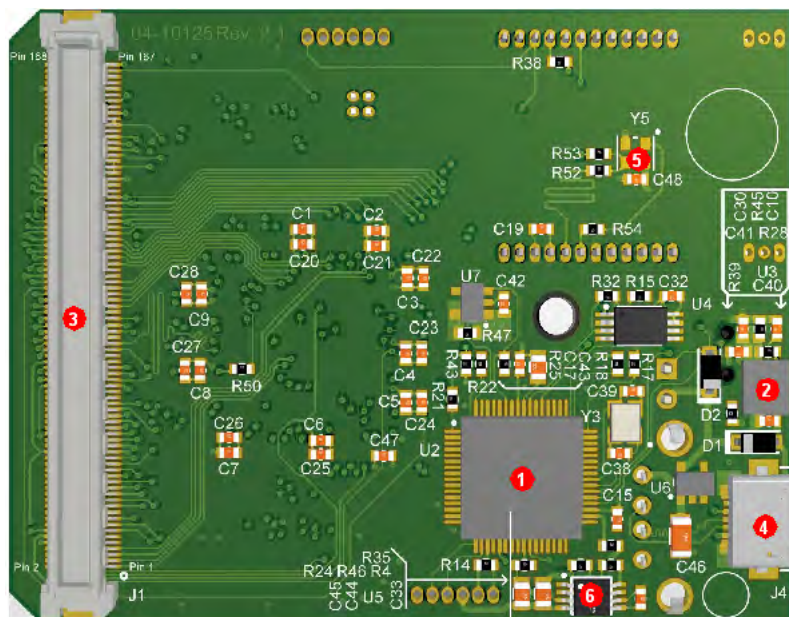


Figure 2: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Back)

3.8.5.3 Multimedia Expansion Board II (MEB II)

Provides details information on the Multimedia Expansion Board II (MEB) II.

Description

Microchip Part Number: DM320005-2

[Product Page](#)

The front side of the MEB II includes these key features, as shown in Figure 1:

1. Display daughter board connector (60-pin Hirose board-to-board connector).
2. mTouch™ buttons.
3. Push Button.
4. Power LED.
5. User LEDs.
6. VGA Camera (OVM7690).
7. PICtail™ Connector.
8. Analog temperature sensor.

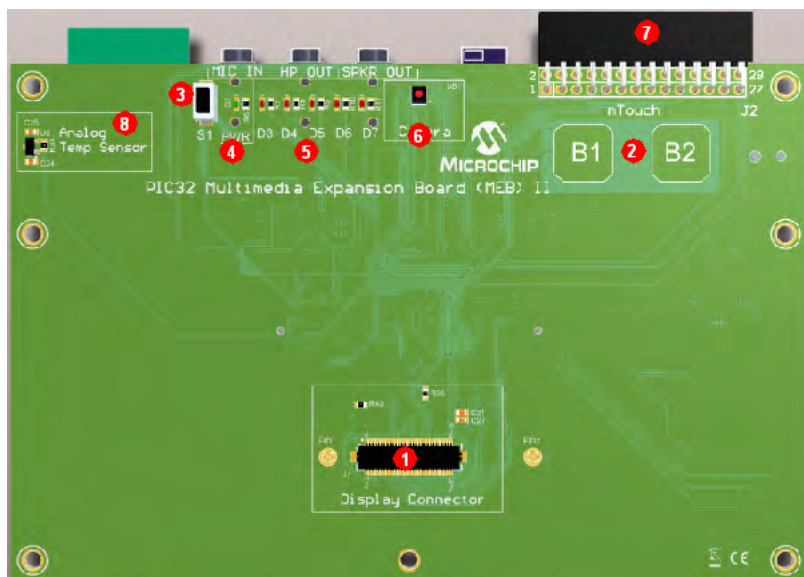


Figure 1: Multimedia Expansion Board II Layout (Front)

The underside of the MEB II includes these key features, as shown in Figure 2:

1. Regulated 5V and 3.3V power supply for powering the board via a 9-15V DC Adapter.
2. PIC32 Starter Kit connector (168-pin Hirose board-to-board connector).
3. 24-bit stereo audio codec (AK4953A).
4. Integrated 802.11bg wireless module (MRF24WG0MA).
5. Low-cost Bluetooth® HCI transceiver (BTM805).
6. EBI SRAM memory (IS61WV102416BLL).
7. microSD slot.
8. Analog accelerometer (ADXL325).

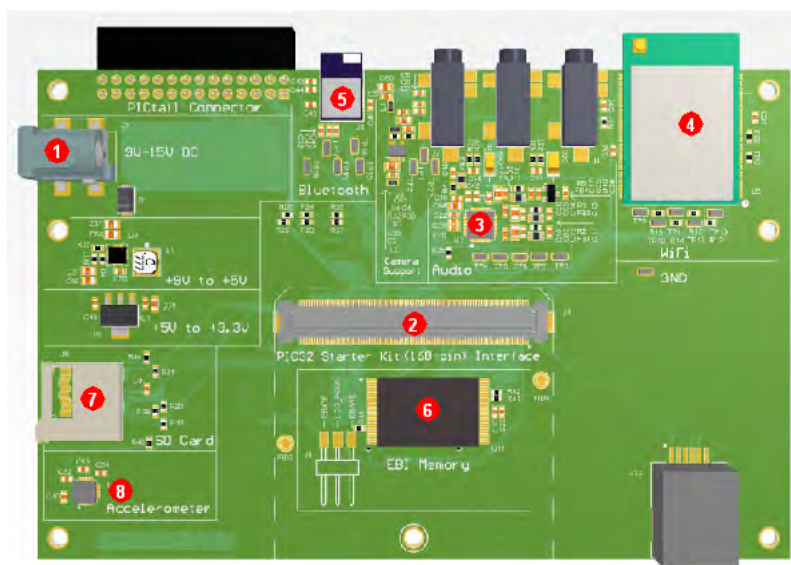


Figure 2: Multimedia Expansion Board II Layout (Back)

3.9 USART Driver Demonstration

3.9.1 Introduction

This help manual contains instructions and information about MPLAB Harmony USART driver demo application, which are contained in the MPLAB Harmony Library distribution.

Description

This application demonstrates how to use the MPLAB Harmony USART driver. This help manual describes the hardware requirement and procedures to build and execute the demo project on Microchip development tools. In this demo application, USART driver will initially transmit strings of data and then [accept](#) and characters received and transmit the data back. Error or success status is indicated by LED display.

To know more about MPLAB Harmony USART driver, configuring the USART driver and API's provided by the USART driver, refer to [USART Driver Library](#) documentation.

3.9.2 Release Notes

MPLAB Harmony Version: v0.70b **USART Driver Demonstration:** 0.01a **Release Date:** 18Nov2013

New This Release:

This is the first release of this USART driver demo application. This demo has been developed to support PIC32MX devices only.

Known Issues:

- Changes in USART driver interfaces will have impact on this demo application as well in subsequent releases
- This demo is has not been developed to run on PIC32MZ devices. Users must add a new configuration if intended to run on PIC32MZ devices.

3.9.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS

BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

3.9.4 Hardware

This section describes how to select and use an appropriate development board and steps to configure the hardware to run this USART driver demo.

3.9.4.1 Required Hardware

This section describes the overview of required device family and supported demonstration boards to run this USART driver demo application.

Description

This section describes the hardware required to execute the Harmony USART driver demo application.

3.9.4.1.1 Supported Demonstration Boards

Lists supported demonstration and hardware development boards.

Description

Demonstration Board	Notes
Explorer 16 Development Board	-

3.9.4.1.2 Hardware Information

Information on hardware used by the USART driver demo application.

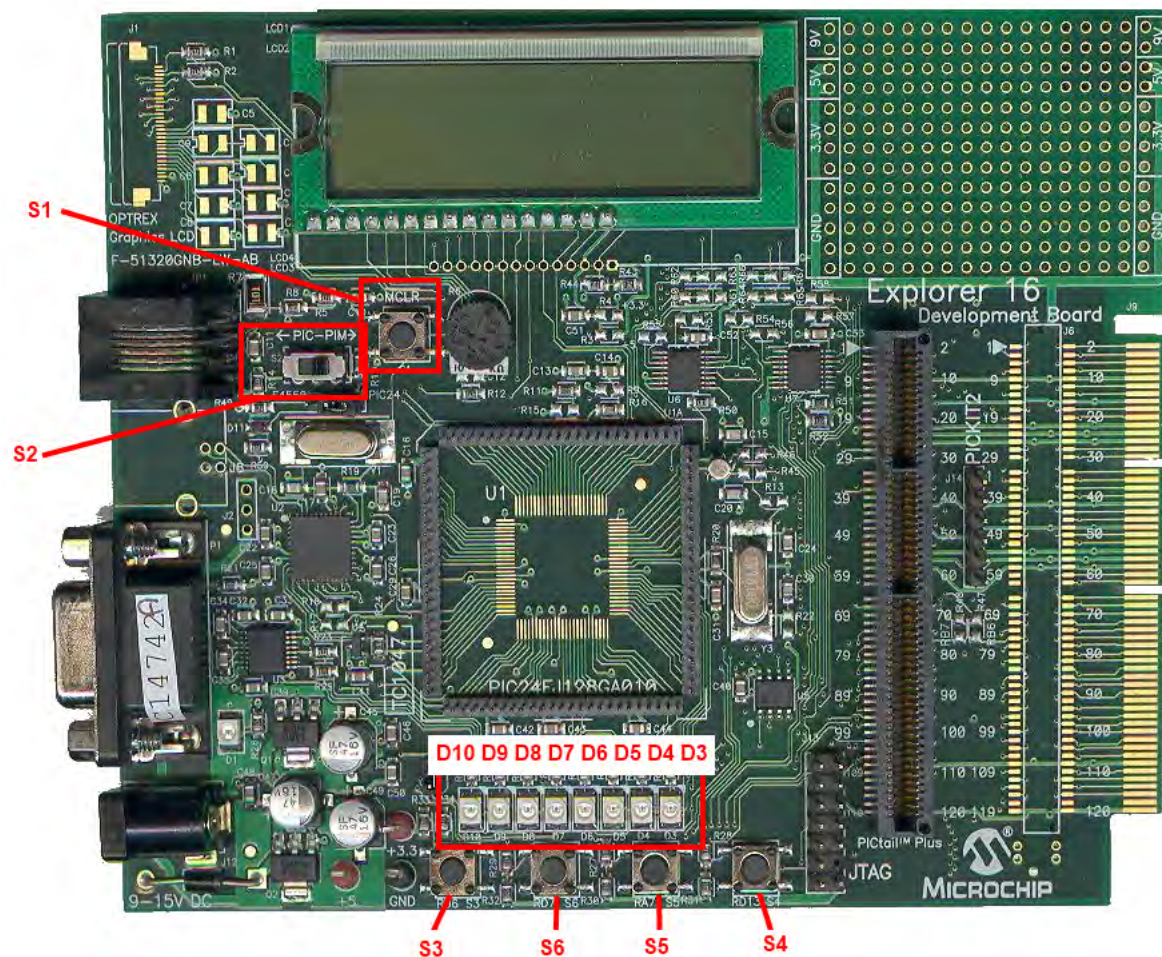
Description

The USART driver demo application can be tried across on hardware boards. This section provide details on the different hardware boards that are used by the demo application.

3.9.4.1.2.1 Explorer 16 Development Board

Provides information on the Explorer 16 Development Board .

Description



S1 - Reset button (MCLR)

S2 - Processor switch. This switch determines which processor is running, the processor on the board or the processor on the Plug-In-Module (PIM).

S3, S4, S5, S6 - Application switches. For information about what pin is connected to this switch, please refer to the information for the PIM in use.

D3 through D10 - Application LEDs. For information about what pin is connected to this LED, please refer to the information for the PIM in use.

More Information:

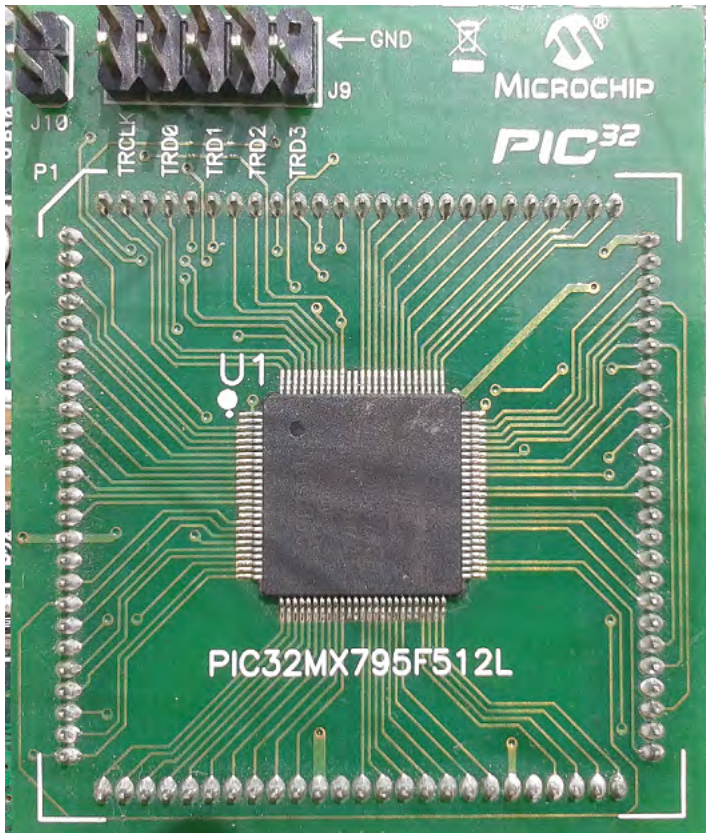
Microchip Part Number: DM240001

3.9.4.1.2.2 PIC32MX795F512L Plug-in Module (PIM)

Provides information on the PIC32MX795F512L Plug-In-Module (PIM).

Description

This PIM is required while using the [Explorer 16 Development Board](#).



More Information

Microchip Part Number: MA320003

[Information Sheet](#)

3.9.4.2 Configuring the Hardware

This section describes how to configure the supported hardware.

Description

Explorer 16 Development Board

Below are the hardware configuration settings required to execute this demo

1. Before attaching the PIC32MX795F512L PIM to the Explorer 16 Development Board, ensure that the processor select switch (S2) is in the PIM Position.
2. Short JP2 on the Explorer 16 Development Board to enable the LEDs.

3.9.5 Building the Application

This section describes how to build this demo project.

Description

Once the hardware is configured, you are ready to build this application.

Project Setup:

From MPLAB X IDE, Select "open project" and browse to this demo project location. This demo application is available in "<Install Path>/apps/driver/usart/usart_echo/firmware/usart_echo.X". This directory contains all the application related source files, header files as well as MPLAB X IDE project related workspace files.

How to build this demo application:

To build this demo in debug mode, Go to "Run" menu in MPLAB X IDE and select "Build Main Project" or select "Build Main Project" in toolbar. Once the project gets compiled successfully, you can run this demo application.

For more information on how to compile and program projects, refer to the MPLAB X IDE help available through the help menu in MPLAB X IDE.

Note: The appropriate project configurations should be selected before building the project for the configured hardware.

3.9.5.1 Configuring the Application

This section describes the list of configuration options , build options and how they are configured for this demo application.

Build options:

The default target device is selected as "PIC32MX795F512L" with optimization level as "0". The various optimization levels can be selected from the list to run this demo.

Note: If the target device is changed, then appropriate PIM needs to be used to run this demo application

Configuration options:

All the demo application related configuration options are available in "system_config.h" file. This file is available in the "<Install Path>/apps/driver/usart/usart_echo/firmware/src/system_config/pic32mx_default" directory based on the configured hardware.

Note: Based on the configurations, the demo application behavior will be changed.

3.9.5.2 Configuring the Libraries

This section describes the configuration options provided by MPLAB Harmony Libraries used by the application. The below parameters are defined in "system_config.h" header file.

Parameter Name	Description
<code>#define DRV_USART_INSTANCES_NUMBER</code>	This macro defines the maximum number of hardware instances that will be used by the application. The USART driver will support a maximum of that many hardware instances.
<code>#define DRV_USART_CLIENTS_NUMBER</code>	This macro defines the maximum number of clients that will be used by the application. The USART driver will support a maximum of that many clients.
<code>#define DRV_USART_INTERRUPT_MODE</code>	If this macro is set to "true", the application will run in interrupt mode. If this macro is set to "false", the application will run in polling mode
<code>#define DRV_USART_INTERRUPT_SOURCE_TX</code>	This macro defines the transmit interrupt source
<code>#define DRV_USART_INTERRUPT_SOURCE_RX</code>	This macro defines the receive interrupt source

#define SYS_USART_ID	This macro defines the USART module number used in the application
#define DRV_USART_XFER_BUFFER_NUMBER	This macro defines the number of transfer buffers used in the application
#define APP_UART_BAUDRATE	This macro defines the baud rate to be used

3.9.5.3 Customizing the Application

This section explains the steps involved in customizing the existing demo application.

Customizing Existing Demo Application:

Changing the USART module

In system_config.h header file, modify the *SYS_USART_ID* macro to change the USART module number.

Changing the demo to interrupt mode

By default, the USART demo application will run in polling mode. Below are the steps to be followed to run the demo in interrupt mode.

1. Change #define *DRV_USART_INTERRUPT_MODE* to true in "system_config.h" file
2. Remove the DRV_USART_Tasks function call in sys_tasks in "system_task.c" file
3. Call the DRV_USART_Tasks function in ISR in "system_int.c" file

Changing USART settings

To change the USART related parameters such as baud rate, parity settings, number of stop bits, etc., select appropriate values in *drvUSARTInit* structure in system_init.c file

After changing the above configurations, compile the customized application and program the target device.

3.9.6 Running the Application

This section demonstrates how to run the USART driver demo.

Description

Once the demo application has been compiled successfully for the intended configuration, program the firmware into the target device.

Upon execution, the application will transmit the following strings through USART with settings as defined in the application.

```
/*****
```

```
Welcome to Microchip USART Echo Demo App.
```

```
Press any character. Data will be echoed back.
```

```
Press 'ESC' key to exit the application
```

```
*****/
```

After transmitting the above text, the firmware will read any characters received through USART and it will transmit back the same data. If the received character is "ESC" (0x1B), the program will enter into idle mode. Once in Idle mode, the firmware will neither transmit nor receive any data. If the application enters Idle mode, LED 5 of the Explorer 16 Development Board will

illuminate. If any error occurs, LED 9 of the Explorer 16 Development Board will illuminate.

3.9.7 Files

This section provides a complete list of source and header files used by the application and describes the purpose of adding each files.

MPLAB Harmony installation follows a standard directory structure for all application solutions. In this directory structure, the driver, plib, system services related source files, header files for this demo application are available in "<Installed Path>/framework/". Below are the list of files added in this demo application.

Files	Description
driver.h	Driver Library Interface Header
drv_common.h	This file defines all common macros and definitions used by the driver
drv_usart.h	USART device driver interface file.
system.h	System Service Interface Headers
sys_common.h	This file contains system Common definitions and declarations
sys_module.h	This file contains system module definitions
system_config.h	This is file contains all the system configurations
system_definitions.h	This contains system prototypes and definitions
app.h	This file contains application related definitions and prototypes
config_performance.c	BSP file for the Explorer 16 Development Board
drv_usart_dynamic.c	This file contains USART device driver dynamic implementation
plib_int_pic32.c	Interrupt Source implementation for PIC32 devices
sys_clk.c	This file contains clock system service implementation
sys_clk_pic32.c	This file contains clock system service implementation specific to PIC32 devices
sys_queue.c	This file contains system queue services
sys_int_pic32.c	This file contains functions related to the Interrupt System Service for PIC32 devices
sys_ports.c	This file contains Ports System Service interface implementation.
app.c	This file contains application related implementations
main.c	This file contains application's main entry point
system_init.c	This file contains source code necessary to initialize the system
system_int.c	This file contains a definitions of the raw ISRs required to support the interrupt sub-system
system_tasks.c	This file will contain any source code necessary to maintain various tasks in the system

3.10 USB Demonstrations

3.10.1 USB Demonstrations

3.10.1.1 Introduction

USB Library Demonstration Applications Help

Description

This distribution package contains a variety of USB-related firmware projects that demonstrate the capabilities of the MPLAB Harmony USB stack. This help file describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To know more about MPLAB Harmony USB stack and configuring the USB stack and the APIs provided by the USB stack, refer to the [USB Library](#) documentation.

3.10.1.2 Release Notes

MPLAB Harmony Version: v0.70b **USB Demonstration:** 0.01a **Release Date:** 18Nov2013

New This Release:

Nothing to report in this release.

Known Issues:

Device Stack Demonstration Applications:

All Device Stack Demonstration applications uses a custom Board Support Package (BSP) to access demonstration board switches and LEDs. The board support package does not use PORTS Peripheral Library. In the next release, all Device Stack Demonstration Applications will be updated to use the standard Harmony BSP. The demonstration application specific issues are listed here:

cdc_com_port_single:

Nothing to report in this release.

cdc_com_port_dual:

Nothing to report in this release.

cdc_serial emulator:

- This version of the demo does not support baud rate change. The supported baud rate is 9600bps, 1 stop bit and no parity.

- The application code accesses the UART2 peripheral SFR directly. This will be updated in the next release to use the USART driver and the System Clock Service.

hid_basic:

Nothing to report in this release.

hid_keyboard:

Nothing to report in this release.

hid_joystick:

Nothing to report in this release.

hid_mouse:

Nothing to report in this release.

audio_speaker:

While running the pic32mz_sk_meb2_int_dyn configuration, a periodic glitch is heard in the output audio. This is a known issue and will be fixed in the next release of the stack.

Host Stack Demonstration Applications:

All Host Stack Demonstration applications uses a custom Board Support Package (BSP) to access demonstration board switches and LEDs. The board support package does not use PORTS Peripheral Library. In the next release, all Host Stack Demonstration Applications will be updated to use the standard Harmony BSP.

cdc_basic:

This demo is not tested for the pic32mx_usb_sk_int_dyn configuration and may not function when built for this configuration.

msd_basic:

While running this demo, if the USB Flash drive already contains a file named "File.txt", the file contents may become corrupted.

3.10.1.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

3.10.1.4 Demonstration Application Configurations

This topic provides information on the USB demonstration project configuration that are available in MPLAB Harmony.

Description

The available USB Demonstration application MPLAB X IDE projects feature support for multiple configurations. Selecting these configurations allow for the demonstration projects to run across different PIC32 microcontrollers and development boards. The following project configurations are available:

- **pic32mx_usb_sk_int_dyn** – Selecting this configuration will set up the demonstration application to run on the [PIC32 USB Starter Kit II](#) development board, with the PIC32MX795F512L microcontroller. The system will be configured for interrupt mode operation and drivers will be configured for dynamic operation mode.
- **pic32mz_sk_int_dyn** – Selecting this configuration will set up the demonstration application to run on the [PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#) development board, with the PIC32MZ2048ECH144 microcontroller. The system will be configured for interrupt mode operation and drivers will be configured for dynamic operation mode.
- **explorer16_pic32mx795f512l_int_dyn** – Selecting this configuration will set up the demonstration application to run on the [Explorer 16 Development Board](#) along with the PIC32MX795F512L microcontroller Plug In Module . The system will be configured for interrupt mode operation and drivers will be configured for dynamic operation mode.
- **pic32mx_usb_audio_int_dyn** - Selecting this configuration will set up the demonstration application to run on [PIC32 USB Digital Audio Accessory Board](#) along with the PIC32MX250F128B microcontroller. The system will be configured for interrupt mode operation and drivers will be configured for dynamic operation mode.
- **pic32mz_sk_meb2_int_dyn** – Selecting this configuration will set up the demonstration application to run on the PIC32MZ EC Starter Kit development board, with the PIC32MZ2048ECH144 microcontroller board attached to the [Multimedia Expansion Board II \(MEB II\)](#). The system will be configured for interrupt mode operation and drivers will be configured for dynamic operation mode.

The following figure shows how a configuration can be selected in MPLAB X IDE.



Alternatively, the active configuration can be selected in the Project Properties dialog box. The following table shows the availability of a configuration across available USB device demonstration applications. Green indicates support. Red indicates no support.

Configurations					
Demo Apps	pic32mx_usb_sk_int_dyn	pic32mz_sk_int_dyn	explorer16_pic32mx795f512l_int_dyn	pic32mx_usb_audio_int_dyn	pic32mz_sk_meb2_int_dyn
cdc_com_port_single	Green	Green	Red	Red	Red
cdc_com_port_dual	Green	Green	Red	Red	Red
cdc_serial_emulator	Red	Red	Green	Red	Red
hid_basic	Green	Green	Red	Red	Red
hid_keyboard	Green	Green	Red	Red	Red
hid_mouse	Green	Green	Red	Red	Red
hid_joystick	Green	Green	Red	Red	Red
generic	Green	Green	Red	Red	Red
audio_speaker	Red	Red	Red	Green	Green
msd_basic	Green	Green	Red	Red	Red

The following table shows the availability of a configuration across available USB device demonstration applications. Green indicates support. Red indicates no support.

Configurations					
Demo Apps	pic32mx_usb_sk_int_dyn	pic32mz_sk_int_dyn	explorer16_pic32mx795f512l_int_dyn	pic32mx_usb_audio_int_dyn	pic32mz_sk_meb2_int_dyn
cdc_basic	Red	Green	Red	Red	Red
msd_basic	Green	Green	Red	Red	Red

3.10.1.5 Demonstrations

Description of USB Demonstrations

Description

The USB Demonstrations are grouped into USB Device Stack and USB Host Stack Demonstrations.

3.10.1.5.1 Device

3.10.1.5.1.1 cdc_com_port_single

Demonstrates a USB CDC device, emulating a serial COM port.

Description

This demonstration application creates a USB CDC Device that enumerates as a single COM port on the host PC. The application demonstrates two-way communication between the USB device and the PC host.

3.10.1.5.1.1.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.10.1.5.1.1.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

3.10.1.5.1.1.3 Running the Demonstration

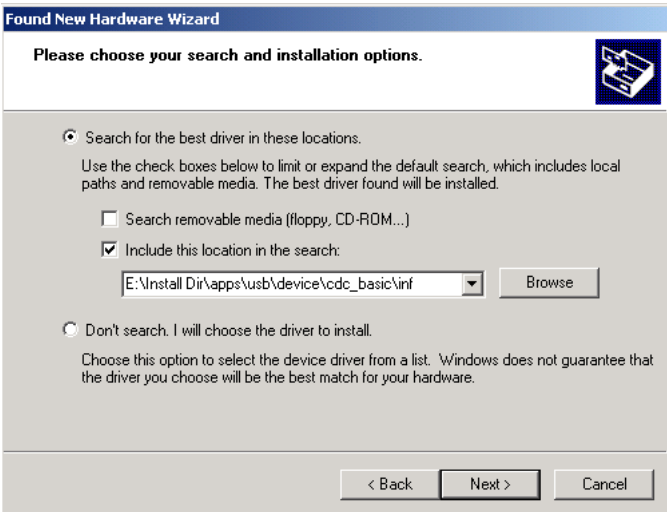
Provides instructions on how to build and run the CDC single COM port demonstration.

Description

This demonstration allows the device to appear like a serial (COM) port to the host. In order to run this demonstration first compile and program the target device. While compiling, select the pic32mx_usb_sk_XXX_XXX configurations for the PIC32M USB Starter Kit II or select the pic32mz_sk_XXX_XXX configurations for PIC32MZ Embedded Connectivity (EC) Starter Kit. Attach the device to the host. If the host is a PC and this is the first time you have plugged this device into the computer then you may be asked for a .inf file.



Select the “Install from a list or specific location (Advanced)” option. Point to the “<Install Directory>/apps/usb/device/cdc_com_port_single/inf” directory.



Once the device is successfully installed, open up a terminal program, such as HyperTerminal. Select the appropriate COM port. On most machines this will be COM5 or higher.

Once connected to the device, there are two ways to run this example project. Typing a key in the terminal window will result in the device echoing the key pressed. So if the user presses “b”, the device will echo “c”. If the pushbutton SW1 is pressed the device will echo “PUSH BUTTON PRESSED” to the terminal window.

Note: Some terminal programs, like HyperTerminal, require users to click the disconnect button before removing the device from the computer. Failing to do so may result in having to close and open the program again in order to reconnect to the device.

3.10.1.5.1.2 cdc_com_port_dual

Demonstrates a USB CDC device, emulating dual serial COM ports - one looping back into the other.

Description

This demonstration application creates a USB CDC Device that enumerates as 2 serial ports on USB Host PC. This application demonstrates the ability of the MPLAB Harmony USB stack to support multiple instances of the same device class.

3.10.1.5.1.2.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.10.1.5.1.2.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Connectivity (EC) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

3.10.1.5.1.2.3 Running the Demonstration

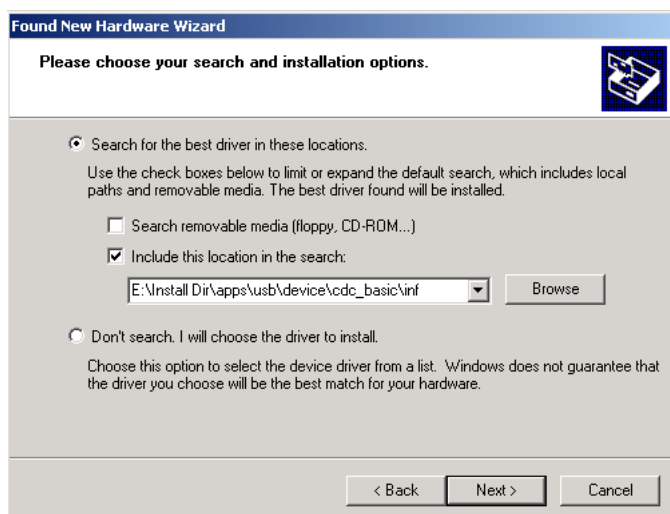
Provides instructions on how to build and run the CDC dual COM port demonstration.

Description

This demonstration allows the device to appear like dual serial (COM) ports to the host. In order to run this demonstration first compile and program the target device. While compiling, select the pic32mx_usb_sk_xxx_xxx configurations for the PIC32 USB Starter Kit II or select the pic32mz_sk_xxx_xxx configurations for PIC32MZ Embedded Connectivity (EC) Starter Kit. Attach the device to the host. If the host is a PC and this is the first time you have plugged this device into the computer then you may be asked for a .inf file.



Select the “Install from a list or specific location (Advanced)” option. Point to the “<Install Directory>\apps\usb\device\cdc_com_port_dual\inf” directory.



Once the device is successfully installed, open up two instances of a terminal program, such as HyperTerminal. Select the appropriate COM port for each of these terminal instance.

To run the demonstration, type a character or string in one terminal window. The same character or string appears on the second terminal window. Similarly, any character typed on second window appears on the first window.

Note: Some terminal programs, like HyperTerminal, require users to click the disconnect button before removing the device from the computer. Failing to do so may result in having to close and open the program again in order to reconnect to the device.

3.10.1.5.1.3 cdc_serial_emulator

This application demonstrates the use of the CDC device class in implementing a USB-to-Serial Dongle.

Description

This application demonstrates the use of the CDC device class in implementing a USB-to-Serial Dongle. The application enumerates a COM port on the PC. Data received through the CDC USB interface is forwarded to a UART. Data received on the UART is forwarded to the CDC USB interface. This emulates a USB-to-Serial Dongle.

3.10.1.5.1.3.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB PIC32MX795F512L Plug-In Module (PIM)	1
Any commercially available USB-to-Serial dongle	-

Note 1: This board cannot be used by itself. It requires an Explorer 16 Development Board (DM240001) and a USB PICtail Plus Daughter Board (AC164131) in order to operate.

3.10.1.5.1.3.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

Explorer 16 Development Board Based Demonstrations

For all of the Explorer 16-based demonstration boards, please follow the following instructions:

1. Connect the USB PICtail Plus Daughter Board to the Explorer 16 Development Board.
2. Short JP2 and JP3 on the USB PICtail Plus Daughter Board
3. Open JP1 and JP4 on the USB PICtail Plus Daughter Board
4. Make sure that S2 on the Explorer 16 Development Board is switched to the "PIM" setting.
5. Short JP2 on the Explorer 16 Development Board to enable the LEDs.
6. Follow any processor specific instructions below. All instructions apply to the PIM unless otherwise stated:
PIC32MX795F512L PIM
7. Open J10.
8. Short pins 1 (marked "USB") and pin 2 (center) of jumpers J1 and J2.

3.10.1.5.1.3.3 Running the Demonstration

Provides instructions on how to build and run the CDC Serial Emulator Demonstration. Please refer to [Release Notes](#) for demonstration limitations, if any.

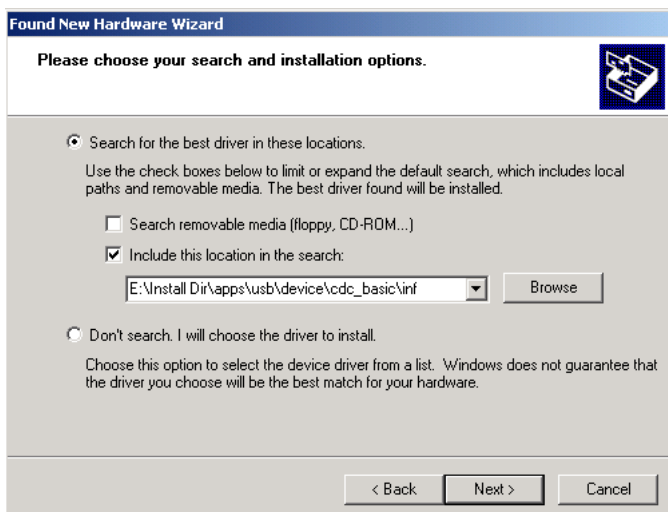
Description

This application demonstrates the use of the CDC device class in implementing a USB to Serial Dongle. The application enumerates a COM port on the PC. Data received through the CDC USB interface is forwarded to a UART. Data received on the UART is forwarded to the CDC USB interface. This emulates a USB-to-Serial Dongle.

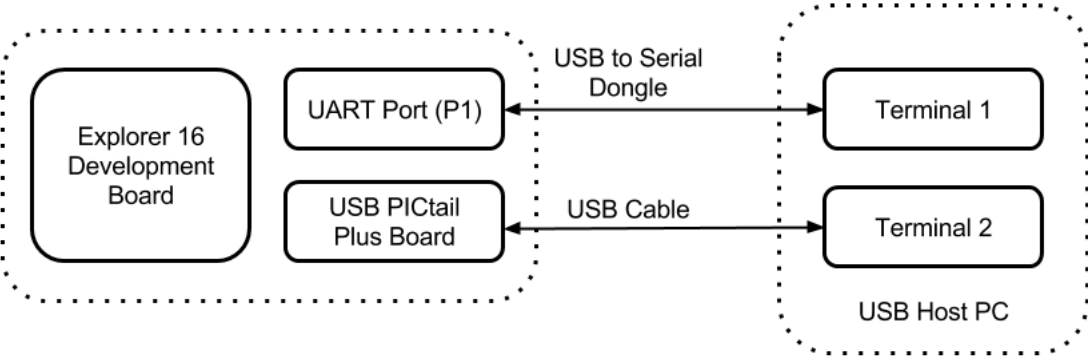
1. Open the project in MPLAB X IDE. Make sure that exp16_pic32mx795f512l_int_dyn configuration is selected.
2. Build the code and program the device.
3. Connect the USB PICtail Plus Daughter Board to the Explorer 16 Development Board.
4. Power up the Explorer 16 Development Board and **connect** a debugger/programmer of your choice to the Explorer 16 Development Board.
5. Build the code and program the device.
6. Connect the mini-B device connector on the USB PICtail Plus Daughter Board to the PC. If the host is a PC and this is the first time you have plugged this device into the computer, you may be asked for a .inf file.



7. Select the “Install from a list or specific location (Advanced)” option. Point to the “<Install Directory>/apps/usb/device/cdc_serial_emulator/inf” directory.



- When enumerated successfully, LED D5 will be the only LED illuminated on the Explorer 16 Development board.
- Open a terminal emulation program of your choice and select the enumerated USB COM port.
 - Connect the USB-to-Serial Dongle to the same PC.
 - Open another instance of the terminal emulation program and select the USB-to-Serial Dongle.
 - Connect the serial connector of the USB-to-Serial Dongle to the UART connector (P1) on the Explorer 16 Development Board.
 - Choose a baud rate of 9600, 1 stop bit and no parity while opening both of the terminal emulation programs.
- The setup should be similar to the following diagram.



Any text entered into the terminal 1 program will be echoed on terminal 2 and vice versa.

3.10.1.5.1.4 hid_basic

This demonstration application creates a custom HID device that can be controlled by a PC based utility.

Description

This application creates a custom HID device that can be controlled by a PC based utility. The device allows the USB Host utility to control the LEDs on the board and query the status of a switch.

3.10.1.5.1.4.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.10.1.5.1.4.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Connectivity (EC) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

3.10.1.5.1.4.3 Running the Demonstration

Provides instructions on how to build and run the HID Basic demonstration.

Description

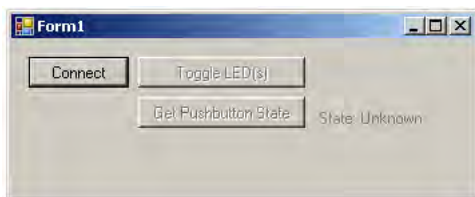
This demonstration uses the selected hardware platform as a HID class USB device, but uses the HID class for general purpose I/O operations. While compiling, select the `pic32mx_usb_sk_xxx_xxx` configurations for the PIC32 USB Starter Kit II or select the `pic32mz_sk_xxx_xxx` configurations for PIC32MZ Embedded Connectivity (EC) Starter Kit. Typically, the HID class is used to implement human interface products, such as mice and keyboards. The HID protocol is however quite flexible, and can be adapted and used to **send**/receive general purpose data to/from a USB device. Using the HID class for general purpose I/O operations is quite advantageous, in that it does not require any kind of custom driver installation process. HID class drivers are already provided by and are distributed with common operating systems. Therefore, upon plugging in a HID class device into a typical computer system, no user installation of drivers is required, the installation is fully automatic.

HID devices primarily communicate through one interrupt IN endpoint and one interrupt OUT endpoint. In most applications, this effectively limits the maximum achievable bandwidth for full speed HID devices to 64 kBytes/s of IN traffic, and 64 kBytes/s of OUT traffic (64 kB/s, but effectively “full duplex”).

The `GenericHIDSimpleDemo.exe` program, and the associated firmware demonstrate how to use the HID protocol for basic general purpose USB data transfer.

Before you can run the `GenericHIDSimpleDemo.exe` executable, you will need to have the Microsoft® .NET Framework Version 2.0 Redistributable Package (later versions probably okay, but not tested) installed on your computer. Programs which were built in the Visual Studio® .NET languages require the .NET redistributable package in order to run. The redistributable package can be freely downloaded from Microsoft's website. Users of Windows Vista® operating systems will not need to install the .NET framework, as it comes pre-installed as part of the operating system.

To launch the application, simply double click on the executable “`GenericHIDSimpleDemo.exe`” in the “<Install Directory>\apps\usb\device\hid_basic\bin” directory. A window like that shown below should appear:



If instead of this window, an error message pops up while trying to launch the application, it is likely the Microsoft .NET Framework Version 2.0 Redistributable Package has not yet been installed. Please install it and try again.

In order to begin sending/receiving packets to the device, you must first find and “**connect**” to the device. As configured by default, the application is looking for HID class USB devices with VID = 0x04D8 and PID = 0x003F. The device descriptor in the firmware project meant to be used with this demonstration uses the same VID/PID. If you plug in a USB device programmed with the correct pre-compiled .hex file, and hit the “Connect” button, the other push buttons should become enabled. If hitting the **connect** button has no effect, it is likely the USB device is either not connected, or has not been programmed with the correct firmware.

Hitting the Toggle LED(s) should **send** a single packet of general purpose generic data to the HID class USB peripheral device. The data will arrive on the interrupt OUT endpoint. The firmware has been configured to receive this generic data packet, parse the packet looking for the “Toggle LED(s)” command, and should respond appropriately by controlling the LED(s) on the demonstration board.

The “Get Pushbutton State” button will [send](#) one packet of data over the USB to the peripheral device (to the interrupt OUT endpoint) requesting the current pushbutton state. The firmware will process the received Get Pushbutton State command, and will prepare an appropriate response packet depending upon the pushbutton state.

Following table shows the button that has to be pressed on the demonstration board to see the push button state changing.

Demonstration Board	Button
PIC32 USB Starter Kit II	SW1

3.10.1.5.1.5 hid_mouse

Demonstrates a USB HID device, emulating a mouse pointing device.

Description

This demonstration application creates a USB HID based 2 button mouse device. When connected, the device emulates mouse operation by moving the cursor in a circular pattern.

3.10.1.5.1.5.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.10.1.5.1.5.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

3.10.1.5.1.5.3 Running the Demonstration

Provides instructions on how to build and run the HID Mouse Demonstration.

Description

This demonstration uses the selected hardware platform as a USB mouse. While compiling, select the `pic32mx_usb_sk_xxx_xxx` configurations for the PIC32 USB Starter Kit II or select the `pic32mz_sk_xxx_xxx` configurations for PIC32MZ Embedded Connectivity (EC) Starter Kit. Before connecting the board to the computer through the USB cable please be aware that the device will start moving the mouse cursor around on the computer. There are two ways to stop the device from making the cursor to continue to move. The first way is to disconnect the device from the computer. The second is to press the correct button on the hardware platform. Pressing the button again will cause the mouse cursor to start moving in a circle again.

Following table shows the button that has to be pressed on the demonstration board to stop the circular motion:

Demonstration Board	Button
PIC32 USB Starter Kit II	SW1

3.10.1.5.1.6 hid_keyboard

Demonstrates a USB HID device, emulating a keyboard.

Description

This demonstration application creates a Generic HID keyboard. Pressing a key on the board emulates a keyboard key press.

3.10.1.5.1.6.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.10.1.5.1.6.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

3.10.1.5.1.6.3 Running the Demonstration

Provides instructions on how to build and run the USB HID Keyboard demonstration.

Description

This demonstration uses the selected hardware platform as a USB keyboard. While compiling, select the `pic32mx_usb_sk_XXX_XXX` configurations for the PIC32 USB Starter Kit II or select the `pic32mz_sk_XXX_XXX` configurations for PIC32MZ Embedded Connectivity (EC) Starter Kit. Before pressing the button, select a window in which it is safe to type text freely. Pressing the button on the demonstration board will cause the device to print a character on the screen.

Following table shows the button that has to be pressed on the demonstration board to print a character.

Demonstration Board	Button
PIC32 USB Starter Kit II	SW1

3.10.1.5.1.7 hid_joystick

Demonstrates a USB HID device emulating a joystick.

Description

This demonstration application creates a custom HID joystick. This application is only intended to demonstrate creation of Joystick HID Report descriptors and may not be a definite end solution. The end application requirements may need the report descriptor to be modified.

3.10.1.5.1.7.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.10.1.5.1.7.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

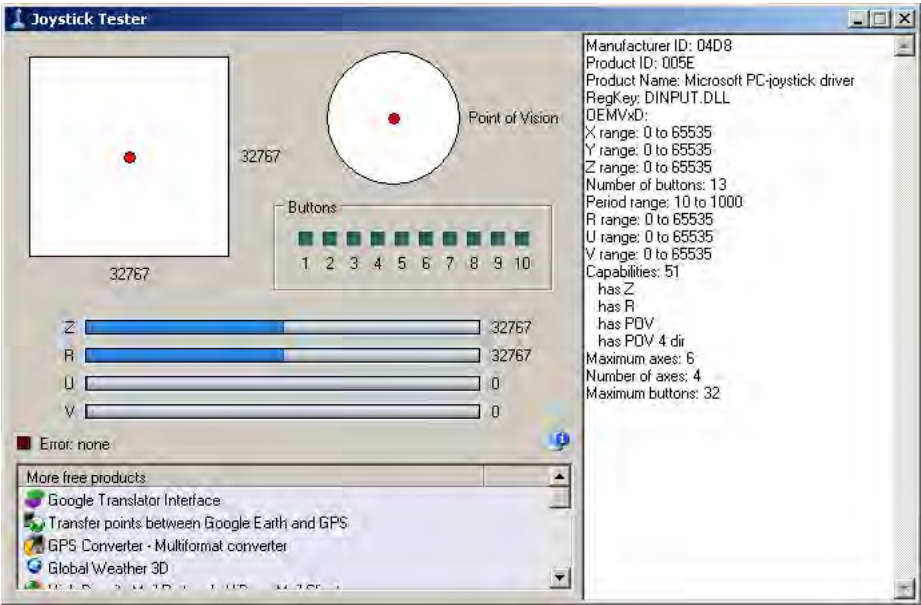
No hardware related configuration or jumper setting changes are necessary.

3.10.1.5.1.7.3 Running the Demonstration

Provides instructions on how to build and run the USB HID Joystick demonstration.

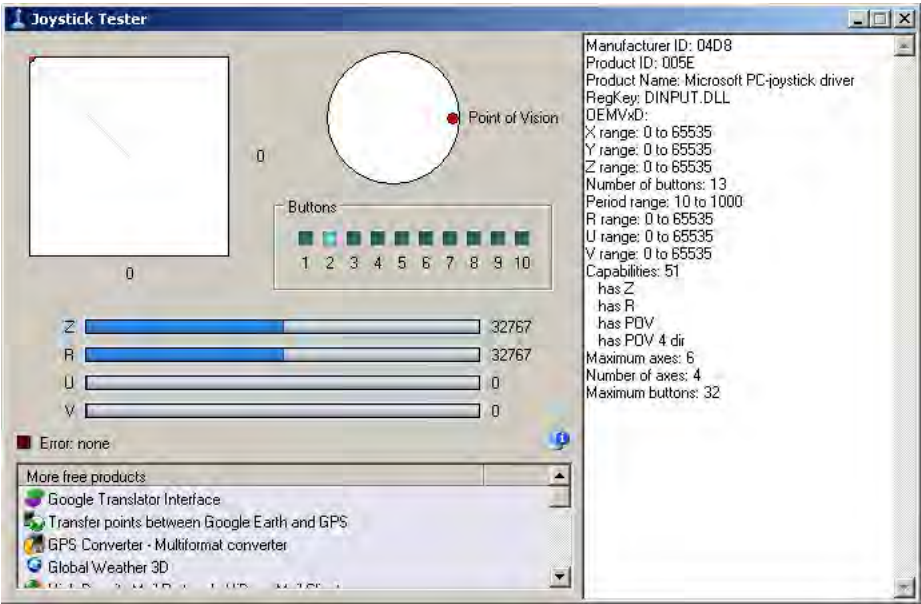
Description

This demonstration uses the selected hardware platform as a USB Joystick. While compiling, select the pic32mx_usb_sk_XXX_XXX configurations for the PIC32 USB Starter Kit II or select the pic32mz_sk_XXX_XXX configurations for PIC32MZ Embedded Connectivity (EC) Starter Kit. To test the joystick feature, go to the "<Install Directory>\apps\usb\device\hid_joystick\bin" directory and open the JoystickTester.exe:



Pressing the button will cause the device to:

- Indicate that the “x” button is pressed, but none others;
- Move the hat switch to the "east" position;
- Move the X and Y coordinates to the their extreme values;



Following table shows the button that has to be pressed on the demonstration board to emulate the joystick.

Demonstration Board	Button
PIC32 USB Starter Kit II	SW1
PIC32 USB Starter Kit	SW1

3.10.1.5.1.8 msd_basic

Demonstrates a USB MSC Device emulating a Flash Drive.

Description

This demonstration application creates a flash drive using the Mass Storage Device Class.

3.10.1.5.1.8.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-

3.10.1.5.1.8.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

3.10.1.5.1.8.3 Running the Demonstration

Provides instructions on how to build and run the USB MSD Flash Drive demonstration.

Description

This demonstration uses the selected hardware platform as an drive on the computer using the internal flash of the device as the drive storage media. Connect the hardware platform to a computer through a USB cable. The device should appear as a new drive on the computer named "Drive Name". The volume label or file information can be changed in the Files.c file located in the project directory.

3.10.1.5.1.9 generic_device

Demonstrates a custom USB Device created by using the Generic Device Library.

Description

This demonstration application creates a custom USB device using the MPLAB Harmony generic device function driver.

3.10.1.5.1.9.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Starter Kit II	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	=

3.10.1.5.1.9.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

[PIC32 USB Starter Kit II](#)

No hardware related configuration or jumper setting changes are necessary.

[PIC32MZ Embedded Connectivity \(EC\) Starter Kit](#)

No hardware related configuration or jumper setting changes are necessary.

3.10.1.5.1.9.3 Running the Demonstration

Provides instructions on how to build and run the Generic Device demonstration.

Description

This demonstration uses the selected hardware platform as a Libusb class USB device. Libusb-Win32 is a USB Library for the Windows operating systems. The library allows user space applications to access any USB device on Windows in a generic way without writing any line of kernel driver code. This driver allows users to have access to interrupt, bulk, and control transfers directly.

The SimpleLibUSBDemo.exe program and the associated firmware demonstrate how to use the Libusb device drivers for basic general purpose USB data transfer. To make the PC source code as easy to understand as possible, the demonstration has deliberately been made simple, and only sends/receives small amounts of data.

Before you can run the SimpleLibUSBDemo.exe executable, you will need to have the Microsoft® .NET Framework Version 3.5 Redistributable Package (later versions probably okay, but not tested) installed on your computer. Programs which were built in the Visual Studio® .NET languages require the .NET redistributable package in order to run. The redistributable package can be freely downloaded from Microsoft's website. Users of Windows Vista® operating systems will not need to install the .NET framework, as it comes pre-installed as part of the operating system.

To launch the application, simply double click on the executable "SimpleLibusbDemo.exe" in the "<Install Directory>\apps\usb\device\generic_device\bin" directory. A window like that shown below should appear:



If instead of this window, an error message pops up while trying to launch the application, it is likely the Microsoft .NET Framework Version 3.5 Redistributable Package has not yet been installed. Please install it and try again.

In order to begin sending/receiving packets to the device, you must first find and "**connect**" to the device. As configured by default, the application is looking for USB devices with VID = 0x04D8 and PID = 0x0204. The device descriptor in the firmware project meant to be used with this demonstration uses the same VID/PID. To run the demonstration program the USB device with the correct precompiled .hex file. If you are connecting the device for the first time, Windows pops up a window asking you

to install the driver for the device. When asked for the driver point it to the inf file provided along with the demonstration. Windows takes while to install the driver for the USB device that is just plugged in. Open the Device manager and ensure that the USB device is listed under the 'Libusb Demo Devices'. Once the driver is installed hit the "Connect" button, the other pushbuttons should become enabled. If hitting the [connect](#) button has no effect, it is likely the USB device is either not connected, or has not been programmed with the correct firmware.

If a different VID/PID combination from the default is desired, then the descriptors in the firmware must be changed as well as the inf file. The easiest way to change the inf file is to use the utility provided with the LibUSB download for windows on the LibUSB website. This utility can create a new inf file based on a connected device. So make sure to change the VID/PID combination first in the firmware, [connect](#) the device, and then run the inf file creator utility. After completing the utility, a new signed driver with inf file is created.

Once the driver is installed hit the "Connect" button, the other pushbuttons should become enabled. If hitting the [connect](#) button has no effect, it is likely the USB device is either not connected, or has not been programmed with the correct firmware.

Hitting the Toggle LED(s) should [send](#) a single packet of general purpose generic data to the Custom class USB peripheral device. The data will arrive on the Bulk OUT endpoint. The firmware has been configured to receive this generic data packet, parse the packet looking for the "Toggle LED(s)" command, and should respond appropriately by controlling the LED(s) on the demonstration board.

The "Get Pushbutton State" button will [send](#) one packet of data over the USB to the peripheral device (to the Bulk OUT endpoint) requesting the current pushbutton state. The firmware will process the received Get Pushbutton State command, and will prepare an appropriate response packet depending upon the pushbutton state.

The PC then requests a packet of data from the device (which will be taken from the Bulk IN endpoint). Once the PC application receives the response packet, it will update the pushbutton state label.

Try experimenting with the application by holding down the appropriate pushbutton on the demonstration board, and then simultaneously clicking on the "Get Pushbutton State" button. Then try to repeat the process, but this time without holding down the pushbutton on the demonstration board.

Following table shows the button that has to be pressed on the demonstration board to see the push button state changing.

Demonstration Board	Button
PIC32 USB Starter Kit II	SW1

3.10.1.5.1.10 audio_speaker

Demonstrates a USB Audio Device that emulates a USB speaker.

Description

This demonstration application creates uses the USB Audio Device class to implement a speaker.

3.10.1.5.1.10.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32 USB Digital Audio Accessory Board	-
PIC32MZ Embedded Connectivity (EC) Starter Kit	-
Multimedia Expansion Board II (MEB II)	-

3.10.1.5.1.10.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Digital Audio Accessory Board

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Connectivity (EC) Starter Kit and Multimedia Expansion Board II (MEB II)

- 1) Fix the PIC32MZ Embedded Connectivity (EC) Starter Kit on the MEB II board.
- 2) Connect a Headphone to the HP OUT jack on the MEB II board.

3.10.1.5.1.10.3 Running the Demonstration

Provides instructions on how to build and run the Audio Speaker Demonstration.

Description

This demonstration functions as a speaker when plugged into a computer. Using any feature on the computer that normal produces sound on the speaker will work with this demonstration. The feature unit only supports the Mute control.

Please note that some applications lock into a sound source when they open or close (such as some web browsers or plug-ins), so that if you plug in the speaker with the web page or video already playing, the sound might not get redirected to the USB based speakers until you close and reopen the browser.

The audio device created in this demonstration has the following characteristics:

- Sampling rate of 48 kHz
- 2 Channel (Stereo)
- PCM Format - 16 bits per Sample
- Asynchronous Audio Endpoint

3.10.1.5.2 Host

3.10.1.5.2.1 cdc_basic

This application demonstrates the use of the CDC Host Class Driver to enumerate and operate a CDC Device.

Description

This application demonstrates the use of the CDC Host Class Driver to enumerate and operate a CDC Device. The application uses the USB Host layer and CDC class driver to enumerate a CDC USB device. The demonstration host application then operates and uses the functionality of the attached CDC Device.

3.10.1.5.2.1.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-
PIC32 USB Starter Kit II	-
Hardware required for USB Device CDC Serial Emulator Demonstration Application	-

3.10.1.5.2.1.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description**PIC32MZ Embedded Connectivity (EC) Starter Kit**

This hardware does not require any configuration.

PIC32 Starter Kit USB II

This hardware does not require any configuration.

3.10.1.5.2.1.3 Running the Demonstration

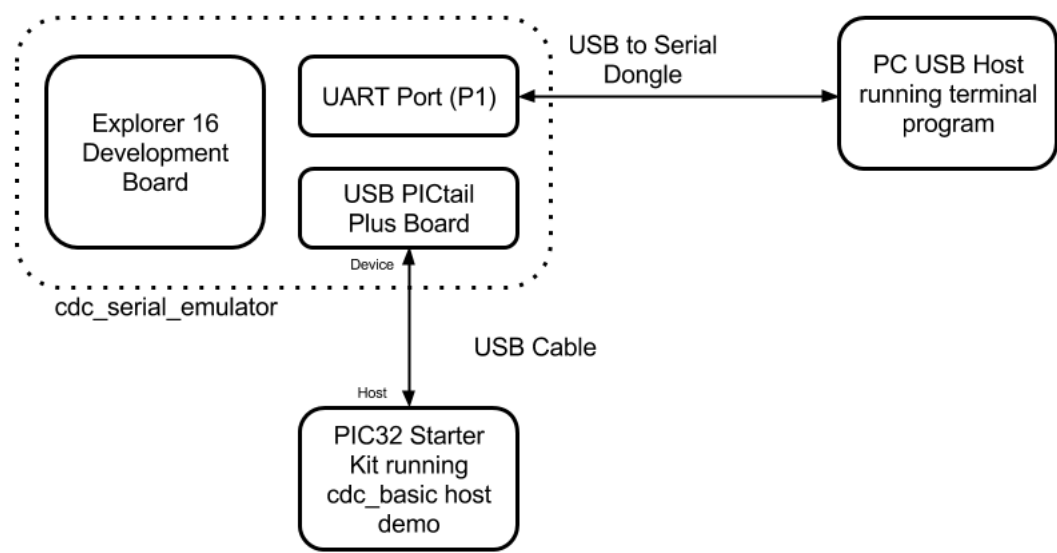
Provides instructions on how to build and run the USB Host CDC Basic Demo. Please refer to the [Release Notes](#) for demonstration limitations, if any.

Description

This application demonstrates the use of the CDC Host Class Driver to enumerate and operate a CDC Device. The application uses the USB Host layer and CDC class driver to enumerate a CDC USB device. The demonstration host application then operates and uses the functionality of the attached CDC Device.

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. Follow the directions for setting up and running the [cdc_serial_emulator](#) USB device demonstration.
4. Connect the UART (P1) port on the Explorer 16 Development Board (running the [cdc_serial_emulator](#) demonstration) to a USB Host PC via a commercially available Serial-to-USB Dongle.
5. Start a terminal program on the USB Host PC and select the Serial-to-USB Dongle as the communication port. Select the baud rate as 9600, no parity, 1 stop bit and no flow control.
6. Connect the mini – B connector on the USB PICtail Plus Daughter Board , of the [cdc_serial_emulator](#) demonstration setup, to the Type – A USB host connector on the starter kit.
7. A prompt ("LED : ") will be displayed immediately on the terminal emulation program.
8. Pressing either the 1, 2, or 3 key on the USB Host keyboard will cause LED 1, 2, or 3 on the PIC32 Starter kit (running the USB CDC Host application) to switch on, respectively.
9. The prompt will again be displayed on terminal emulation program, and step 8 can be repeated.

The setup should be similar to the following diagram.



The `cdc_serial_emulator` demonstration emulates a USB-to-Serial Dongle. The CDC Host (running the `cdc_basic` demonstration application) sends the prompt message to the CDC device. The CDC device forwards the prompt to the UART port from where it is transmitted to the PC USB Host through the USB-to-Serial Dongle. A key press on the PC USB Host is transmitted to the CDC device, which in turn presents the key press data to the CDC host. The `cdc_basic` demonstration then analyzes the key press data and switches on the respective LED.

3.10.1.5.2.2 msd_basic

This application demonstrates the use of the MSD Host Class Driver to write a file to USB Flash Drive.

Description

This application demonstrates the use of the MSD Host Class Driver to write a file to a USB Flash drive. The application uses the USB Host layer , MSD class driver and the MPLAB Harmony File System Framework to enumerate a USB Flash drive and to write a file to it.

3.10.1.5.2.2.1 Supported Demonstration Boards

Lists supported demonstration and development hardware boards.

Description

Demonstration Board (click link for board information)	Notes
PIC32MZ Embedded Connectivity (EC) Starter Kit	-
PIC32 USB Starter Kit II	-

3.10.1.5.2.2.2 Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ Embedded Connectivity (EC) Starter Kit

This hardware does not require any configuration.

PIC32 Starter Kit USB II

This hardware does not require any configuration.

3.10.1.5.2.2.3 Running the Demonstration

Provides instructions on how to build and run the USB Host MSD Basic demonstration. Please refer to the [Release Notes](#) for demonstration limitations, if any.

Description

This application demonstrates the use of the MSD Host Class Driver to write a file to USB Flash drive. The application uses the USB Host layer, MSD class driver and the MPLAB Harmony File System Framework to enumerate a USB Flash drive and to write a file to it.

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. With the code running, attach a USB Flash drive to the Host connector on the starter kit.
4. The demonstration application will then create a file named "File.txt". It will then write the text "Hello World" to this file, and then close the file.
5. The demonstration will then move to Idle mode, which is indicated when LED 2 on the starter kit lights up.
6. The USB Flash drive can then be attached to a USB Host PC to verify the demonstration application operation.
7. Steps 3 through 6 can be repeated.
8. If the USB Flash drive already contains a file with the name "File.txt", the demonstration application will append the text "Hello World" to the end of the file name.

3.10.1.6 Demonstration Board Information

Information on Hardware used by the USB Library Demo Applications.

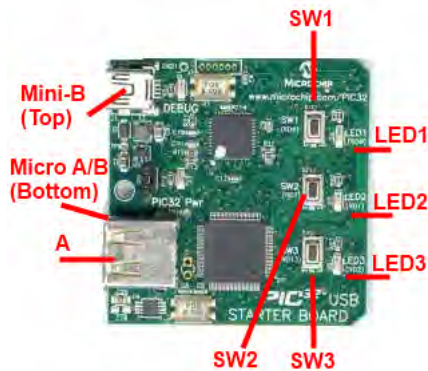
Description

The USB Library Demo Applications can be tried across on hardware boards. This section provide details on the different hardware boards that are used by the demo applications.

3.10.1.6.1 PIC32 USB Starter Kit II

Provides details information on the PIC32 USB Starter II.

Description



SW1 - Application switch. Tied to RD6.

SW2 - Application switch. Tied to RD7.

SW3 - Application switch. Tied to RD13.

LED1 - Application LED. Tied to RD0.

LED2 - Application LED. Tied to RD1.

LED3 - Application LED. Tied to RD2.

This board supports the following USB Device demonstrations:

- CDC Single COM port ([cdc_com_port_single](#)).
- CDC Dual COM port ([cdc_com_port_dual](#))
- HID Basic ([hid_basic](#))
- HID Mouse ([hid_mouse](#))
- HID Keyboard ([hid_keyboard](#))
- HID Joystick ([hid_joystick](#))
- MSD Flash Drive ([msd_basic](#))
- Generic USB Device ([generic_device](#))

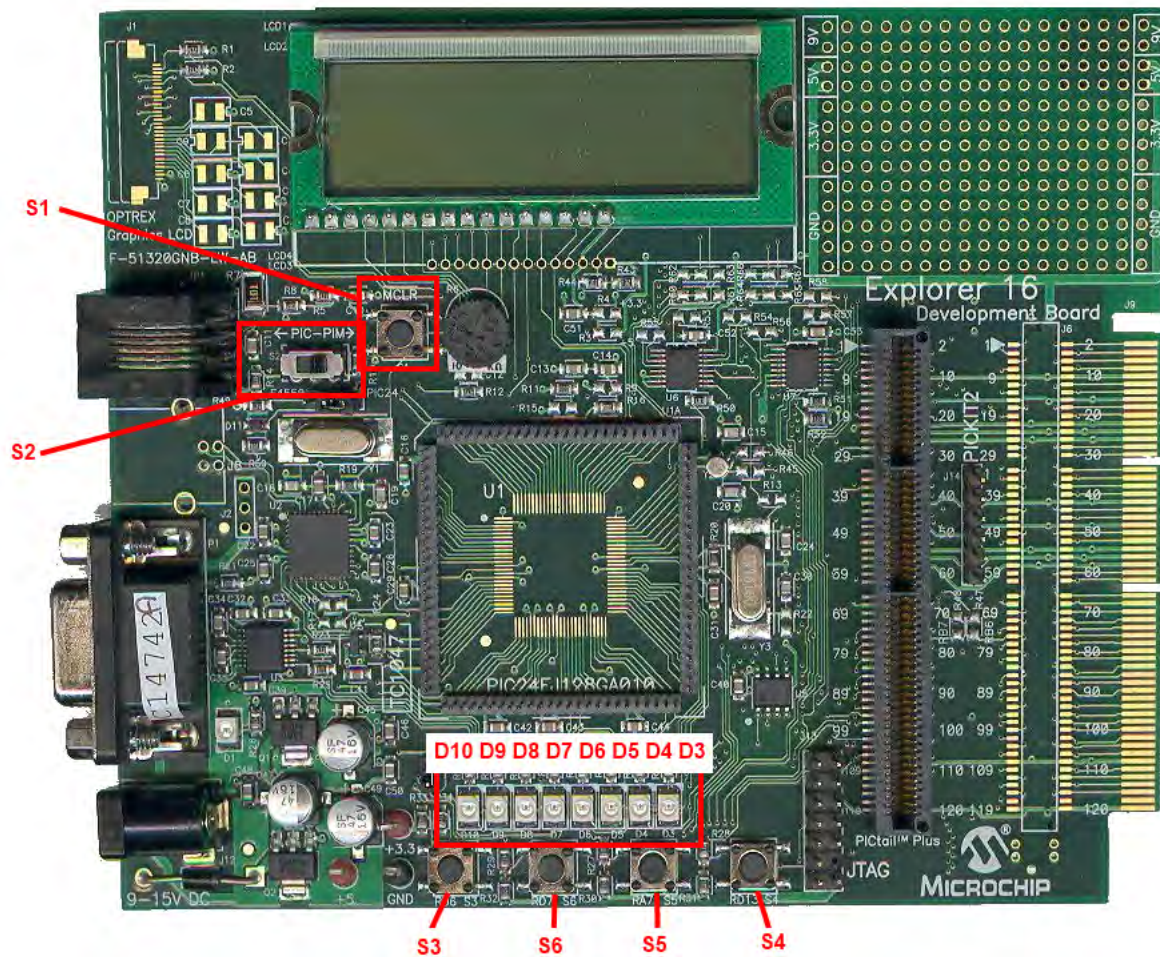
More Information

Microchip Part Number : DM320003-2

3.10.1.6.2 Explorer 16 Development Board

Provides information on the Explorer 16 Development Board

Description



S1 - Reset button (MCLR)

S2 - Processor switch. This switch determines which processor is running, the processor on the board or the processor on the Plug-In-Module (PIM).

S3, S4, S5, S6 - Application switches. For information about what pin is connected to this switch, please refer to the information for the PIM in use.

D3 through D10 - Application LEDs. For information about what pin is connected to this LED, please refer to the information for the PIM in use.

This board support the following USB Host demos:

1. CDC Host (cdc_serial)

More Information:

Microchip Part Number: DM240001

3.10.1.6.3 PIC32MX795F512L Plug-In-Module (PIM)

Provides information on the PIC32MX795F512L Plug-In-Module (PIM).

Description

This PIM is required while using the [Explorer 16 Development Board](#).

More Information

Microchip Part Number: MA320003

[Information Sheet](#)

3.10.1.6.4 PIC32 USB Digital Audio Accessory Board

Provides information on the PIC32 USB Digital Audio Accessory Board

Description

This board support the following USB Device demonstration:

Audio Speaker ([audio_speaker](#))

More Information:

Microchip Part Number: DM320014

3.10.1.6.5 PIC32MZ Embedded Connectivity (EC) Starter Kit

Provides details information on the PIC32MZ Embedded Connectivity (EC) Starter Kit.

Description

Microchip Part Number: DM320006 (without Crypto) or DM320006-C (with Crypto)

The top assembly of the board includes these key features, as indicated in Figure 1:

1. PIC32MZ2048ECH144 32-bit microcontroller.
2. Green power indicator LED.
3. On-board crystal or oscillator for precision microcontroller clocking (12 MHz).
4. USB connectivity for on-board debugger communications.
5. Green debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 host-based applications.
9. HOST mode power jumper.
10. Daughter board connectors for flexible Ethernet PHY options.
11. 32 kHz oscillator for RTCC and Timer1 (optional).
12. External 2 GB SQI memory for expanded memory application.
13. Jumper for using or disconnecting the on-board debugger.

NOTE: When running self-powered USB device applications, open the jumper JP1 to prevent possibly back-feeding voltage onto the Vbus from one port on the host to another (or from one host to another).

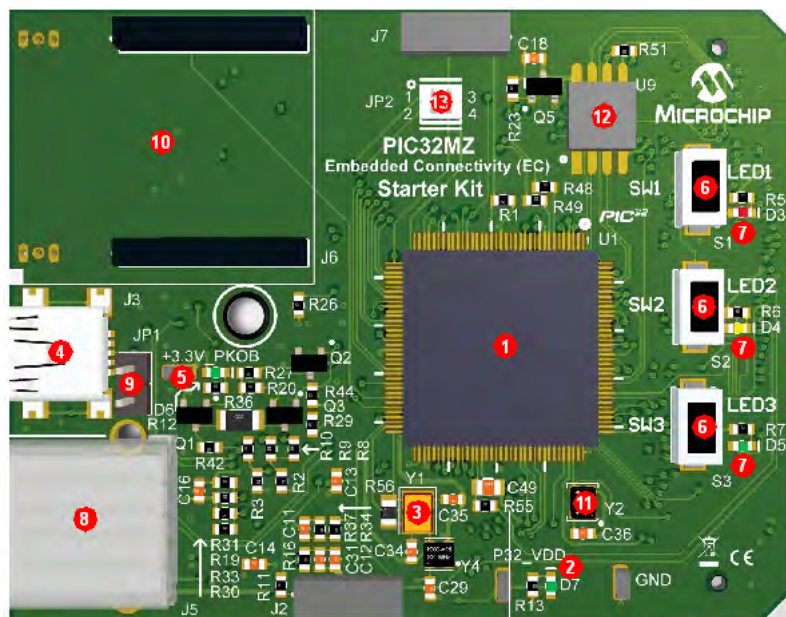


Figure 1: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Front)

The bottom assembly of the board includes these key features, as indicated in Figure 2:

1. PIC24FJ256GB106 USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Type micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.
5. 50 MHz Ethernet PHY oscillator.
6. USB Host and OTG power supply for powering PIC32 USB applications.

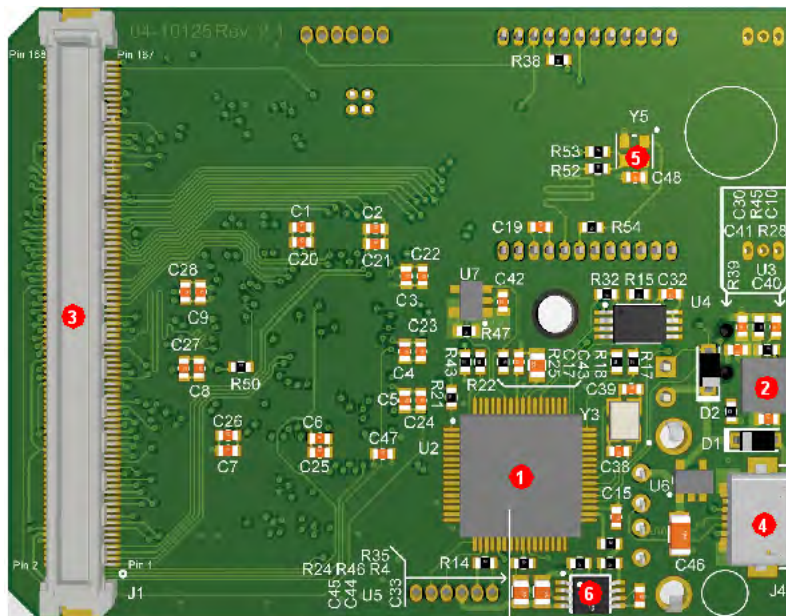


Figure 2: PIC32MZ Embedded Connectivity (EC) Starter Kit Layout (Back)

3.10.1.6.6 Multimedia Expansion Board II (MEB II)

Provides details information on the Multimedia Expansion Board II (MEB) II.

Description

Microchip Part Number: DM320005-2

[Product Page](#)

The front side of the MEB II includes these key features, as shown in Figure 1:

1. Display daughter board connector (60-pin Hirose board-to-board connector).
2. mTouch™ buttons.
3. Push Button.
4. Power LED.
5. User LEDs.
6. VGA Camera (OVM7690).
7. PICtail™ Connector.
8. Analog temperature sensor.

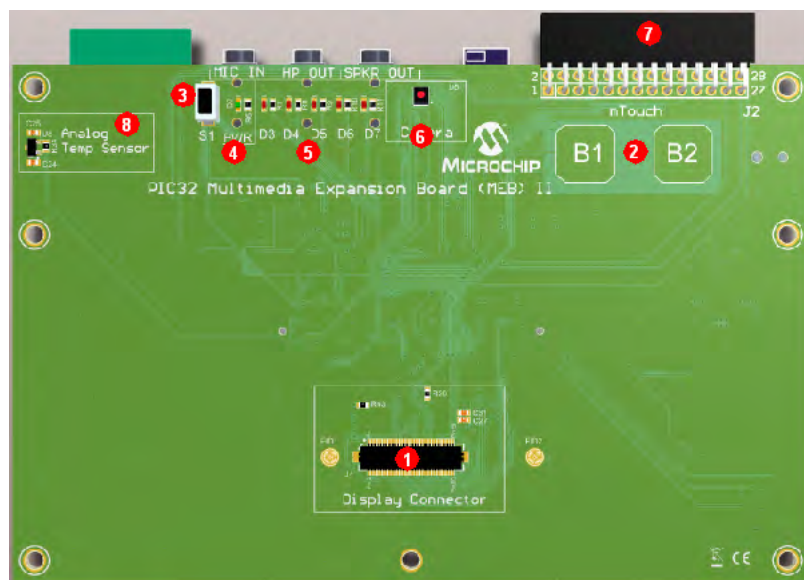


Figure 1: Multimedia Expansion Board II Layout (Front)

The underside of the MEB II includes these key features, as shown in Figure 2:

1. Regulated 5V and 3.3V power supply for powering the board via a 9-15V DC Adapter.
2. PIC32 Starter Kit connector (168-pin Hirose board-to-board connector).
3. 24-bit stereo audio codec (AK4953A).
4. Integrated 802.11bg wireless module (MRF24WG0MA).
5. Low-cost Bluetooth® HCI transceiver (BTM805).
6. EBI SRAM memory (IS61WV102416BLL).
7. microSD slot.

8. Analog accelerometer (ADXL325).

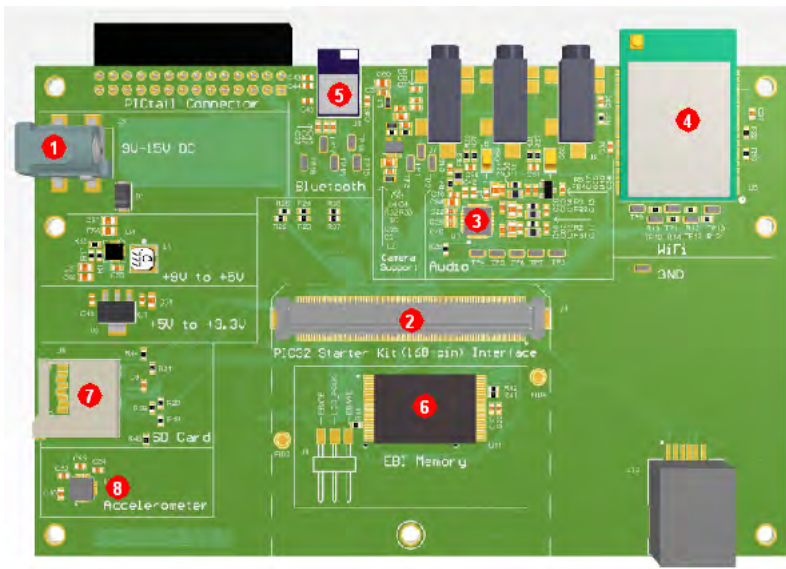


Figure 2: Multimedia Expansion Board II Layout (Back)

4 Building Projects Help

4.1 Building Projects

4.1.1 Introduction

This section describes the MPLAB X IDE projects that are provided to build binary (.a file) versions of key MPLAB Harmony libraries.

Description

The following MPLAB Harmony libraries are provided in prebuilt binary (".a" file) format because source code is not released or to provide optimal performance for users of the free version of the compiler. The MPLAB X IDE projects used to build these libraries are provided so that you can rebuild these libraries with different build parameters, optimization settings, and/or debug symbols if desired.

Note: Building these libraries with high optimization settings will require a fully licensed version of the necessary Microchip XC compiler.

Build Projects Folder: "<install-root>/build"

Library	Description
framework/crypto	MPLAB Harmony Encryption/Decryption Library
framework/zlib	MPLAB Harmony Compress/Decompression Library
framework/math/dsp	MPLAB Harmony Digital Signal Processing Library
framework/math/libq	MPLAB Harmony Fixed-point Math Library
framework/peripheral	MPLAB Harmony Peripheral Library Build Project

4.1.2 Release Notes

MPLAB Harmony Version: v0.70b **Building Projects:** 0.70b **Release Date:** 18Nov2013

New This Release:

Peripheral Libraries

- Nothing to report in this release

Known Issues:

Peripheral Libraries

- Build configurations were not created for all PIC32 devices. As a workaround, select any configuration and choose a different processor from the Project Properties dialog. This will require you to change the name of the output file to match the name of the selected processor.

4.1.3 Using the Build Projects

This section describes how to use the MPLAB Harmony build projects to generate binary (".a" file) libraries that can be linked into your project.

Description

The following information describes how to use the MPLAB Harmony build projects to generate binary (".a" file) libraries that can be linked into your project.

Prerequisites

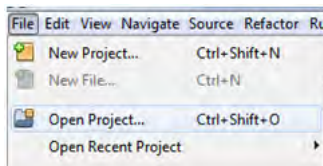
To build MPLAB Harmony libraries, you must have the following installed on your development workstation:

- MPLAB X IDE (v1.95 or higher)
- MPLAB XC32 Compiler (v1.30 or higher)
- Appropriate compiler license for your desired optimization settings

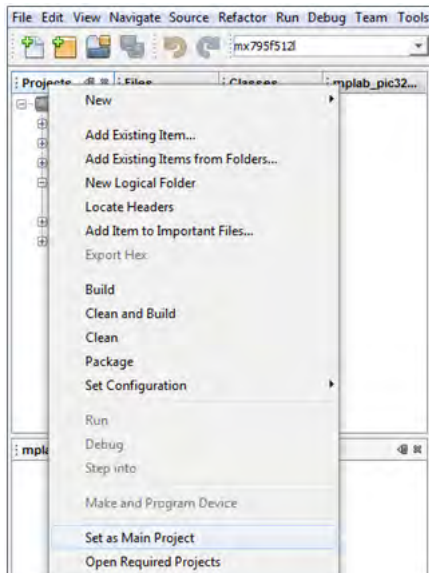
Build Instructions

The following general instructions apply to all MPLAB Harmony library build projects. Refer to the related "Libraries" section for additional information about building specific libraries.

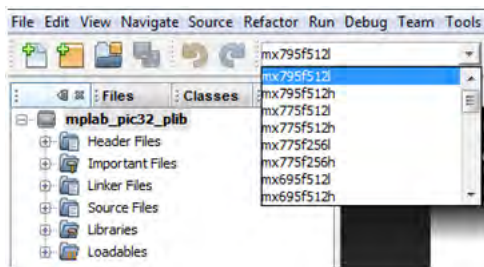
1. Start MPLAB X IDE.
2. Open the desired build project.



3. Set it as the "main" project.



4. Select the desired build configuration.

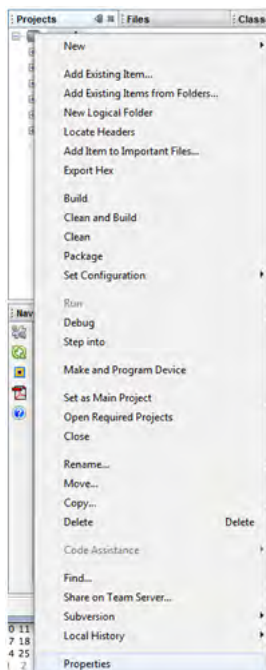


Note: Different libraries provide different build configurations, depending on the set of possible choices that are relevant to that library. For example, the build project for the PIC32MX peripheral libraries (shown above) allows you to select the desired PIC32MX processor. Other libraries may allow you to choose the desired optimization level or feature set. Refer to the related “Libraries” section for information on the configurations provided by the build project for the library you want to build.

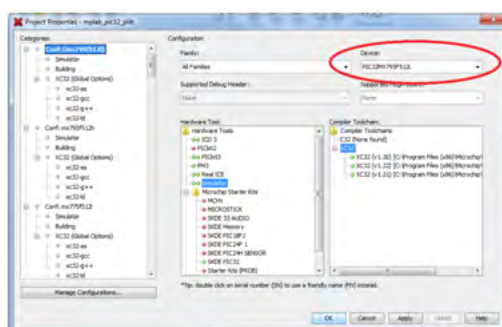
5. If required, select the desired processor (or other project settings, if desired).

Note: Projects that provide processor-specific configurations will not require this step.

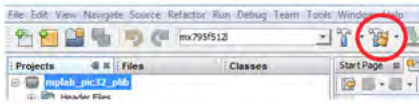
a. Open the “Project Properties” dialog box.



b. Select the desired processor.



6. Build the library.



7. Copy the binary “.a” library file to your project folder.

8. Add the binary “.a” library file to your project.

4.1.4 Supported Libraries

4.1.4.1 Crypto Library

This section describes MPLAB X IDE projects used to build the MPLAB Harmony Crypto libraries.

Description

The Crypto Library project is contained in the build/framework/crypto folder in the crypto.X project. The library contains functions for operations on blocks of data.

Encryption/decryption operations can be done using AES, RSA, DES, Triple DES, and ECC.

Authentication/hashing can be done using MD5, SHA-1, SHA-256, SHA-384, and SHA-512.

Password-based key derivation can be done with HMAC.

Compression and decompression can be done using Huffman Encoding.

Random numbers can be created either one at a time or in a block.

4.1.4.2 Compression Library

This section describes MPLAB X IDE projects used to build the MPLAB Harmony compression/decompression “zlib” libraries.

Description

The “zlib” libraries consists of routines for handling gzip (.gz) files, include compression and decompression.

The library also contains functions that provide the Huffman compression available in the Crypto library.

4.1.4.3 DSP Math Library

This section describes MPLAB X IDE projects used to build the MPLAB Harmony DSP math libraries.

Description

The DSP Math Library consists of routines that are optimized in assembly to take advantage of the microAptiv™ core of the PIC32MZ devices. The library operates on fixed point integers, which are scaled to represent floating point numbers. Many functions are available in both 16-bit and 32-bit numerical formats.

The library contains functions for mathematical operations on a vector (or array) of values, complex scalar values and matrixes.

Operations include add, subtract, multiply, power, data generation, and in the cases of vectors more complex functions including statistics.

The DSP Math Library also contains a full set of digital filtering functions which include FIR and IIR primitives, as well as more complex architectures (e.g., cascade and parallel bi-quad structures).

Finally the library contains a number of transform functions include FFT and inverse FFT, as well as a number of windowing functions.

Some functions in the DSP Math Library require the use of the LIBQ Fixed Point Math library as well. In those cases the LIBQ library must also be installed with your project. Details on dependent functions are found in the DSP Math Library help file under the remarks section for that specific function.

4.1.4.4 Fixed-Point Math Library

This section describes MPLAB X IDE projects used to build the MPLAB Harmony fixed-point math “libq” libraries.

Description

The Fixed-Point Math Library consists of routines in optimized assembly for performing advanced mathematical operations on a scalar. This is similar in construction to the floating point equivalent math.h file that is included with the compiler. All functions are performed using fixed-point integer operations and generally available in both high precision and low precision variants. Lower precision functions generally perform faster and may be suited for time critical operations.

4.1.4.5 Peripheral Libraries

This section describes MPLAB X IDE projects used to build the MPLAB Harmony peripheral libraries.

Description

The MPLAB Harmony peripheral libraries are implemented almost entirely using C-language inline functions. This is done for efficiency so that the compiler can generate a few simple instructions in place of multiple layers of function calls when function parameters are passed as constants. However, if the project that uses the peripheral library is built with low optimization settings, the compiler may generate a function call instead of “inlining” the function implementation. Unfortunately, this will cause an “undefined symbol” error at link time if the linker cannot find an actual implementation of the function to which it can link the “call”.

To satisfy the linker (and to provide optimized PLIB operation, even when low compiler optimization settings are used in your project), the MPLAB Harmony peripheral libraries must be prebuilt as ‘.a’ file, linkable libraries. The appropriate MPLAB X IDE peripheral library “.a” file for the processor in use must be added to the MPLAB X IDE project so that the linker can use them.

Note: Prebuilt binary “.a” files are provided for all supported processors in the MPLAB Harmony installation in the following folder: <install-dir>/bin/framework/peripheral.

The MPLAB Harmony peripheral library “.a” file build projects are provided so that you may rebuild the binary peripheral library “.a” files using any desired optimization settings. However, if you do this, you must have the appropriate xC compiler and you must copy the resultant “.a” file to your project folder and add it to your MPLAB X IDE project. (See the “Build Instructions” section for details of how to build the library.).

Build Configurations

Different MPLAB X IDE projects are provided for different processor families to build the peripheral libraries for MPLAB Harmony. Each project provides a set of MPLAB X IDE configurations, one for each supported processor. For example, the “mplab_pic32mx_plib.X” project provides configurations for each device in the PIC32MX family. To identify which devices are supported, open the MPLAB X IDE project that is named after the device family in which you are interested, select the MPLAB X IDE configuration named after the device you want to use, and look at the project properties to verify the device that is selected.

Key Configuration Options

To build the peripheral library binary “.a” files, there are two key configuration options that must be defined, as follows:

```
#define INLINE_API extern  
#define INLINE static inline
```

These two configuration items are defined in the source file (“peripehral.c”), which then directly includes the peripheral library implementation headers.

Note: This has already been done for you. This information is provided strictly for reference.

During normal peripheral library use, both of these options are defined as “extern inline” so that the compiler can choose between generating a function call or generating inline code, directly in the calling function (as described above). However, to build the binary library “.a” files the peripheral library Application Program Interface (API) functions must be exposed as global “external” symbols. The “INLINE_API” attribute is placed as an attribute on all PLIB API function implementations. So, defining it as “extern” allows the compiler to expose these functions within the library “.a” file so that the linker can find them. The VREG and other “internal” functions prefixed by the “INLINE” attribute are then defined as “static inline” functions for efficiency.

5 Framework Help

5.1 Driver Library Help

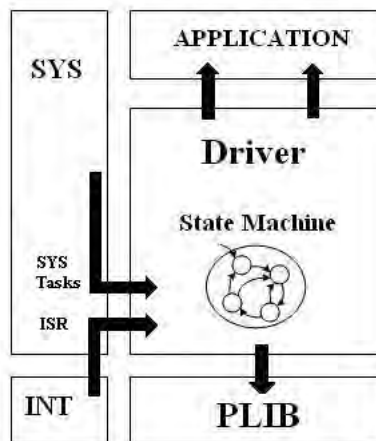
5.1.1 Driver Library Overview

A device driver provides a simple well-defined interface to a hardware peripheral that can be used without operating system support or that can be easily ported to a variety of operating systems. The basic driver operations allow an application to interact with a device, reading and writing data, as if it was a simple file. More specific operations are present on most drivers and the kind of specific operations available depends on the peripheral whose functionality is being exposed by the driver. A driver has the following fundamental responsibilities:

1. Providing a highly-abstracted interface to a peripheral
2. Controlling access to a peripheral
3. Managing the state of a peripheral

The diagram below illustrates how a driver interacts with the other pieces of the system.

- The application calls the well defined driver interface to use the services provided by the driver.
- The driver calls various system services to perform the tasks that are possibly shared across other drivers.
- The driver also calls the peripheral library of the peripheral that its abstracting the interface to.
- Driver State machine can be invoked by the system task service (polling system) or the driver state machine can be invoked from an ISR.



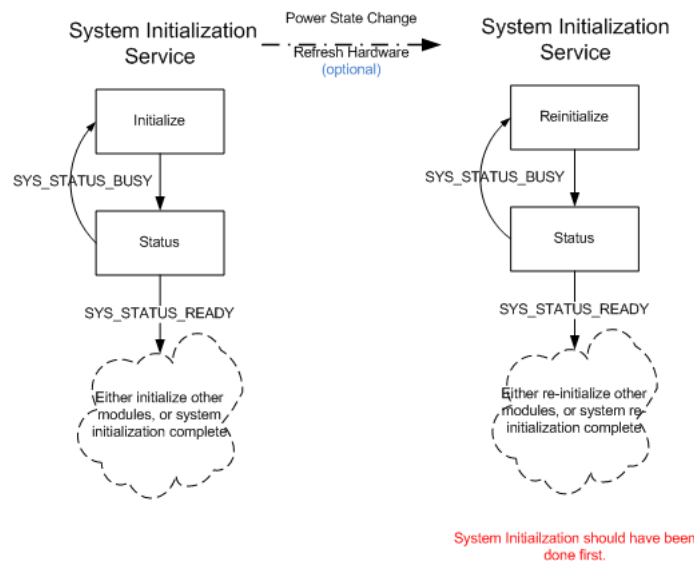
The driver provides the interfaces which can be broken down into the following two categories : (*<mod>* is the abbreviation identifying the module)

- **System Operation** - The interfaces for the system operation should be called by the system. Each driver can support all of the system interfaces, or it can choose to not support the optional system interfaces. The system interfaces are
 - **DRV_<mod>_Initialize** - This routine initializes hardware for the index instance of the <module> module, using the hardware initialization given data. It also initializes any internal data structures. The DRV_<mod>_Status operation will return SYS_STATUS_READY when this operation has completed. Every driver module should define its own initialization

data structure type named "DRV_<mod>_INIT". This structure must be an extension of the [SYS_MODULE_INIT](#) structure (i.e. its first member must be the [SYS_MODULE_INIT](#) structure or equivalent). Any parameter that can change the power state of the module must be included in the data structure. Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

- **DRV_<mod>_Reinitialize** - This routine reinitializes and refreshes the hardware using the hardware initialization given data. The DRV_<mod>_Status operation will return SYS_STATUS_READY when this operation has completed. It does not clear or reinitialize internal data structures (although it may change the value of a few appropriate data items necessary to manage the new hardware state) and it does not disconnect or interrupt any ongoing client operations. This operation can be used to change the power state of the peripheral the module manages.
- **DRV_<mod>_Deinitialize** - This routine de-initializes the index instance of the <module> module, disabling its operation (and any hardware for driver modules). The DRV_<mod>_Status operation will return SYS_STATUS_READY when this operation has completed.
- **DRV_<mod>_Status** - The Status operation can be used to determine when any of the other three module operations has completed. If the Status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the Status operation returns "SYS_STATUS_READY", any previous operations have completed.
 - The value of SYS_STATUS_READY is 1. Values between 1 and 10 are reserved for system defined "ready" states. A module may define module-specific "ready" or "run" states greater than or equal to 10 (SYS_STATUS_READY_EXTENDED).
 - The value of SYS_STATUS_ERROR is -1. Values between -1 and -10 are reserved for system-defined errors. A module may define module-specific error values of less than or equal to -10 (SYS_STATUS_ERROR_EXTENDED).
 - If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations. (Remember to check the value returned by the Status routine after calling any of the module operations to find out when they have completed.)
- **DRV_<mod>_Tasks** - Used to maintain the driver's state machine and implement its ISR. It is normally only called by the system's tasks routine or by an Interrupt Service Routine (ISR).

The usage model of the system interfaces is demonstrated in the diagram below:

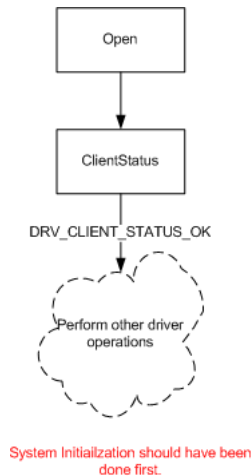


The system is also responsible for either calling the Tasks routine through an ISR or poll the Tasks routine in a polling environment.

- **Client Operation** - The interfaces provided by the driver for client operation, can be called by the application. Each driver can provide a set of client interfaces, which supports the operation model of the peripheral that the driver is supporting. The client interfaces can include

- **DRV_<mod>_Open** - This routine opens a driver for use by any client module and provides an “open-instance” handle that must be provided to any of the other driver operations to identify the caller and the instance of the driver/hardware module.
- **DRV_<mod>_Close** - This routine closes an opened-instance of a driver, invalidating the given handle. This routine is optional, if the driver is designed to never be closed.
- **DRV_<mod>_ClientStatus** - This routine provides the client, the status of the driver.
- **DRV_<mod>_<operation>** - This is the basic format of a driver interface routine. Driver interface routines (other than the system module routines and the basic device driver routines) should follow this format.

The general usage model of the client interfaces is demonstrated in the diagram below, each driver will document usage model specific to the driver in the driver help file.



Driver Configuration

A driver provides a set of configurations which can be divided into following categories:

1. **Build Configuration Selection Configurations** - The driver supports selecting the driver instance at build time (static configuration) or at run time (dynamic configuration). A driver also supports selecting the number of clients that can possibly connect to an hardware instance. All drivers will provide DRV_<mod>_DRIVER_OBJECTS_NUMBER, when this configuration macro is defined driver is built in dynamic driver configuration, else it is built in static driver configuration. The number assigned to DRV_<mod>_DRIVER_OBJECTS_NUMBER controls the maximum number of driver objects that can be used. If a driver supports multiple clients it will provide DRV_<mod>_CLIENT_OBJECTS_NUMBER, when this configuration macro is defined driver is built to support multiple clients, else the driver is built to support single client. The number assigned to DRV_<mod>_DRIVER_OBJECTS_NUMBER controls the maximum number of client objects that can be used.
2. **Initialization Overrides** - Initialization overrides are provided to enable configuration of the hardware statically and with low run time overhead. These override the parameters if also passed dynamically to Initialize or Reinitialize routine.
3. **Other operational configurations** - These can control the functionality of the driver or other setting of the driver that are required for support core or optional functionality.

A driver allows the application to use the dynamic interface for all the above configuration when the application chooses the appropriate configurations at build time. This is the preferred method of usage for the drivers.

As an exception, if the application wants to use multiple configurations in the same build for a driver type, or it needs to use multiple instances of the same driver type, it needs to directly use the static single - client configuration interfaces, or it needs to use the static multiple - client configuration interfaces.

While using the driver the static single client driver configuration, the application and system can choose to ignore the SYS_MODULE_INDEX, SYS_MODULE_OBJ and DRV_HANDLE passed and returned to the driver and system interfaces.

While using the driver the static multiple client driver configuration, the application and system can choose to ignore the

[SYS_MODULE_INDEX](#) and [SYS_MODULE_OBJ](#) passed and returned to the driver and system interfaces.

5.1.1.1 Data Types

Enumerations

	Name	Description
	DRV_CLIENT_STATUS	Identifies the current status/state of a client's connection to a driver.
	DRV_IO_BUFFER_TYPES	Identifies to which buffer a device operation will apply.
	DRV_IO_INTENT	Identifies the intended usage of the device when it is opened.

Types

	Name	Description
	DRV_HANDLE	Handle to an opened device driver.

Description

5.1.1.1.1 DRV_CLIENT_STATUS Enumeration

C

```
typedef enum {
    DRV_CLIENT_STATUS_ERROR_EXTENDED = -10,
    DRV_CLIENT_STATUS_ERROR = -1,
    DRV_CLIENT_STATUS_CLOSED = 0,
    DRV_CLIENT_STATUS_BUSY = 1,
    DRV_CLIENT_STATUS_READY = 2,
    DRV_CLIENT_STATUS_READY_EXTENDED = 10
} DRV_CLIENT_STATUS;
```

Description

Driver Client Status

This enumeration identifies the current status/state of a client's link to a driver.

Members

Members	Description
DRV_CLIENT_STATUS_ERROR_EXTENDED = -10	Indicates that a non-system defined error has occurred.
DRV_CLIENT_STATUS_ERROR = -1	An un-specified error has occurred.
DRV_CLIENT_STATUS_CLOSED = 0	An operation is currently in progress
DRV_CLIENT_STATUS_BUSY = 1	An operation is currently in progress
DRV_CLIENT_STATUS_READY = 2	Any previous operations have succeeded and the module is ready for additional operations
DRV_CLIENT_STATUS_READY_EXTENDED = 10	Indicates that the module is in a non-system defined ready/run state.

Remarks

The enumeration used as the return type for the client-level status routines defined by each device driver or system module (for example, [DRV_USART_ClientStatus](#)) must be based on the values in this enumeration.

5.1.1.1.2 DRV_HANDLE Type

C

```
typedef uintptr_t DRV_HANDLE;
```

Description

Device Handle

This handle identifies the open instance of a device driver. It must be passed to all other driver routines (except the initialization, deinitialization, or power routines) to identify the caller.

Remarks

Every application or module that wants to use a driver must first call the driver's open routine. This is the only routine that is absolutely required for every driver.

If a driver is unable to allow an additional module to use it, it must then return the special value `DRV_HANDLE_INVALID`. Callers should check the handle returned for this value to ensure this value was not returned before attempting to call any other driver routines using the handle.

5.1.1.1.3 DRV_IO_BUFFER_TYPES Enumeration

C

```
typedef enum {
    DRV_IO_BUFFER_TYPE_NONE = 0x00,
    DRV_IO_BUFFER_TYPE_READ = 0x01,
    DRV_IO_BUFFER_TYPE_WRITE = 0x02,
    DRV_IO_BUFFER_TYPE_RW = DRV_IO_BUFFER_TYPE_READ | DRV_IO_BUFFER_TYPE_WRITE
} DRV_IO_BUFFER_TYPES;
```

Description

Device Driver IO Buffer Identifier

This enumeration identifies to which buffer (read, write, both, or neither) a device operation will apply. This is used for "flush" (or similar) operations.

Members

Members	Description
DRV_IO_BUFFER_TYPE_NONE = 0x00	Operation does not apply to any buffer
DRV_IO_BUFFER_TYPE_READ = 0x01	Operation applies to read buffer
DRV_IO_BUFFER_TYPE_WRITE = 0x02	Operation applies to write buffer
DRV_IO_BUFFER_TYPE_RW = DRV_IO_BUFFER_TYPE_READ DRV_IO_BUFFER_TYPE_WRITE	Operation applies to both read and write buffers

5.1.1.1.4 DRV_IO_INTENT Enumeration

C

```
typedef enum {
    DRV_IO_INTENT_READ,
    DRV_IO_INTENT_WRITE,
    DRV_IO_INTENT_READWRITE,
    DRV_IO_INTENT_BLOCKING,
    DRV_IO_INTENT_NONBLOCKING,
    DRV_IO_INTENT_EXCLUSIVE,
    DRV_IO_INTENT_SHARED
}
```



```
} DRV_IO_INTENT;
```

Description

Device Driver I/O Intent

This enumeration identifies the intended usage of the device when the caller opens the device. It identifies the desired behavior of the device driver for the following:

- Blocking or non-blocking I/O behavior (do I/O calls such as read and write block until the operation is finished or do they return immediately and require the caller to call another routine to check the status of the operation)
- Support reading and/or writing of data from/to the device
- Identify the buffering behavior (sometimes called "double buffering" of the driver. Indicates if the driver should maintain its own read/write buffers and copy data to/from these buffers to/from the caller's buffers.
- Identify the DMA behavior of the peripheral

Members

Members	Description
DRV_IO_INTENT_READ	Read
DRV_IO_INTENT_WRITE	Write
DRV_IO_INTENT_READWRITE	Read and Write
DRV_IO_INTENT_BLOCKING	The driver will block and will return when the operation is complete
DRV_IO_INTENT_NONBLOCKING	The driver will return immediately
DRV_IO_INTENT_EXCLUSIVE	The driver will support only one client at a time
DRV_IO_INTENT_SHARED	The driver will support multiple clients at a time

Remarks

The buffer allocation method is not identified by this enumeration. Buffers can be allocated statically at build time, dynamically at run-time, or even allocated by the caller and passed to the driver for its own usage if a driver-specific routine is provided for such. This choice is left to the design of the individual driver and is considered part of its interface.

These values can be considered "flags". One selection from each of the groups below can be ORed together to create the complete value passed to the driver's open routine.

5.1.1.2 Constants

Macros

	Name	Description
	<u>DRV_CONFIG_NOT_SUPPORTED</u>	Not supported configuration.
	<u>DRV_HANDLE_INVALID</u>	Invalid device handle.
	<u>DRV_IO_ISBLOCKING</u>	Returns if the I/O intent provided is blocking
	<u>DRV_IO_ISEXCLUSIVE</u>	Returns if the I/O intent provided is non-blocking.
	<u>DRV_IO_ISNONBLOCKING</u>	Returns if the I/O intent provided is non-blocking.

Description**5.1.1.2.1 DRV_CONFIG_NOT_SUPPORTED Macro****C**

```
#define DRV_CONFIG_NOT_SUPPORTED (((unsigned short) -1))
```

Description

Not supported configuration

If the configuration option is not supported on an instance of the peripheral, use this macro to equate to that configuration. This option should be listed as a possible value in the description of that configuration option.

5.1.1.2.2 DRV_HANDLE_INVALID Macro**C**

```
#define DRV_HANDLE_INVALID (((DRV_HANDLE) -1))
```

Description

Invalid Device Handle

If a driver is unable to allow an additional module to use it, it must then return the special value DRV_HANDLE_INVALID. Callers should check the handle returned for this value to ensure this value was not returned before attempting to call any other driver routines using the handle.

Remarks

None.

5.1.1.2.3 DRV_IO_ISBLOCKING Macro**C**

```
#define DRV_IO_ISBLOCKING(intent) (intent & DRV_IO_INTENT_BLOCKING)
```

Description

Device Driver Blocking Status Macro

This macro returns if the I/O intent provided is blocking.

Remarks

None.

5.1.1.2.4 DRV_IO_ISEXCLUSIVE Macro**C**

```
#define DRV_IO_ISEXCLUSIVE(intent) (intent & DRV_IO_INTENT_EXCLUSIVE)
```

Description

Device Driver Exclusive Status Macro

This macro returns if the I/O intent provided is non-blocking.

Remarks

None.

5.1.1.2.5 DRV_IO_ISNONBLOCKING Macro

C

```
#define DRV_IO_ISNONBLOCKING(intent) (intent & DRV_IO_INTENT_NONBLOCKING )
```

Description

Device Driver Non Blocking Status Macro

This macro returns if the I/ intent provided is non-blocking.

Remarks

None.

5.1.1.3 Files

Files

Name	Description
driver.h	This file aggregates all of the driver library interface headers.
driver_common.h	This file defines the common macros and definitions used by the driver definition and implementation headers.

Description

5.1.1.3.1 driver.h

Driver Library Interface Header

This file aggregates all of the driver library interface headers so client code only needs to include this one single header to obtain prototypes and definitions for the interfaces to all driver libraries.

A device driver provides a simple well-defined interface to a hardware peripheral that can be used without operating system support or that can be easily ported to a variety of operating systems.

A driver has the fundamental responsibilities:

- Providing a highly-abstraced interface to a peripheral
- Controlling access to a peripheral
- Managing the state of a peripheral

Remarks

The directory in which this file resides should be added to the compiler's search path for header files.

File Name

drv.h

Company

Microchip Technology Inc.

5.1.1.3.2 driver_common.h

Driver Common Header

This file defines the common macros and definitions used by the driver definition and the implementation header.

Remarks

The directory in which this file resides should be added to the compiler's search path for header files.

Enumerations

	Name	Description
	<u>DRV_CLIENT_STATUS</u>	Identifies the current status/state of a client's connection to a driver.
	<u>DRV_IO_BUFFER_TYPES</u>	Identifies to which buffer a device operation will apply.
	<u>DRV_IO_INTENT</u>	Identifies the intended usage of the device when it is opened.

Macros

	Name	Description
	<u>DRV_CONFIG_NOT_SUPPORTED</u>	Not supported configuration.
	<u>DRV_HANDLE_INVALID</u>	Invalid device handle.
	<u>DRV_IO_ISBLOCKING</u>	Returns if the I/O intent provided is blocking
	<u>DRV_IO_ISEXCLUSIVE</u>	Returns if the I/O intent provided is non-blocking.
	<u>DRV_IO_ISNONBLOCKING</u>	Returns if the I/O intent provided is non-blocking.

Types

	Name	Description
	<u>DRV_HANDLE</u>	Handle to an opened device driver.

File Name

drv_common.h

Company

Microchip Technology Inc.

5.1.2 ADC Driver Library

5.1.2.1 Introduction

ADC Driver for Microchip Microcontrollers

This Analog-to-Digital Converter (ADC) driver provides an interface to manage the ADC module on the Microchip family of microcontrollers.

Description

Overview

An ADC is a vital part of any system that interfaces to real-world signals. While there are many techniques for analog-to-digital

conversion, the Microchip family of microcontrollers uses Successive Approximation as one of its primary techniques.

5.1.2.2 Release Notes

MPLAB Harmony Version: v0.70b **ADC Driver Library Version :** 0.02a **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.1.2.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.2.4 Using the Library

This topic describes the basic architecture of the ADC Driver Library and provides information and examples on how to use it.

Interface Header File: drv_adc.h

The interface to the ADC Driver Library is defined in the "drv_adc.h" header file.

Please refer to the [MPLAB Harmony Overview](#) for how the driver interacts with the framework.

5.1.2.4.1 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

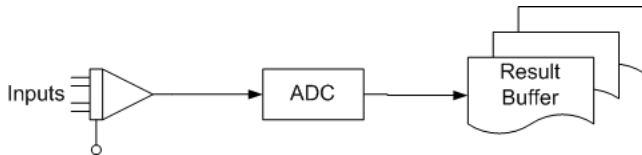
The following table lists the interface section and its brief description.

Section	Description
Configuring the Library	Provides macros for configuring the system. It is required that the system configures the driver to build correctly by choosing appropriate configuration options as listed in this section.

System Interaction Functions	Provides system module interfaces, Device initialization, deinitialization, reinitialization and status functions.
Client Core Configuration Functions	Provides interfaces for core functionality of the driver.

5.1.2.4.2 Abstraction Model

The ADC driver is modeled using the abstraction model, as shown in the following diagram.



- [DRV_ADC_InputsRegister](#) allows selection of inputs
- [DRV_ADC_SamplesRead](#) and [DRV_ADC_SamplesReadLatest](#) allow reading of the sample from the result buffer
- [DRV_ADC_Start](#) and [DRV_ADC_Stop](#) will control the operation of the ADC

5.1.2.4.3 How the Library Works

The library provides interfaces to support:

- System Interaction
- Client Core Functionality

5.1.2.4.3.1 System Initialization

This section describes the system initialization and reinitialization settings for the ADC Peripheral Library.

Description

System Initialization and Reinitialization

The system initialization and the reinitialization settings, effect only the instance of the peripheral that is being initialized or re-initialized. During system initialization configure each instance of the module with the following configuration settings that are supported by the specific ADC device hardware (refer to [DRV_ADC_INIT](#))

1. Device requested power state: One of the system module power states.
2. Acquisition time: If the hardware supports this feature, configure this variable from [DRV_ADC_ACQUISITION_TIME](#).
3. Voltage reference: If the hardware supports this feature, select the one from the available options(from [DRV_ADC_VOLTAGE_REFERENCE](#)).
4. Conversion clock: If the hardware supports this feature, select the conversion clock prescaler value (from [DRV_ADC_CONVERSION_CLOCK_PRESCALER](#)).
5. Conversion clock source: If the hardware supports this feature, select the conversion clock prescaler value (from [DRV_ADC_CONVERSION_CLOCK_SOURCE](#)).
6. Clock frequency: Peripheral clock frequency configured for the device.
7. Conversion trigger source: If the hardware supports this feature, select the conversion trigger source(from [DRV_ADC_CONVERSION_TRIGGER_SOURCE](#)).
8. Samples per interrupt: Configure the number of samples before to be generating interrupt(from

[DRV_ADC_SAMPLES_PER_INTERRUPT](#)).

9. Output data format: Select the output data format.

10. Interrupt source: Select the interrupt source.

Example: Auto Sampling Mode

```
#define MY_ADC_INSTANCE DRV_ADC_INDEX_0

SYS_MODULE_OBJ      myAdcObj;
DRV_ADC_INIT        adcInitData;
SYS_STATUS           adcStatus;

// Populate the adcInitData structure
adcInitData.plibModuleId = ADC_ID_1;
adcInitData.acquisitionTime = PLIB_ADC_ACQUISITION_TIME_20_TAD;
adcInitData.voltageReference = PLIB_ADC_VREF_POS_TO_VDD_VREF_NEG_TO_VSS;
adcInitData.clockFrequency = 4000000; //4MHz
adcInitData.conversionClock = PLIB_ADC_CONV_CLOCK_20_TCY;
adcInitData.conversionClockSource = PLIB_ADC_CLOCK_SRC_SYSTEM_CLOCK;
adcInitData.conversionTriggerSource = PLIB_ADC_CONVERSION_TRIGGER_INTERNAL_COUNT;
adcInitData.dataOutputFormat = PLIB_ADC_OUTPUT_FORMAT_INTEGER_16BIT;
adcInitData.initFlags = DRV_ADC_AUTO_SAMPLING;
adcInitData.interruptSource= PLIB_INT_SOURCE_ADC_1;
adcInitData.samplesPerInterrupt = PLIB_ADC_SAMPLE_PER_INTERRUPT_AT_EACH_SAMPLE;

myAdcObj = DRV_ADC_Initialize(MY_ADC_INSTANCE, (SYS_MODULE_INIT*)&adcInitData);

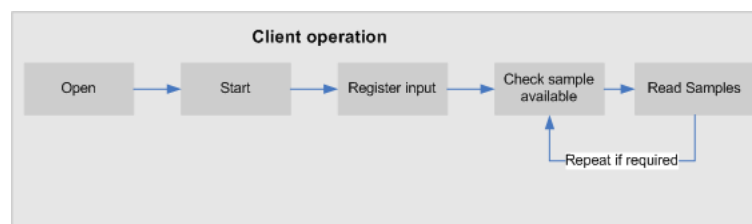
adcStatus = DRV_ADC_Status(myAdcObj);
if (SYS_STATUS_BUSY == adcStatus)
{
    // do something else and check back later
}
else if (SYS_STATUS_ERROR >= adcStatus)
{
    // Handle error
}
```

5.1.2.4.3.2 Client Core Functionality

Core functionality provides a basic interface for the driver operation.

Description

General Client Operation



Core functionality provides a basic interface for the driver operation.

Applications using the timer core functionality, need to perform the following:

1. The system should have completed necessary initialization and [DRV_ADC_Tasks](#) should either be running in a polled environment, or in an interrupt environment.
2. Open the driver using [DRV_ADC_Open](#) (the timer driver supports exclusive access only).
3. Registers the inputs to be used by the clients using [DRV_ADC_InputsRegister](#).

4. Start the driver using [DRV_ADC_Start](#).
5. Poll for the elapse status using [DRV_ADC_SamplesAvailable](#), and then read the samples using either [DRV_ADC_SamplesReadLatest](#) or [DRV_ADC_SamplesRead](#).
6. The client will be able to stop the started ADC instance using [DRV_ADC_Stop](#) at any point and will be able to close it using [DRV_ADC_Close](#) when it is no longer required.

Example:

```

DRV_HANDLE adcHandle;           //Handle returned by DRV_ADC_Open function.
uint16_t dataBuffer;           //Buffer to which the data will be written.

/*
Open a new client. The handle returned by the this function should be used
a passing parameter for all the specific client related operations.
*/
adcHandle = DRV_ADC_Open(MY_ADC_INSTANCE, DRV_IO_INTENT_NONBLOCKING);

DRV_ADC_InputsRegister ( adcHandle , PLIB_ADC_INPUT_AN1|PLIB_ADC_INPUT_AN2 );

/*
ADC is not yet enabled. Enable it once the inputset is registered. Starts automatically
since auto sample mode is enabled.
*/
DRV_ADC_Start(adcHandle);

if ( DRV_ADC_SamplesAvailable(adcHandle) )
{
    DRV_ADC_SamplesReadLatest ( adcHandle, &dataBuffer, 1);

    /*
    Once the above function returns success, the data is
    in 'dataBuffer'. Application can use this data.
    */
}

```

5.1.2.4.3.3 Code Examples**Example1**

//PIC32/PIC24 in auto sampling , Polling mode
#define MY_ADC_INSTANCE DRV_ADC_INDEX_0

```

DRV_HANDLE          adcHandle;           //Handle returned by DRV_ADC_Open function.
uint16_t dataBuffer;           //Buffer to which the data will be written.

/*
Open a new client. The handle returned by the this function should be used
a passing parameter for all the specific client related operations.
*/
adcHandle = DRV_ADC_Open(MY_ADC_INSTANCE, DRV_IO_INTENT_NONBLOCKING);

ipHandle = DRV_ADC_InputsRegister ( adcHandle ,inputSetConfig );

/*
ADC is not yet enabled. Enable it once the inputset is registered. Starts automatically
since auto sample mode is enabled.
*/
DRV_ADC_Start(adcHandle);

while (DRV_ADC_SampleAvailable(adcHandle))
{
    DRV_ADC_InputSetRead ( adcHandle, &dataBuffer, 1);
}

```

```

    /*
    Once the above function returns success, the data is
    in 'dataBuffer'. Application can use this data.
    */
}

```

Example 2:

//PIC32/PIC24 in Manual triggering , Polling mode

```
#define MY_ADC_INSTANCE DRV_ADC_INDEX_0
```

```

int main()
{
    SYS_MODULE_OBJ    myAdcObj;           //Object returned by DRV_ADC_Initialize function.
    DRV_ADC_INIT      adcInitData;        //Contains all the initialization values.
    DRV_HANDLE         adcHandle;          //Handle returned by DRV_ADC_Open function.
    DRV_ADC_INPUTSET_HANDLE ipHandle;      //Handle returned by DRV_ADC_InputSetRegister
function.
    DRV_ADC_INPUTSET_CONFIG inputSetConfig; //Contains all values to configure the inputset.
    uint16_t dataBuffer;                  //Buffer to which the data will be written.

    /*
    These initialization values should be based on the requirement of the
    application and the way it wants the hardware to be operating. Most of
    these values will be written directly to the hardware registers.
    */
    adcInitData.plibModuleId = ADC_ID_1;
    adcInitData.acquisitionTime = PLIB_ADC_ACQUISITION_TIME_15_TAD;
    adcInitData.voltageReference = PLIB_ADC_VREF_POS_TO_VDD_VREF_NEG_TO_VSS;
    adcInitData.clockFrequency = 4000000; //4MHz
    adcInitData.conversionClock = PLIB_ADC_CONV_CLOCK_5_TCY;
    adcInitData.conversionClockSource = PLIB_ADC_CLOCK_SRC_INTERNAL_RC;
    adcInitData.conversionTriggerSource = PLIB_ADC_CONVERSION_TRIGGER_INTERNAL_COUNT;
    adcInitData.dataOutputFormat = PLIB_ADC_OUTPUT_FORMAT_INTEGER_16BIT;

    adcInitData.interruptSource= PLIB_INT_SOURCE_ADC_1;
    adcInitData.samplesPerInterrupt = PLIB_ADC_SAMPLE_PER_INTERRUPT_AT_EACH_SAMPLE;

    myAdcObj = DRV_ADC_Initialize(MY_ADC_INSTANCE, (SYS_MODULE_INIT*)&adcInitData);

    /*
    Open a new client. The handle returned by the this function should be used
    a passing parameter for all the specific client related operations.
    */
    adcHandle = DRV_ADC_Open(MY_ADC_INSTANCE, DRV_IO_INTENT_NONBLOCKING);

    /*
    Driver invokes the registered callback function every successful data read.
    Application can perform an action or do a state change in the callback
    */
    inputSetConfig.callback = adcCallback;
    inputSetConfig.input = POTENTIOMETER_ANALOG_INPUT;
    inputSetConfig.errorTolerance = 10;
    inputSetConfig.samplingFrequency = 40000;

    ipHandle = DRV_ADC_InputSetRegister ( adcHandle ,inputSetConfig );

    /*
    ADC is not yet enabled. Enable it once the inputset is registered. Starts automatically
    since auto sample mode is enabled.
    */
    DRV_ADC_Start(adcHandle);

    while (1)
    {
        /*

```

```

    This function can be either called in a sequential way or it could
    be called from a timer periodically
    */
    triggerAdc();
DRV_HANDLE handle;

    DRV_ADC_Tasks (myAdcObj);

    DRV_ADC_InputSetRead ( adcHandle, &dataBuffer, 1);

    /*
    Once the above function returns success, the data is
    in 'dataBuffer'. Application can use this data.
    */
}

}

void adcCallback (void)
{
    // Application can do something here
}

void triggerAdc(void)
{
    DRV_ADC_OperationSetup(handle, DRV_ADC_START_SAMPLING);

    /* Give some delay between the two operations. */
    for (i=0; i<100; i++);
    DRV_ADC_OperationSetup(handle, DRV_ADC_START_CONVERSION);
}

```

Example 3:

```

//PIC32/PIC24 in auto sampling , Interrupt mode
#define MY_ADC_INSTANCE DRV_ADC_INDEX_0

int main()
{
    SYS_MODULE_OBJ    myAdcObj;           //Object returned by DRV_ADC_Initialize function.
    DRV_ADC_INIT      adcInitData;        //Contains all the initialization values.
    DRV_HANDLE        adcHandle;          //Handle returned by DRV_ADC_Open function.
    DRV_ADC_INPUTSET_HANDLE ipHandle;      //Handle returned by DRV_ADC_InputSetRegister
    function.
    DRV_ADC_INPUTSET_CONFIG inputSetConfig;//Contains all values to configure the inputset.
    uint16_t dataBuffer;                  //Buffer to which the data will be written.

    /*
    These initialization values should be based on the requirement of the
    application and the way it wants the hardware to be operating. Most of
    these values will be written directly to the hardware registers.
    */
    adcInitData.plibModuleId = ADC_ID_1;
    adcInitData.acquisitionTime = PLIB_ADC_ACQUISITION_TIME_15_TAD;
    adcInitData.voltageReference = PLIB_ADC_VREF_POS_TO_VDD_VREF_NEG_TO_VSS;
    adcInitData.clockFrequency = 4000000; //4MHz
    adcInitData.conversionClock = PLIB_ADC_CONV_CLOCK_5_TCY;
    adcInitData.conversionClockSource = PLIB_ADC_CLOCK_SRC_INTERNAL_RC;
    adcInitData.conversionTriggerSource = PLIB_ADC_CONVERSION_TRIGGER_INTERNAL_COUNT;
    adcInitData.dataOutputFormat = PLIB_ADC_OUTPUT_FORMAT_INTEGER_16BIT;
    adcInitData.initFlags = DRV_ADC_AUTO_SAMPLING;
    adcInitData.interruptSource= PLIB_INT_SOURCE_ADC_1;
    adcInitData.samplesPerInterrupt = PLIB_ADC_SAMPLE_PER_INTERRUPT_AT_EACH_SAMPLE;

```

```

myAdcObj = DRV_ADC_Initialize(MY_ADC_INSTANCE, (SYS_MODULE_INIT*)&adcInitData);

/*
Open a new client. The handle returned by the this function should be used
a passing parameter for all the specific client related operations.
*/
adcHandle = DRV_ADC_Open(MY_ADC_INSTANCE, DRV_IO_INTENT_NONBLOCKING);

/*
Driver invokes the registered callback function every successful data read.
Application can perform an action or do a state change in the callback
*/
inputSetConfig.callback = adcCallback;
inputSetConfig.input     = POTENTIOMETER_ANALOG_INPUT;
inputSetConfig.errorTolerance = 10;
inputSetConfig.samplingFrequency = 40000;

ipHandle = DRV_ADC_InputSetRegister ( adcHandle ,inputSetConfig );

/*
ADC is not yet enabled. Enable it once the inputset is registered. Starts automatically
since auto sample mode is enabled.
*/
DRV_ADC_Start(adcHandle);

while (1)
{
    /*
    Task function need not be called. Task function
    is registered as ISR in case of interrpt mode
    */

    DRV_ADC_InputSetRead ( adcHandle, &dataBuffer, 1);

    /*
    Once the above function returns success, the data is
    in 'dataBuffer'. Application can use this data.
    */
}

}

void adcCallback (void)
{
    // Application can do something here
}

// Do something else...
} while(total < MY_BUFFER_SIZE);

```

Example 4:

```

//PIC32/PIC24 in Manual triggering , Interrupt mode
#define MY_ADC_INSTANCE DRV_ADC_INDEX_0

int main()
{
    SYS_MODULE_OBJ      myAdcObj;           //Object returned by DRV_ADC_Initialize function.
    DRV_ADC_INIT         adcInitData;       //Contains all the initialization values.
    DRV_HANDLE           adcHandle;         //Handle returned by DRV_ADC_Open function.
    DRV_ADC_INPUTSET_HANDLE ipHandle;       //Handle returned by DRV_ADC_InputSetRegister
function.

```

```

DRV_ADC_INPUTSET_CONFIG    inputSetConfig; //Contains all values to configure the inputset.
uint16_t dataBuffer;      //Buffer to which the data will be written.

/*
These initialization values should be based on the requirement of the
application and the way it wants the hardware to be operating. Most of
these values will be written directly to the hardware registers.
*/
adcInitData.plibModuleId = ADC_ID_1;
adcInitData.acquisitionTime = PLIB_ADC_ACQUISITION_TIME_15_TAD;
adcInitData.voltageReference = PLIB_ADC_VREF_POS_TO_VDD_VREF_NEG_TO_VSS;
adcInitData.clockFrequency = 4000000; //4MHz
adcInitData.conversionClock = PLIB_ADC_CONV_CLOCK_5_TCY;
adcInitData.conversionClockSource = PLIB_ADC_CLOCK_SRC_INTERNAL_RC;
adcInitData.conversionTriggerSource = PLIB_ADC_CONVERSION_TRIGGER_INTERNAL_COUNT;
adcInitData.dataOutputFormat = PLIB_ADC_OUTPUT_FORMAT_INTEGER_16BIT;

adcInitData.interruptSource= PLIB_INT_SOURCE_ADC_1;
adcInitData.samplesPerInterrupt = PLIB_ADC_SAMPLE_PER_INTERRUPT_AT_EACH_SAMPLE;

myAdcObj = DRV_ADC_Initialize(MY_ADC_INSTANCE, (SYS_MODULE_INIT*)&adcInitData);

/*
Open a new client. The handle returned by the this function should be used
a passing parameter for all the specific client related operations.
*/
adcHandle = DRV_ADC_Open(MY_ADC_INSTANCE, DRV_IO_INTENT_NONBLOCKING);

/*
Driver invokes the registered callback function every successful data read.
Application can perform an action or do a state change in the callback
*/
inputSetConfig.callback = adcCallback;
inputSetConfig.input    = POTENTIOMETER_ANALOG_INPUT;
inputSetConfig.errorTolerance = 10;
inputSetConfig.samplingFrequency = 40000;

ipHandle = DRV_ADC_InputSetRegister ( adcHandle ,inputSetConfig );

/*
ADC is not yet enabled. Enable it once the inputset is registered. Starts automatically
since auto sample mode is enabled.
*/
DRV_ADC_Start(adcHandle);

while (1)
{
    /*
    This function can be either called in a sequential way or it could
    be called from a timer periodically
    */
    triggerAdc();

    /*
    Task function need not be called. Task function
    is registered as ISR in case of interrpt mode
    */

    DRV_ADC_InputSetRead ( adcHandle, &dataBuffer, 1);

    /*
    Once the above function returns success, the data is
    in 'dataBuffer'. Application can use this data.
    */
}

```

```
}

void adcCallback (void)
{
    // Application can do something here
}

void triggerAdc(void)
{
    DRV_ADC_OperationSetup(handle, DRV_ADC_START_SAMPLING);

    /* Give some delay between the two operations. */
    for (i=0; i<100; i++);
    DRV_ADC_OperationSetup(handle, DRV_ADC_START_CONVERSION);
}
```

5.1.2.5 Configuring the Library

The configuration of the ADC device driver is based on the file sys_config.h

This header file contains the configuration selection for the ADC device driver build. Based on the selections made here and the system setup the ADC device driver will support or not selected features. These configuration settings will apply to all instances of the device driver.

This header can be placed anywhere in the application specific folders and the path of this header needs to be presented to the include search for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.2.6 Building the Library

Files

Name	Description
drv_adc.c	ADC Driver source file.
drv_adc_client_multi.c	ADC Driver multiple client implementations.
drv_adc_client_single.c	ADC Driver single client implementations.
drv_adc_hw_dynamic.c	ADC Driver build variant implementation for the dynamic driver.
drv_adc_hw_static.c	ADC Driver build variant implementation for the static driver.

Description

5.1.2.6.1 drv_adc.c

ADC Driver Source File

This file implements the ADC Driver Interface routines. While building the driver from source, ALWAYS use this file in the build.

File Name

drv_adc.c

Company

Microchip Technology Inc.

5.1.2.6.2 drv_adc_client_multi.c

ADC Driver Multiple Client

This file implements the functions for the multiple client support. While building the driver from source, use this file in the build when `DRV_ADC_CLIENTS_NUMBER` is defined in the system configuration or when `DRV_ADC_INSTANCES_NUMBER` is defined in the system configuration.

File Name

drv_adc_client_multi.c

Company

Microchip Technology Inc.

5.1.2.6.3 drv_adc_client_single.c

ADC Driver Single Client

This file implements the functions for the single client support. While building the driver from source, use this file in the build when `DRV_ADC_CLIENTS_NUMBER` is not defined in system configuration.

File Name

drv_adc_client_single.c

Company

Microchip Technology Inc.

5.1.2.6.4 drv_adc_hw_dynamic.c

ADC Driver Build Variant implementation for dynamic driver

This file defines the build variant implementations for the dynamic driver. While building the driver from source, use this file in the build when `DRV_ADC_INSTANCES_NUMBER` is defined in the system configuration.

File Name

drv_adc_hw_dynamic.c

Company

Microchip Technology Inc.

5.1.2.6.5 drv_adc_hw_static.c

ADC Driver build variant implementation for static driver

This file defines the build variant implementations for the static driver. While building the driver from source, use this file in the build when `DRV_ADC_INSTANCES_NUMBER` is not defined in the system configuration.

File Name

drv_adc_hw_static.c

Company

Microchip Technology Inc.

5.1.2.7 Library Interface

Data Types and Constants

	Name	Description
	<u>DRV_ADC_CLIENT_STATUS</u>	Defines the client-specific status of the ADC driver.
	<u>DRV_ADC_INIT</u>	Defines the data required to initialize or reinitialize the ADC driver.
	<u>DRV_ADC_INIT_FLAGS</u>	Identifies the initialization flags of the ADC module.
	<u>DRV_ADC_ACQUISITION_TIME</u>	Defines the acquisition time.
	<u>DRV_ADC_ALTERNATE_INPUT_SAMPLING_ENABLE</u>	Enable the alternate input sampling feature of the ADC.
	<u>DRV_ADC_ANALOG_INPUT</u>	Defines the analog input channel.
	<u>DRV_ADC_AUTO_SAMPLING_ENABLE</u>	Enable the auto-sampling feature of the ADC.
	<u>DRV_ADC_CLIENTS_NUMBER</u>	Selects the maximum number of clients.
	<u>DRV_ADC_CONVERSION_CLOCK_PRESCALER</u>	Defines the conversion clock.
	<u>DRV_ADC_CONVERSION_CLOCK_SOURCE</u>	Defines the conversion clock source.
	<u>DRV_ADC_CONVERSION_TRIGGER_SOURCE</u>	Defines the conversion trigger source.
	<u>DRV_ADC_INDEX</u>	ADC static index selection.
	<u>DRV_ADC_INDEX_0</u>	ADC driver index definitions.
	<u>DRV_ADC_INDEX_1</u>	This is macro DRV_ADC_INDEX_1.
	<u>DRV_ADC_INDEX_2</u>	This is macro DRV_ADC_INDEX_2.
	<u>DRV_ADC_INDEX_COUNT</u>	Number of valid ADC driver indices.
	<u>DRV_ADC_INSTANCES_NUMBER</u>	Sets up the maximum number of hardware instances that can be supported.
	<u>DRV_ADC_INTERNAL_BUFFER_SIZE</u>	Define the internal buffer size.
	<u>DRV_ADC_INTERRUPT_MODE</u>	Controls operation of the driver in the interrupt or polled mode.
	<u>DRV_ADC_INTERRUPT_SOURCE</u>	Defines the interrupt source of the static driver.
	<u>DRV_ADC_PERIPHERAL_ID</u>	ADC PLIB ID Selection
	<u>DRV_ADC_POWER_STATE</u>	Controls the power state of the ADC.
	<u>DRV_ADC_RESULT_FORMAT</u>	Defines the data output format.
	<u>DRV_ADC_SAMPLES_PER_INTERRUPT</u>	Define the sample per interrupt.
	<u>DRV_ADC_STOP_ON_CONVERSION_ENABLE</u>	Enable the stop on conversion feature of the ADC.
	<u>DRV_ADC_VOLTAGE_REFERENCE</u>	Defines the voltage reference.

Client Core Configuration Functions

	Name	Description
	<u>DRV_ADC_ClientStatus</u>	Gets the current client-specific status the ADC driver.
	<u>DRV_ADC_Close</u>	Closes an opened-instance of the ADC driver.
	<u>DRV_ADC_InputsRegister</u>	Registers an input set with the driver for sampling.
	<u>DRV_ADC_Open</u>	Opens the specified ADC driver instance and returns a handle to it.
	<u>DRV_ADC_SamplesAvailable</u>	Identifies if any the ADC driver has any samples available to read.
	<u>DRV_ADC_Start</u>	Starts the ADC driver sampling and converting analog to digital values.
	<u>DRV_ADC_Stop</u>	Stops the ADC driver from sampling and converting analog to digital values.

Other Functions

	Name	Description
≡	DRV_ADC_SampleMaxGet	Gets the max sample value.
≡	DRV_ADC_SampleMinGet	Gets the min sample value.
≡	DRV_ADC_SamplesRead	Reads the converted sample data from the ADC driver.
≡	DRV_ADC_SamplesReadLatest	Reads the most recently converted sample data from the ADC driver.

System Interaction Functions

	Name	Description
≡	DRV_ADC_Deinitialize	Deinitializes the specified instance of the ADC driver module.
≡	DRV_ADC_Initialize	Initializes the ADC driver.
≡	DRV_ADC_Reinitialize	Reinitializes the ADC instance for the specified module ID.
≡	DRV_ADC_Status	Provides the current status of the ADC driver module.
≡	DRV_ADC_Tasks	Maintains the driver's state machine and implements its ISR.

Description**5.1.2.7.1 System Interaction Functions****5.1.2.7.1.1 DRV_ADC_Deinitialize Function****C**

```
void DRV_ADC_Deinitialize(
    SYS_MODULE_OBJ object
);
```

Description

This function deinitializes the specified instance of the ADC driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Preconditions

The [DRV_ADC_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_ADC_Initialize

Returns

None.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the [DRV_ADC_Status](#) operation. The system has to use [DRV_ADC_Status](#) to find out when the module is in the ready state.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_ADC_Initialize
SYS_STATUS        status;

DRV_ADC_Deinitialize(object);

status = DRV_ADC_Status(object);
if (SYS_MODULE_DEINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

5.1.2.7.1.2 DRV_ADC_Initialize Function**C**

```
SYS_MODULE_OBJ DRV_ADC_Initialize(
    const SYS_MODULE_INDEX drvIndex,
    const SYS_MODULE_INIT * const init
);
```

Description

This function initializes the ADC driver, making it ready for clients to open and use it.

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver. This pointer may be null if no data is required because static overrides have been provided.

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#).

Remarks

This function must be called before any other ADC function is called.

This function should only be called once during system initialization unless [DRV_ADC_Deinitialize](#) is called to deinitialize the driver instance.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to re-initialize, it will be reported by the [DRV_ADC_Status](#) operation. The system must use [DRV_ADC_Status](#) to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Example

```
DRV_ADC_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the init structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.adcID             = ADC_ID_1;
```

```

init.initFlags          = DRV_ADC_AUTO_SAMPLING;
init.clockFrequency     = 4000000; // 4MHz
init.acquisitionTime    = ADC_ACQUISITION_TIME_15_TAD;
init.voltageReference   = ADC_VREF_POS_TO_VDD_VREF_NEG_TO_VSS;
init.conversionClockPrescaler = ADC_CONV_CLOCK_5_TCY;
init.conversionClockSource = ADC_CLOCK_SRC_INTERNAL_RC;
init.conversionTriggerSource = ADC_CONVERSION_TRIGGER_INTERNAL_COUNT;
init.samplesPerInterrupt = ADC_SAMPLE_PER_INTERRUPT_AT_EACH_SAMPLE;
init.resultFormat       = ADC_RESULT_FORMAT_INTEGER_16BIT;
init.analogInput         = ADC_INPUT_AN2;
init.interruptSource     = INT_SOURCE_ADC_1;

// Initialize the ADC driver
objectHandle = DRV_ADC_Initialize(DRV_ADC_INDEX_0, (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}

```

5.1.2.7.1.3 DRV_ADC_Reinitialize Function

C

```

void DRV_ADC_Reinitialize(
    SYS_MODULE_OBJ object,
    const SYS_MODULE_INIT * const init
);

```

Description

This function reinitializes and refreshes the hardware for the index instance of the ADC module using the hardware initialization given data. It does not clear or reinitialize internal data structures (although it may change the value of a few appropriate data items necessary to manage the new hardware state).

Preconditions

The [DRV_ADC_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_ADC_Initialize

Returns

None.

Example

```

DRV_ADC_INIT    init;
SYS_MODULE_OBJ  objectHandle; // Returned from DRV_ADC_Initialize
SYS_STATUS      adcStatus;

// Populate the init structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.adcID            = ADC_ID_1;
init.initFlags        = DRV_ADC_AUTO_SAMPLING;
init.clockFrequency   = 4000000; // 4MHz
init.acquisitionTime  = ADC_ACQUISITION_TIME_15_TAD;
init.voltageReference = ADC_VREF_POS_TO_VDD_VREF_NEG_TO_VSS;
init.conversionClockPrescaler = ADC_CONV_CLOCK_5_TCY;
init.conversionClockSource = ADC_CLOCK_SRC_INTERNAL_RC;
init.conversionTriggerSource = ADC_CONVERSION_TRIGGER_INTERNAL_COUNT;
init.samplesPerInterrupt = ADC_SAMPLE_PER_INTERRUPT_AT_EACH_SAMPLE;
init.resultFormat     = ADC_RESULT_FORMAT_INTEGER_16BIT;
init.analogInput       = ADC_INPUT_AN2;
init.interruptSource   = INT_SOURCE_ADC_1;

```

```

// Reinitialize the ADC driver
DRV_ADC_Reinitialize(objectHandle, (SYS_MODULE_INIT*)&init);

// Check the status of the driver
adcStatus = DRV_ADC_Status(myAdcObj);
if (SYS_STATUS_BUSY == adcStatus)
{
    // do something else and check back later
}
else if (SYS_STATUS_ERROR >= adcStatus)
{
    // Handle error
}

```

Remarks: This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

This function will NEVER block **for** hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the DRV_ADC_Status operation. The system must use DRV_ADC_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed **using this** function.

5.1.2.7.1.4 DRV_ADC_Status Function

C

```

SYS_STATUS DRV_ADC_Status(
    SYS_MODULE_OBJ object
);

```

Description

This function provides the current status of the ADC driver module.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_ADC_Initialize

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to [accept](#) new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR

The this operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns `SYS_STATUS_BUSY`, the a previous operation has not yet completed. Once the status operation returns `SYS_STATUS_READY`, any previous operations have completed.

The value of `SYS_STATUS_ERROR` is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_TMR_Initialize
SYS_STATUS        status;

status = DRV_ADC_Status(object);
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

5.1.2.7.1.5 DRV_ADC_Tasks Function

C

```
void DRV_ADC_Tasks(
    SYS_MODULE_OBJ object
);
```

Description

This function is used to maintain the driver's internal state machine and implement its ISR for interrupt-driven implementations.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called for the specified ADC driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_ADC_Initialize)

Returns

None

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function ([SYS_Tasks](#)) or by the appropriate raw ISR.

This function may excute in an ISR context and will never block or access any resources that may cause it to block.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_ADC_Initialize

while (true)
{
    DRV_ADC_Tasks (object);

    // Do other tasks
}
```

5.1.2.7.2 Client Core Configuration Functions

5.1.2.7.2.1 DRV_ADC_ClientStatus Function

C

```
DRV_ADC_CLIENT_STATUS DRV_ADC_ClientStatus(  
    DRV_HANDLE handle  
)  
;
```

Description

This function gets the client-specific status of the ADC driver associated with the given handle.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Returns

A [DRV_ADC_CLIENT_STATUS](#) value describing the current status of the driver.

Remarks

This function will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE      handle; // Returned from DRV_ADC_Open  
DRV_ADC_CLIENT_STATUS status;  
  
status = DRV_ADC_ClientStatus(handle);  
if(DRV_ADC_CLIENT_STATUS_ERROR >= status)  
{  
    // Handle the error  
}
```

5.1.2.7.2.2 DRV_ADC_Close Function

C

```
void DRV_ADC_Close(  
    DRV_HANDLE handle  
)  
;
```

Description

This function closes an opened-instance of the ADC driver, invalidating the handle.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called for the specified ADC driver instance.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Returns

None

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver functions. A new handle must be obtained by calling [DRV_ADC_Open](#) before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the [DRV_ADC_Status](#) operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Example

```
DRV_HANDLE handle; // Returned from DRV_ADC_Open

DRV_ADC_Close(handle);
```

5.1.2.7.2.3 DRV_ADC_InputsRegister Function**C**

```
void DRV_ADC_InputsRegister(
    DRV_HANDLE handle,
    uint32_t inputsMask
);
```

Description

This function registers an input set with the driver for sampling.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called for the specified ADC device instance and the [DRV_ADC_Status](#) must have returned SYS_STATUS_READY.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
inputsMask	Mask bits recognising the various Analog Channels

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_ADC_Open

DRV_ADC_InputsRegister (handle, ADC_INPUT_AN2|ADC_INPUT_AN3);
```

5.1.2.7.2.4 DRV_ADC_Open Function**C**

```
DRV_HANDLE DRV_ADC_Open(
    const SYS_MODULE_INDEX drvIndex,
    const DRV_IO_INTENT intent
);
```

Description

This function opens the specified ADC driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is [DRV_HANDLE_INVALID](#).

Remarks

The handle returned is valid until the [DRV_ADC_Close](#) function is called.

This function will NEVER block waiting for hardware.

If the [DRV_IO_INTENT_BLOCKING](#) is requested and the driver was built appropriately to support blocking behavior, other client-level operations may block waiting on hardware until they are complete.

If [DRV_IO_INTENT_NON_BLOCKING](#) is requested the driver client can call the [DRV_ADC_ClientStatus](#) operation to find out when the module is in the ready state.

If the requested intent flags are not supported, the function will return [DRV_HANDLE_INVALID](#).

Example

```
DRV_HANDLE  handle;

handle = DRV_ADC_Open(DRV_ADC_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

5.1.2.7.2.5 DRV_ADC_SamplesAvailable Function**C**

```
bool DRV_ADC_SamplesAvailable(
    DRV_HANDLE handle
);
```

Description

This function identifies if any the ADC driver has any samples available to read.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

The desired analog input set must have been selected by calling [DRV_ADC_InputsRegister](#).

[DRV_ADC_Start](#) must have been called to start the driver sampling and converting analog input samples to digital values.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Returns

- true - If one or more samples are available for the registered input set
- false - If no samples are available

Remarks

None.

Example

```
DRV_HANDLE    handle; // Returned from DRV_ADC_Open
DRV_ADC_SAMPLE buffer;

// An input set must have been registered and the ADC started.

if (DRV_ADC_SamplesAvailable(handle))
{
    DRV_ADC_SamplesRead(handle, &buffer, sizeof(buffer));
}
```

5.1.2.7.2.6 DRV_ADC_Start Function**C**

```
void DRV_ADC_Start(
    DRV_HANDLE handle
);
```

Description

This function starts the ADC driver sampling the selected analog inputs and converting the samples to digital values.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

The desired analog input set must have been selected by calling [DRV_ADC_InputsRegister](#).

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Returns

None.

Remarks

Call [DRV_ADC_SamplesAvailable](#) to find out when one or more samples is available.

Example

```
DRV_HANDLE handle; // Returned from DRV_ADC_Open

// Use DRV_ADC_InputsRegister to register the desired inputs.

DRV_ADC_Start(handle);
```

5.1.2.7.2.7 DRV_ADC_Stop Function

C

```
void DRV_ADC_Stop(  
    DRV_HANDLE handle  
);
```

Description

This function stops the ADC driver from sampling analog inputs and converting the samples to digital values.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Returns

None.

Remarks

Call [DRV_ADC_Start](#) to restart sampling and conversion of analog inputs to digital values.

Example

```
DRV_HANDLE handle; // Returned from DRV_ADC_Open  
  
DRV_ADC_Stop(handle);
```

5.1.2.7.3 Other Functions

5.1.2.7.3.1 DRV_ADC_SampleMaxGet Function

C

```
ADC_SAMPLE DRV_ADC_SampleMaxGet(  
    DRV_HANDLE handle  
);
```

Description

This function returns the max sample value.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Returns

Max sample value.

Remarks

None

Example

```
DRV_HANDLE      handle; // Returned from DRV_ADC_Open
ADC_SAMPLE      maxValue, minValue;

maxValue = DRV_ADC_SampleMaxGet(handle);
minValue = DRV_ADC_SampleMinGet(handle);
adcRange = maxValue - minValue;
```

5.1.2.7.3.2 DRV_ADC_SampleMinGet Function**C**

```
ADC_SAMPLE DRV_ADC_SampleMinGet(
    DRV_HANDLE handle
);
```

Description

This function returns the min sample value.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Returns

Min sample value.

Remarks

None

Example

```
DRV_HANDLE      handle; // Returned from DRV_ADC_Open
ADC_SAMPLE      maxValue, minValue;

maxValue = DRV_ADC_SampleMaxGet(handle);
minValue = DRV_ADC_SampleMinGet(handle);
adcRange = maxValue - minValue;
```

5.1.2.7.3.3 DRV_ADC_SamplesRead Function**C**

```
unsigned short DRV_ADC_SamplesRead(
    DRV_HANDLE handle,
    ADC_SAMPLE * buffer,
    unsigned short bufferSize
);
```

Description

This function reads converted sample data from the ADC driver into the given buffer. How many samples depends on how many samples are available and on the relative sizes of the samples and the buffer passed in.

Zero (0) samples are copied if the bufferSize is less than the size of a complete set of samples for the registered inputs.

N sets of samples where the bufferSize / size of a complete set of samples = N, unless less than N samples are currently available. Then, the number of samples currently available are copied.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

The desired analog input set must have been selected by calling [DRV_ADC_InputsRegister](#).

[DRV_ADC_Start](#) must have been called to start the driver sampling and converting analog input samples to digital values.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
buffer	A pointer to the buffer to where the sample data will be copied
bufferSize	Size of the buffer (in bytes)

Returns

Number of bytes of sample data copied to the specified buffer.

Remarks

The [DRV_ADC_SamplesAvailable](#) function can be used to determine if any sample data is available.

Calling this function removes the samples from the driver's internal buffer queue of samples.

Example

```
DRV_HANDLE    handle; // Returned from DRV_ADC_Open
ADC_SAMPLE    buffer;

// An input set must have been registered and the ADC started.

if (DRV_ADC_SamplesAvailable(handle))
{
    DRV_ADC_SamplesRead(handle, &buffer, sizeof(buffer));
}
```

5.1.2.7.3.4 DRV_ADC_SamplesReadLatest Function

C

```
unsigned short DRV_ADC_SamplesReadLatest(
    DRV_HANDLE handle,
    ADC_SAMPLE * buffer,
    unsigned short bufferSize
);
```

Description

This function reads only the most recently converted sample data from the ADC driver into the given buffer. Only the data for a single set of samples for the registered inputs is copied to the caller's buffer. If the buffer size is less than the size of a complete set of samples for the registered inputs, no data is copied to the caller's buffer. Also, no sample data is copied to the caller's buffer if no sample data is currently available.

Preconditions

The [DRV_ADC_Initialize](#) function must have been called.

[DRV_ADC_Open](#) must have been called to obtain a valid opened device handle.

The desired analog input set must have been selected by calling [DRV_ADC_InputsRegister](#).

[DRV_ADC_Start](#) must have been called to start the driver sampling and converting analog input samples to digital values.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
buffer	A pointer to the buffer to where the sample data will be copied
bufferSize	Size of the buffer (in bytes)

Returns

Number of bytes of sample data copied to the specified buffer.

Remarks

The [DRV_ADC_SamplesAvailable](#) function can be used to determine if any sample data is available.

This function does not remove any data from the driver's internal buffer queue of sample data.

Example

```
DRV_HANDLE      handle; // Returned from DRV_ADC_Open
ADC_SAMPLE      buffer;

// An input set must have been registered and the ADC started.

if (DRV_ADC_SamplesAvailable(handle))
{
    DRV_ADC_SamplesReadLatest(handle, &buffer, sizeof(buffer));
}
```

5.1.2.7.4 Data Types and Constants

5.1.2.7.4.1 DRV_ADC_CLIENT_STATUS Enumeration

C

```
typedef enum {
    DRV_ADC_CLIENT_STATUS_STARTED,
    DRV_ADC_CLIENT_STATUS_STOPPED,
    DRV_ADC_CLIENT_STATUS_READY,
    DRV_ADC_CLIENT_STATUS_BUSY,
    DRV_ADC_CLIENT_STATUS_INVALID,
    DRV_ADC_CLIENT_STATUS_OVERFLOW,
    DRV_ADC_CLIENT_STATUS_BUFFER_TOO_SMALL
} DRV_ADC_CLIENT_STATUS;
```

Description

ADC Client Status

This enumeration defines the client-specific status codes of the ADC driver.

Members

Members	Description
DRV_ADC_CLIENT_STATUS_STARTED	ADC Started
DRV_ADC_CLIENT_STATUS_STOPPED	stopped on error
DRV_ADC_CLIENT_STATUS_READY	Driver OK, ready for client operations
DRV_ADC_CLIENT_STATUS_BUSY	An operation is currently in progress
DRV_ADC_CLIENT_STATUS_INVALID	Client in an invalid (or un-opened) state
DRV_ADC_CLIENT_STATUS_OVERFLOW	Driver Overflowed

DRV_ADC_CLIENT_STATUS_BUFFER_TOO_SMALL	Input Set Read Buffer Too Small

Remarks

Returned by the [DRV_ADC_ClientStatus](#) function.

5.1.2.7.4.2 DRV_ADC_INIT Structure**C**

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    ADC_MODULE_ID adcId;
    DRV_ADC_INIT_FLAGS initFlags;
    uint32_t clockFrequency;
    ADC_ACQUISITION_TIME acquisitionTime;
    ADC_VOLTAGE_REFERENCE voltageReference;
    ADC_CONVERSION_CLOCK conversionClockPrescaler;
    ADC_CLOCK_SOURCE conversionClockSource;
    ADC_CONVERSION_TRIGGER_SOURCE conversionTriggerSource;
    ADC_SAMPLES_PER_INTERRUPT samplesPerInterrupt;
    ADC_RESULT_FORMAT resultFormat;
    ADC_INPUTS_POSITIVE analogInput;
    INT_SOURCE interruptSource;
} DRV_ADC_INIT;
```

Description

ADC Driver Initialization Data

This structure defines the data required to initialize or reinitialize the ADC driver.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
ADC_MODULE_ID adcId;	Identifies timer hardware module (PLIB-level) ID
DRV_ADC_INIT_FLAGS initFlags;	Initialization Flags
uint32_t clockFrequency;	Clock Frequency
ADC_ACQUISITION_TIME acquisitionTime;	Acquisition Time
ADC_VOLTAGE_REFERENCE voltageReference;	Voltage Reference Selection
ADC_CONVERSION_CLOCK conversionClockPrescaler;	Clock Setup for the Conversion
ADC_CLOCK_SOURCE conversionClockSource;	Clock Source for Conversion
ADC_CONVERSION_TRIGGER_SOURCE conversionTriggerSource;	Conversion Trigger Source
ADC_SAMPLES_PER_INTERRUPT samplesPerInterrupt;	Samples Per Interrupt valid values = 0- 15
ADC_RESULT_FORMAT resultFormat;	Result Format
ADC_INPUTS_POSITIVE analogInput;	Input Channel to convert
INT_SOURCE interruptSource;	Interrupt Source for the module

Remarks

Not all init features are available for all devices. Refer to the specific data sheet to determine availability.

5.1.2.7.4.3 DRV_ADC_INIT_FLAGS Enumeration

C

```
typedef enum {
    DRV_ADC_STOP_CONVERSION_ON_INTERRUPT,
    DRV_ADC_ALTERNATE_INPUT_SAMPLING,
    DRV_ADC_AUTO_SAMPLING,
    DRV_ADC_MANUAL_SAMPLING
} DRV_ADC_INIT_FLAGS;
```

Description

ADC Initialization Flags

This data type identifies the initialization flags of the ADC module.

Members

Members	Description
DRV_ADC_STOP_CONVERSION_ON_INTERRUPT	Stops the conversion on the interrupt
DRV_ADC_ALTERNATE_INPUT_SAMPLING	Alternate Input Sampling
DRV_ADC_AUTO_SAMPLING	Begin sampling automatically after previous conversion
DRV_ADC_MANUAL_SAMPLING	Manual Sampling

Remarks

Not all modes are available on all devices. Refer to the specific data sheet to determine availability.

5.1.2.7.4.4 DRV_ADC_ACQUISITION_TIME Macro

C

```
#define DRV_ADC_ACQUISITION_TIME ADC_ACQUISITION_TIME_4_TAD
```

Description

ADC Acquisition Time

This macro defines the acquisition time of the ADC driver. This provides static override of the dynamic selection of the acquisition time. If this macro is defined, this will be used for setting up the acquisition time and not the acquisition time value provided by [DRV_ADC_INIT](#).

Remarks

None.

5.1.2.7.4.5 DRV_ADC_ALTERNATE_INPUT_SAMPLING_ENABLE Macro

C

```
#define DRV_ADC_ALTERNATE_INPUT_SAMPLING_ENABLE false
```

Description

ADC Alternate Input Sampling Enable

This macro enables the alternate input sampling feature of the ADC. This macro can take the following values:

- true - Enables the alternate Input sampling feature of the ADC
- false - Disables the alternate Input sampling feature of the ADC
- [DRV_CONFIG_NOT_SUPPORTED](#) - When the feature is not supported on the instance

Remarks

None.

5.1.2.7.4.6 DRV_ADC_ANALOG_INPUT Macro**C**

```
#define DRV_ADC_ANALOG_INPUT ADC_INPUT_AN2
```

Description

ADC Analog input channel

This macro defines the analog input channel for the ADC driver. This provides static override of the dynamic selection of the analog input. If this macro is defined, this will be used for setting up the analog input and not the analog input value provided by [DRV_ADC_INIT](#).

Remarks

None.

5.1.2.7.4.7 DRV_ADC_AUTO_SAMPLING_ENABLE Macro**C**

```
#define DRV_ADC_AUTO_SAMPLING_ENABLE true
```

Description

ADC Auto Sampling Enable

This macro enables the auto-sampling feature of the ADC. This macro can take the following values:

- true - Enables the auto-sampling feature of the ADC
- false - Disables the auto-sampling feature of the ADC
- [DRV_CONFIG_NOT_SUPPORTED](#) - When the feature is not supported on the instance

Remarks

None.

5.1.2.7.4.8 DRV_ADC_CLIENTS_NUMBER Macro**C**

```
#define DRV_ADC_CLIENTS_NUMBER 1
```

Description

ADC Maximum Number of Clients

This definition selectd the maximum number of clients that the ADC driver can support at run time.

Remarks

None.

5.1.2.7.4.9 DRV_ADC_CONVERSION_CLOCK_PRESCALER Macro**C**

```
#define DRV_ADC_CONVERSION_CLOCK_PRESCALER ADC_CONV_CLOCK_4_TCY
```

Description

ADC Conversion Clock

This macro defines the conversion clock for the ADC driver. This provides static override of the dynamic selection of the conversion clock. If this macro is defined, this will be used for setting up the conversion clock and not the conversion clock value provided by [DRV_ADC_INIT](#).

Remarks

None.

5.1.2.7.4.10 DRV_ADC_CONVERSION_CLOCK_SOURCE Macro**C**

```
#define DRV_ADC_CONVERSION_CLOCK_SOURCE ADC_CLOCK_SRC_SYSTEM_CLOCK
```

Description

ADC Conversion Clock Source

This macro defines the conversion clock source for the ADC driver. This provides static override of the dynamic selection of the conversion clock source. If this macro is defined, this will be used for setting up the conversion clock source and not the conversion clock source value provided by [DRV_ADC_INIT](#).

Remarks

None.

5.1.2.7.4.11 DRV_ADC_CONVERSION_TRIGGER_SOURCE Macro**C**

```
#define DRV_ADC_CONVERSION_TRIGGER_SOURCE ADC_CONVERSION_TRIGGER_INTERNAL_COUNT
```

Description

Conversion Trigger Source

This macro defines the conversion trigger source for the ADC driver. This provides static override of the dynamic selection of the conversion trigger source. If this macro is defined, this will be used for setting up the conversion trigger source and not the conversion trigger source value provided by [DRV_ADC_INIT](#).

Remarks

None.

5.1.2.7.4.12 DRV_ADC_INDEX Macro**C**

```
#define DRV_ADC_INDEX DRV_ADC_INDEX_0
```

Description

ADC Static Index Selection

ADC static index selection for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

5.1.2.7.4.13 DRV_ADC_INDEX_0 Macro

C

```
#define DRV_ADC_INDEX_0 0
```

Description

Driver ADC Module Index Numbers

These constants provide ADC driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the [DRV_ADC_Initialize](#) function to identify the driver instance in use.

5.1.2.7.4.14 DRV_ADC_INDEX_1 Macro

C

```
#define DRV_ADC_INDEX_1 1
```

Description

This is macro DRV_ADC_INDEX_1.

5.1.2.7.4.15 DRV_ADC_INDEX_2 Macro

C

```
#define DRV_ADC_INDEX_2 2
```

Description

This is macro DRV_ADC_INDEX_2.

5.1.2.7.4.16 DRV_ADC_INDEX_COUNT Macro

C

```
#define DRV_ADC_INDEX_COUNT _ADC_EXISTS
```

Description

Driver ADC Driver Module Index Count

This constant identifies ADC driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is device-specific.

5.1.2.7.4.17 DRV_ADC_INSTANCES_NUMBER Macro

C

```
#define DRV_ADC_INSTANCES_NUMBER 1
```

Description

ADC hardware instance configuration

This macro sets up the maximum number of hardware instances that can be supported.

Remarks

None.

5.1.2.7.4.18 DRV_ADC_INTERNAL_BUFFER_SIZE Macro

C

```
#define DRV_ADC_INTERNAL_BUFFER_SIZE 2
```

Description

ADC Internal buffer size

This macro defines the internal buffer size.

Remarks

None.

5.1.2.7.4.19 DRV_ADC_INTERRUPT_MODE Macro

C

```
#define DRV_ADC_INTERRUPT_MODE true
```

Description

ADC Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of adc operation is desired
- false - Select if polling mode of adc operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

5.1.2.7.4.20 DRV_ADC_INTERRUPT_SOURCE Macro

C

```
#define DRV_ADC_INTERRUPT_SOURCE PLIB_INT_SOURCE_ADC_1
```

Description

ADC Interrupt Source

Macro to define the interrupt source of the static driver.

Remarks

Refer to the Interrupt Peripheral Library document for more information on the PLIB_INT_SOURCE enumeration.

5.1.2.7.4.21 DRV_ADC_PERIPHERAL_ID Macro

C

```
#define DRV_ADC_PERIPHERAL_ID ADC_ID_1
```

Description

ADC PLIB ID Selection

This macro selects the ADC PLIB ID Selection. This is an initialization override of the `adcID` member of the initialization configuration.

Remarks

None.

5.1.2.7.4.22 DRV_ADC_POWER_STATE Macro

C

```
#define DRV_ADC_POWER_STATE SYS_MODULE_POWER_IDLE_STOP
```

Description

ADC power state configuration

This macro controls the power state of the ADC.

Remarks

This feature may not be available in the device or the ADC module selected.

5.1.2.7.4.23 DRV_ADC_RESULT_FORMAT Macro

C

```
#define DRV_ADC_RESULT_FORMAT ADC_RESULT_FORMAT_INTEGER_16BIT
```

Description

ADC Data Output Format

This macro defines the data output format for the ADC driver. This provides static override of the dynamic selection of the data output format. If this macro is defined, this will be used for setting up the data output format and not the data output format value provided by [DRV_ADC_INIT](#).

Remarks

None.

5.1.2.7.4.24 DRV_ADC_SAMPLES_PER_INTERRUPT Macro

C

```
#define DRV_ADC_SAMPLES_PER_INTERRUPT 2
```

Description

Samples per Interrupt

This macro defines the samples per interrupt of the ADC driver. This provides static override of the dynamic selection of the sample per interrupt. If this macro is defined, this will be used for setting up the samples per interrupt and not the samples per interrupt value provided by [DRV_ADC_INIT](#).

Remarks

Select this size based on the device available and the number of samples that are required to form a set.

5.1.2.7.4.25 DRV_ADC_STOP_ON_CONVERSION_ENABLE Macro

C

```
#define DRV_ADC_STOP_ON_CONVERSION_ENABLE false
```


Description

ADC Stop on conversion Enable

This macro enables the stop on conversion feature of the ADC. This macro can take the following values:

- true - Enables the ADC to stop on conversion
- false - Disables the ADC to stop on conversion
- [DRV_CONFIG_NOT_SUPPORTED](#) - When the feature is not supported on the instance

Remarks

None.

5.1.2.7.4.26 DRV_ADC_VOLTAGE_REFERENCE Macro**C**

```
#define DRV_ADC_VOLTAGE_REFERENCE ADC_VREF_POS_TO_VDD_VREF_NEG_TO_VSS
```

Description

ADC Voltage Reference

This macro defines the voltage reference of the ADC driver. This provides static override of the dynamic selection of the voltage reference. If this macro is defined, this will be used for setting up the voltage reference and not the voltage reference value provided by [DRV_ADC_INIT](#).

Remarks

None.

5.1.2.8 Files**Files**

Name	Description
drv_adc.h	ADC Driver interface definition.

Description**5.1.2.8.1 drv_adc.h**

ADC Driver Interface Definition

The ADC device driver provides a simple interface to manage the ADC modules on Microchip microcontrollers. This file defines the interface definition for the ADC driver.

Enumerations

	Name	Description
	DRV_ADC_CLIENT_STATUS	Defines the client-specific status of the ADC driver.
	DRV_ADC_INIT_FLAGS	Identifies the initialization flags of the ADC module.

Functions

	Name	Description
≡	DRV_ADC_ClientStatus	Gets the current client-specific status the ADC driver.
≡	DRV_ADC_Close	Closes an opened-instance of the ADC driver.
≡	DRV_ADC_Deinitialize	Deinitializes the specified instance of the ADC driver module.
≡	DRV_ADC_Initialize	Initializes the ADC driver.
≡	DRV_ADC_InputsRegister	Registers an input set with the driver for sampling.
≡	DRV_ADC_Open	Opens the specified ADC driver instance and returns a handle to it.
≡	DRV_ADC_Reinitialize	Reinitializes the ADC instance for the specified module ID.
≡	DRV_ADC_SampleMaxGet	Gets the max sample value.
≡	DRV_ADC_SampleMinGet	Gets the min sample value.
≡	DRV_ADC_SamplesAvailable	Identifies if any the ADC driver has any samples available to read.
≡	DRV_ADC_SamplesRead	Reads the converted sample data from the ADC driver.
≡	DRV_ADC_SamplesReadLatest	Reads the most recently converted sample data from the ADC driver.
≡	DRV_ADC_Start	Starts the ADC driver sampling and converting analog to digital values.
≡	DRV_ADC_Status	Provides the current status of the ADC driver module.
≡	DRV_ADC_Stop	Stops the ADC driver from sampling and converting analog to digital values.
≡	DRV_ADC_Tasks	Maintains the driver's state machine and implements its ISR.

Macros

	Name	Description
≡	DRV_ADC_INDEX_0	ADC driver index definitions.
≡	DRV_ADC_INDEX_1	This is macro DRV_ADC_INDEX_1.
≡	DRV_ADC_INDEX_2	This is macro DRV_ADC_INDEX_2.
≡	DRV_ADC_INDEX_COUNT	Number of valid ADC driver indices.

Structures

	Name	Description
≡	DRV_ADC_INIT	Defines the data required to initialize or reinitialize the ADC driver.

File Name

drv_adc.h

Company

Microchip Technology Inc.

5.1.3 Ethernet MAC Driver Library

5.1.3.1 Introduction

Ethernet (Media Access) Controller Driver Library

for

Microchip Microcontrollers

This library provides a driver-level abstraction of the on-chip Ethernet Controller found on many PIC32 devices.

The "Host-To-Network" layer of a TCP/IP stack organization covers the Data Link and Physical Layers of the standard OSI stack. The Ethernet Controller provides the Data Link or Media Access Control Layer, in addition to other functions discussed below. An external Ethernet "PHY" provides the Physical layer, providing conversion between the digital and analog.

Description

The PIC32 Ethernet Controller is a bus master module that interfaces with an off-chip PHY in order to implement a complete Ethernet node in a system. The following are some of the key features of this module:

- Supports 10/100 Ethernet
 - Full-Duplex and Half-Duplex operation
 - Broadcast, Multicast and Unicast packets
 - Manual and automatic flow control
 - Supports Auto-MDIX enabled PHYs
 - Reduced Media Independent Interface (RMII) and Media Independent Interface (MII) PHY data interfaces
 - Performance statistics metrics in hardware.
- RAM descriptor based DMA operation for both receive and transmit path
- Fully configurable interrupts
- Configurable receive packet filtering using:
 - 64-bit Hash Table
 - 64-byte Pattern Match
 - Magic Packet™ Filtering
 - Runt Packet Detection and Filtering
- Supports Packet Payload Checksum calculation
- CRC Check

Support for the Serial Management Interface (SMI) (also known as the MIIM interface) is provided by the Ethernet PHY Driver Library.

5.1.3.2 Release Notes

MPLAB Harmony Version: v0.70b **Ethernet MAC Library Version :** 7.10 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.1.3.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its

licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.3.4 Using the Library

This topic describes the basic architecture of the Ethernet MAC Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_ethmac.h](#)

The interface to the Ethernet MAC library is defined in the "[drv_ethmac.h](#)" header file. This file is included by the "[drv_ethmac.h](#)" file. Any C language source (.c) file that uses the Ethernet MAC Driver Library should include "drv.h".

Library File:

The Ethernet MAC Driver Library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the library interacts with the framework.

5.1.3.4.1 Abstraction Model

The Ethernet Controller provides the modules needed to implement a 10/100 Mbps Ethernet node using an external Ethernet PHY chip. The PHY chip provides a digital-analog interface as part of the Physical Layer and the controller provides the Media Access Controller (MAC) layer above the PHY.

As shown in Figure 1, the Ethernet Controller consists of the following modules:

- Media Access Control (MAC) block: Responsible for implementing the MAC functions of the Ethernet IEEE 802.3 Specification
- Flow Control (FC) block: Responsible for control of the transmission of PAUSE frames. (Reception of PAUSE frames is handled within the MAC.)
- RX Filter (RXF) block: This module performs filtering on every receive packet to determine whether each packet should be accepted or rejected
- TX DMA/TX Buffer Management Engine: The TX DMA and TX Buffer Management engines perform data transfers from the memory (using descriptor tables) to the MAC Transmit Interface
- RX DMA/RX Buffer Management Engine: The RX DMA and RX Buffer Management engines transfer receive packets from the MAC to the memory (using descriptor tables)

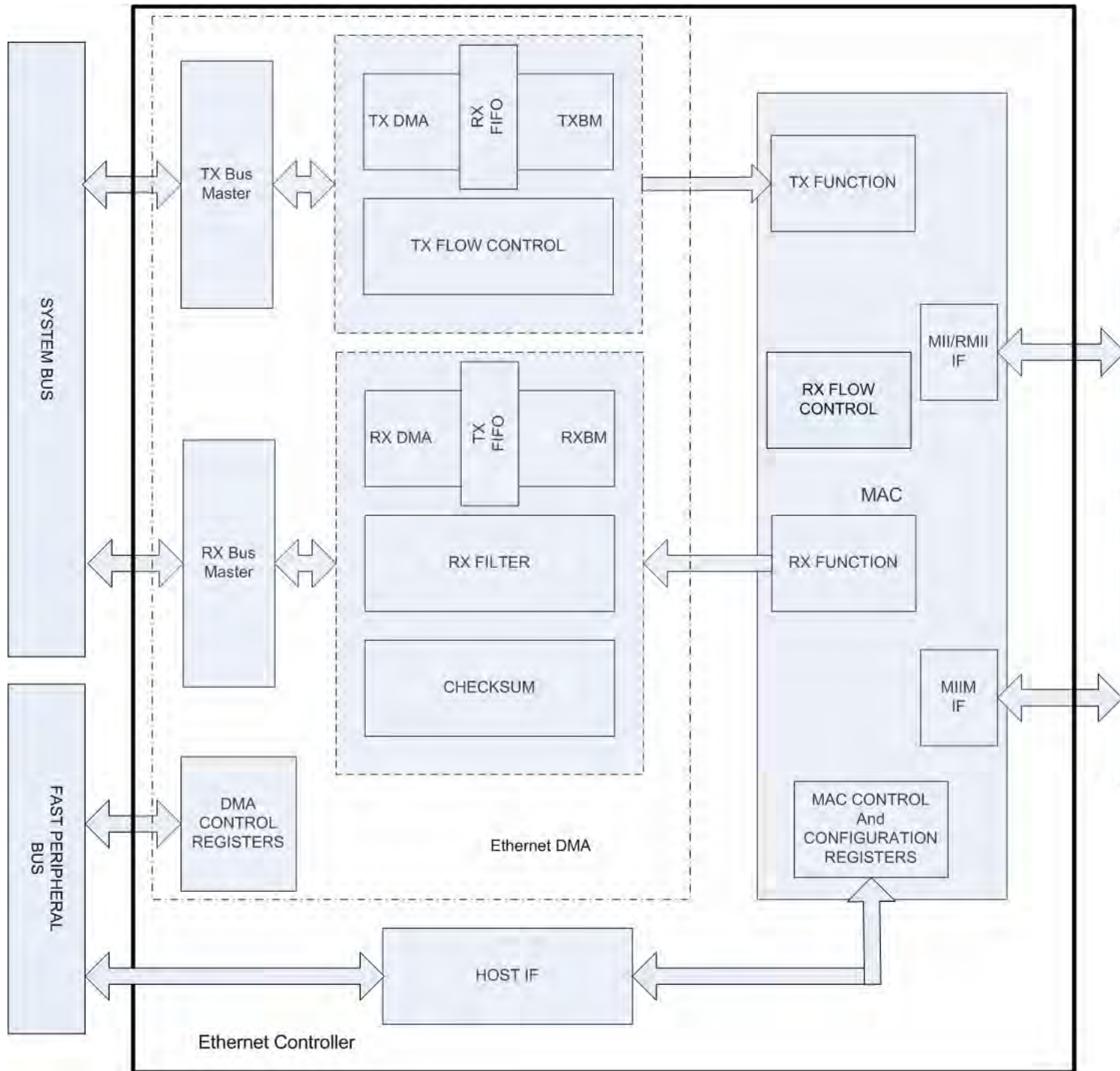


Figure 1: Ethernet Controller Block Diagram

For completeness, we also need to look at the interface diagram of a representative Ethernet PHY. As shown in Figure 2, the PHY has two interfaces, one for configuring and managing the PHY (SMI/MIIM) and another for transmit and receive data (RMII or MII). The SMI/MIIM interface is the responsibility of the Ethernet PHY Driver Library. When setting up the Ethernet PHY this library calls primitives from the Ethernet PHY Driver library. The RMII/MII interface is the responsibility of the Ethernet MAC Driver Library (this library).

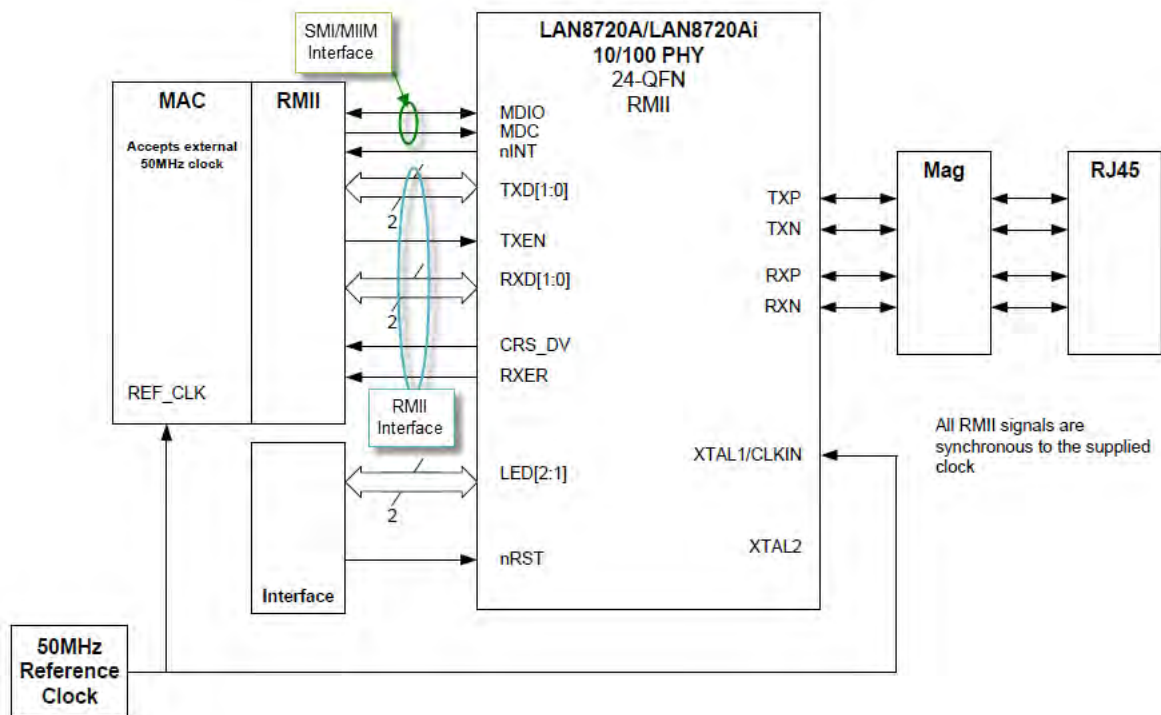


Figure 2: Ethernet PHY Interfaces

5.1.3.4.2 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Ethernet MAC Driver Library.

Library Interface Section	Description
Client Level Functions	PIC32MACOpen, PIC32MACClose, and PIC32MACSetup to support the TCP/IP Stack. Plus link status and power options.
Get and Put Functions	Get/Put functions for bytes, byte arrays, and Ethernet headers.
Base Address and Buffer Size Functions	Base address and sizes for buffers.
Receive Functions	Receive routines.
Transmit Functions	Transmit routines.
Events	Ethernet event support routines.
Other Functions	Checksum support, memory copying, version support.
Data Types and Constants	Typedefs and #defines.

5.1.3.4.3 MPLAB Harmony vs. Unified Stack Functions

The following is a cross-reference linking MPLAB Harmony functions with functions from the legacy TCP/IP Stack:

MPLAB Harmony Function:	MCHP_tcpip_unified Function:
DRV_ETHMAC_BaseAddrTxGet	PIC32MACGetTxBaseAddr
DRV_ETHMAC_RxPacketDiscard	PIC32MACDiscardRx
DRV_ETHMAC_EventAck	PIC32MACEventAck
DRV_ETHMAC_EventDeInit	PIC32MACEventDeInit
DRV_ETHMAC_EventInit	PIC32MACEventInit
DRV_ETHMAC_EventNotifyClear	PIC32TCPIP_MAC_EventNotifyClear
DRV_ETHMAC_EventNotifyHandlerSet	PIC32MACEventSetNotifyHandler
DRV_ETHMAC_EventNotifySet	PIC32MACEventSetNotifyEvents
DRV_ETHMAC_EventPendingGet	PIC32MACEventGetPending
DRV_ETHMAC_ByteArrayGet	PIC32MACGetArray
DRV_ETHMAC_HeaderGet	PIC32MACGetHeader
DRV_ETHMAC_IPBufferChecksumCalc	PIC32MACCalcIPBufferChecksum
DRV_ETHMAC_LinkCheck	PIC32MACCheckLink
DRV_ETHMAC_MemoryCopyIsDone	PIC32MACIsMemCopyDone
DRV_ETHMAC_PIC32MACClose	PIC32MACClose
DRV_ETHMAC_PIC32MACOpen	PIC32MACOpen
DRV_ETHMAC_PIC32MACSetup	PIC32MACInitialize
DRV_ETHMAC_PIC32MACTearDown	PIC32MACDeinitialize
DRV_ETHMAC_PowerMode	PIC32MACPowerMode
DRV_ETHMAC_ByteArrayPut	PIC32MACPutArray
DRV_ETHMAC_HeaderPut	PIC32MACPutHeader
DRV_ETHMAC_ReadPointerBaseSet	PIC32MACSetBaseReadPtr
DRV_ETHMAC_ReadPointerSet	PIC32MACSetReadPtr
DRV_ETHMAC_RxChecksumCalc	PIC32MACCalcRxChecksum
DRV_ETHMAC_RxHashTableEntrySet	PIC32MACSetRXHashTableEntry
DRV_ETHMAC_RxReadPointerGet	PIC32MACGetReadPtrInRx
DRV_ETHMAC_RxReadPointerSet	PIC32MACSetReadPtrInRx
DRV_ETHMAC_TxBufferFlush	PIC32MACFlush
DRV_ETHMAC_TxIsReady	PIC32MACIsTxReady
DRV_ETHMAC_VersionGet	-none-
DRV_ETHMAC_VersionStrGet	-none-
DRV_ETHMAC_WritePointerSet	PIC32MACSetWritePtr

MCHP_tcpip_unified Function	MPLAB Harmony Function:
PIC32MACCalcIPBufferChecksum	DRV_ETHMAC_IPBufferChecksumCalc
PIC32MACCalcRxChecksum	DRV_ETHMAC_RxChecksumCalc
PIC32MACCheckLink	DRV_ETHMAC_LinkCheck
PIC32MACDeinitialize	DRV_ETHMAC_PIC32MACTeardown
PIC32MACDiscardRx	DRV_ETHMAC_RxPacketDiscard
PIC32MAEventAck	DRV_ETHMAC_EventAck
PIC32MAC_EventNotifyClear	DRV_ETHMAC_EventNotifyClear
PIC32MAEventDelInit	DRV_ETHMAC_EventDelInit
PIC32MAEventGetPending	DRV_ETHMAC_EventPendingGet
PIC32MAEventInit	DRV_ETHMAC_EventInit
PIC32MAEventSetNotifyEvents	DRV_ETHMAC_EventNotifySet
PIC32MAEventSetNotifyHandler	DRV_ETHMAC_EventNotifyHandlerSet
PIC32MACFlush	DRV_ETHMAC_TxBufferFlush
PIC32MACGetArray	DRV_ETHMAC_ByteArrayGet
PIC32MACGetHeader	DRV_ETHMAC_HeaderGet
PIC32MACGetReadPtrInRx	DRV_ETHMAC_RxReadPointerGet
PIC32MACGetTxBaseAddr	DRV_ETHMAC_BaseAddrTxGet
PIC32MACIsMemCopyDone	DRV_ETHMAC_MemoryCopyIsDone
PIC32MACIsTxReady	DRV_ETHMAC_TxIsReady
PIC32MACDeinitialize	DRV_ETHMAC_PIC32MACTeardown
PIC32MACInitialize	DRV_ETHMAC_PIC32MACSetup
PIC32MACOpen	DRV_ETHMAC_PIC32MACOpen
PIC32MACClose	DRV_ETHMAC_PIC32MACClose
PIC32MACPowerMode	DRV_ETHMAC_PowerMode
PIC32MACPutArray	DRV_ETHMAC_ByteArrayPut
PIC32MACPutHeader	DRV_ETHMAC_HeaderPut
PIC32MACSetBaseReadPtr	DRV_ETHMAC_ReadPointerBaseSet
PIC32MACSetRXHashTableEntry	DRV_ETHMAC_RxHashTableEntrySet
PIC32MACSetReadPtr	DRV_ETHMAC_ReadPointerSet
PIC32MACSetReadPtrInRx	DRV_ETHMAC_RxReadPointerSet
PIC32MACSetWritePtr	DRV_ETHMAC_WritePointerSet
	DRV_ETHMAC_VersionGet
	DRV_ETHMAC_VersionStrGet

5.1.3.4.4 Support for Legacy "Ethernet Controller Library"

These routines (DRV_ETHMAC_Legacy*) provide MPLAB Harmony support for applications that use the "Ethernet Controller Library for Microchip PIC32MX Microcontrollers" that is installed with XC32 compilers. (See Microchip\xc32\v1.20\docs\pic32-lib-help\hlpETH.chm for information on this legacy support.)

For some functions the argument list is the same, but for others the addition of an index value from the ETH_MODULE_ID enumeration has been added to make the function work from within the context of this MPLAB Harmony driver.

Legacy Controller Library	MPLAB Harmony
EthClose	DRV_ETHMAC_LegacyClose
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthEventsClr	PLIB_ETH_LegacyEventsClr
EthEventsEnableClr	PLIB_ETH_LegacyEventsEnableClr
EthEventsEnableGet	PLIB_ETH_LegacyEventsEnableGet
EthEventsEnableSet	PLIB_ETH_LegacyEventsEnableSet
EthEventsEnableWrite	PLIB_ETH_LegacyEventsEnableWrite
EthEventsGet	PLIB_ETH_LegacyEventsGet
EthInit	DRV_ETHMAC_LegacyInit
EthMACGetAddress	PLIB_ETH_LegacyMACGetAddress
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthMACSetAddress	PLIB_ETH_LegacyMACSetAddress
EthMACSetMaxFrame	PLIB_ETH_LegacyMACSetMaxFrame
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxFiltersClr	PLIB_ETH_LegacyRxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_LegacyRxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_LegacyRxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_LegacyRxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_LegacyRxFiltersSet
EthRxFiltersWrite	PLIB_ETH_LegacyRxFiltersWrite
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthRxSetBufferSize	PLIB_ETH_LegacyRxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_LegacyStatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_LegacyStatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_LegacyStatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_LegacyStatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_LegacyStatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_LegacyStatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_LegacyStatTxSColCnt
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

MAC Driver Function	Legacy Controller Library
DRV_ETHMAC_LegacyClose	EthClose
DRV_ETHMAC_LegacyDescriptorGetBuffer	EthDescriptorGetBuffer
DRV_ETHMAC_LegacyDescriptorsPoolAdd	EthDescriptorsPoolAdd
DRV_ETHMAC_LegacyDescriptorsPoolCleanUp	EthDescriptorsPoolCleanUp
DRV_ETHMAC_LegacyDescriptorsPoolRemove	EthDescriptorsPoolRemove
DRV_ETHMAC_LegacyInit	EthInit
DRV_ETHMAC_LegacyMACOpen	EthMACOpen
DRV_ETHMAC_LegacyRxAcknowledgeBuffer	EthRxAcknowledgeBuffer
DRV_ETHMAC_LegacyRxAcknowledgePacket	EthRxAcknowledgePacket
DRV_ETHMAC_LegacyRxBuffersAppend	EthRxBuffersAppend
DRV_ETHMAC_LegacyRxGetBuffer	EthRxGetBuffer
DRV_ETHMAC_LegacyRxGetPacket	EthRxGetPacket
DRV_ETHMAC_LegacyTxAcknowledgeBuffer	EthTxAcknowledgeBuffer
DRV_ETHMAC_LegacyTxAcknowledgePacket	EthTxAcknowledgePacket
DRV_ETHMAC_LegacyTxGetBufferStatus	EthTxGetBufferStatus
DRV_ETHMAC_LegacyTxGetPacketStatus	EthTxGetPacketStatus
DRV_ETHMAC_LegacyTxSendBuffer	EthTxSendBuffer
DRV_ETHMAC_LegacyTxSendPacket	EthTxSendPacket

5.1.3.5 Configuring the Library

Macros

	Name	Description
	DRV_ETHMAC_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHMAC_INDEX	Ethernet MAC static index selection.
	DRV_ETHMAC_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHMAC_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_ETHMAC_INTERRUPT_SOURCE	Defines an override of the interrupt source in case of static driver.
	DRV_ETHMAC_PERIPHERAL_ID	Defines an override of the peripheral ID.
	DRV_ETHMAC_POWER_STATE	Defines an override of the power state of the Ethernet MAC driver.

Description

The configuration of the Ethernet MAC Driver is based on the file `sys_config.h`

This header file contains the configuration selection for the Ethernet MAC Driver. Based on the selections made, the Ethernet MAC Driver will support or not support selected features. These configuration settings will apply to all instances of the Ethernet MAC Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.3.5.1 DRV_ETHMAC_CLIENTS_NUMBER Macro

C

```
#define DRV_ETHMAC_CLIENTS_NUMBER 1
```

Description

Ethernet MAC Maximum Number of Clients

This definition select the maximum number of clients that the Ethernet MAC driver can support at run time. Not defining it means using a single client.

Remarks

None.

5.1.3.5.2 DRV_ETHMAC_INDEX Macro

C

```
#define DRV_ETHMAC_INDEX DRV_ETHMAC_INDEX_1
```

Description

Ethernet MAC Static Index Selection

This definition selects the Ethernet MAC static index for the driver object reference

Remarks

This index is required to make a reference to the driver object.

5.1.3.5.3 DRV_ETHMAC_INSTANCES_NUMBER Macro

C

```
#define DRV_ETHMAC_INSTANCES_NUMBER 1
```

Description

Ethernet MAC hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver. Not defining it means using a static driver.

Remarks

None.

5.1.3.5.4 DRV_ETHMAC_INTERRUPT_MODE Macro

C

```
#define DRV_ETHMAC_INTERRUPT_MODE true
```

Description

Ethernet MAC Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of timer operation is desired
- false - Select if polling mode of timer operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

5.1.3.5.5 DRV_ETHMAC_INTERRUPT_SOURCE Macro

C

```
#define DRV_ETHMAC_INTERRUPT_SOURCE INT_SOURCE_ETH_1
```

Description

Ethernet MAC Interrupt Source

Defines an override of the interrupt source in case of static driver.

Remarks

Refer to the INT PLIB document for more information on [INT_SOURCE](#) enumeration.

5.1.3.5.6 DRV_ETHMAC_PERIPHERAL_ID Macro

C

```
#define DRV_ETHMAC_PERIPHERAL_ID ETHMAC_ID_1
```

Description

Ethernet MAC Peripheral ID Selection

Defines an override of the peripheral ID, using macros.

Remarks

Some devices also support ETHMAC_ID_0

5.1.3.5.7 DRV_ETHMAC_POWER_STATE Macro

C

```
#define DRV_ETHMAC_POWER_STATE SYS_MODULE_POWER_IDLE_STOP
```

Description

Ethernet MAC power state configuration

Defines an override of the power state of the Ethernet MAC driver.

Remarks

This feature may not be available in the device or the Ethernet MAC module selected.

5.1.3.6 Building the Library

This section list the files that are available in the \src of the Ethernet MAC Driver. It lists which files need to be included in the

build based on either a hardware feature present on the board or configuration option selected by the system.

5.1.3.7 Library Interface

Data Types and Constants

	Name	Description
	ETH_CLOSE_FLAGS	Defines the possible disable codes of Ethernet controller "DRV_ETHMAC_LegacyClose" call.
	ETH_LINK_STATUS	Defines the possible status flags of Ethernet link.
	ETH_MODULE_STATUS	Defines the possible status codes of the Ethernet controller.
	ETH_OPEN_FLAGS	Supported open configuration flags for the Ethernet module (EthMACOpen).
	ETH_PAUSE_TYPE	Defines the possible Ethernet MAC pause types.
	ETH_RESULT_CODE	Defines the possible results of Ethernet operations that can succeed or fail
	ETHPHY_CONFIG_FLAGS	flags for DRV_ETHPHY_Setup() call
	DRV_ETHMAC_INDEX_1	This is macro DRV_ETHMAC_INDEX_1.
	DRV_ETHPHY_INDEX_1	This is macro DRV_ETHPHY_INDEX_1.

Client Level Functions

	Name	Description
	DRV_ETHMAC_PIC32MACClose	Closes a client instance of the PIC32 MAC Driver.
	DRV_ETHMAC_PIC32MACGetConfig	Supports PIC32 Ethernet MAC by copying its configuration..
	DRV_ETHMAC_PIC32MACLinkCheck	Checks current link status.
	DRV_ETHMAC_PIC32MACOpen	Opens a client instance of the PIC32 MAC Driver.
	DRV_ETHMAC_PIC32MACSetup	Supports PIC32 Ethernet MAC setup.
	DRV_ETHMAC_PIC32MACTeardown	Supports PIC32 Ethernet MAC Teardown (opposite of setup).

Event Functions

	Name	Description
	DRV_ETHMAC_PIC32MACEventAcknowledge	This function acknowledges and re-enables processed events. Multiple events can be orr-ed together as they are processed together. The events acknowledged by this function should be the events that have been retrieved from the stack by calling DRV_ETHMAC_PIC32MACEventPendingGet() or have been passed to the user by the stack using the notification handler (PIC32MACEventSetNotifyHandler()) and have been processed and have to be re-enabled.
	DRV_ETHMAC_PIC32MACEventMaskSet	Enables the MAC events.
	DRV_ETHMAC_PIC32MACEventPendingGet	Returns the currently pending events.

Other Functions

	Name	Description
	DRV_ETHMAC_PIC32MACPowerMode	Powers down the Ethernet MAC.
	DRV_ETHMAC_PIC32MACProcess	MAC periodic processing function.

Receive Functions

	Name	Description
	DRV_ETHMAC_PIC32MACPacketRx	A packet is returned if such a pending packet exists.
	DRV_ETHMAC_PIC32MACPacketTx	A packet is submitted to the MAC driver for transmission.

Transmit Functions

	Name	Description
	DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet	Calculates a CRC-32 and sets the appropriate bit in the ETHHTx registers

Description

5.1.3.7.1 Client Level Functions

5.1.3.7.1.1 DRV_ETHMAC_PIC32MACClose Function

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACClose(
    TCPIP_MAC_HANDLE hMac
);
```

Description

This function closes a client instance of the PIC32 MAC Driver.

Preconditions

[DRV_ETHMAC_PIC32MACOpen](#) should have been called.

Parameters

Parameters	Description
macId	MAC identification, from the TCPIP_STACK_MODULE enumeration

Returns

TCPIP_MAC_RES_OK if initialization completed; otherwise, the error enumeration value.

Example

5.1.3.7.1.2 DRV_ETHMAC_PIC32MACGetConfig Function

C

```
size_t DRV_ETHMAC_PIC32MACGetConfig(
    TCPIP_STACK_MODULE modId,
    void* configBuff,
    size_t buffSize,
    size_t* pConfigSize
);
```

Description

Supports PIC32 Ethernet MAC Teardown (opposite of Setup). Used by tcpip_module_manager.

Preconditions

Supports PIC32 Ethernet MAC by copying its configuration..

Parameters

Parameters	Description
modId	Module ID from TCPIP_STACK_MODULE enumeration, identifying MAC.
configBuff	Pointer to configuration buffer to be copied

buffSize	Size of configuration buffer provided by configBuff
pConfigSize	Pointer to size of configuration buffer copied.

Returns

Size of configuration buffer copied.

Remarks

This function deinitializes the Eth controller, the MAC and the associated PHY. It should be called to be able to schedule any Eth transmit or receive operation.

Example**5.1.3.7.1.3 DRV_ETHMAC_PIC32MACLinkCheck Function****C**

```
bool DRV_ETHMAC_PIC32MACLinkCheck(  
    TCPIP_MAC_HANDLE hMac  
);
```

Description

This function checks the link status, performing a MAC reconfiguration if the link went up after being down.

Preconditions

None.

Parameters

Parameters	Description
hMac	Ethernet MAC client handle

Returns

- true - If the link is up
- false - If the link is not up

Remarks

If auto negotiation is enabled the MAC we may have to be reconfigured.

Example**5.1.3.7.1.4 DRV_ETHMAC_PIC32MACOpen Function****C**

```
TCPIP_MAC_HANDLE DRV_ETHMAC_PIC32MACOpen(  
    TCPIP_STACK_MODULE macId  
);
```

Description

This function opens a client instance of the PIC32 MAC Driver. Used by tcpip_module_manager.

Preconditions

[DRV_ETHMAC_PIC32MACSetup](#) should have been called.

Parameters

Parameters	Description
macId	MAC identification, from the TCPIP_STACK_MODULE enumeration

Returns

[TCPIP_MAC_HANDLE](#) - handle (pointer) to MAC client

Example**5.1.3.7.1.5 DRV_ETHMAC_PIC32MACSetup Function****C**

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACSetup(  
    TCPIP_MAC_MODULE_CTRL* const stackData,  
    const TCPIP_MODULE_MAC_PIC32INT_CONFIG* initData  
);
```

Description

This function supports setup of the PIC32 Ethernet MAC. Used by tcpip_module_manager.

Preconditions

DRV_ETHMAC_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
pStackData	Pointer to stack data
initData	Pointer to initialization data

Returns

TCPIP_MAC_RES_OK if initialization completed; otherwise, error enumeration value.

Remarks

This function initializes the Ethernet controller, the MAC and the associated PHY. It should be called to be able to schedule any Ethernet transmit or receive operation.

Example**5.1.3.7.1.6 DRV_ETHMAC_PIC32MACTeardown Function****C**

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACTeardown(  
    const TCPIP_MAC_MODULE_CTRL* const stackData  
);
```

Description

This function supports teardown of the PIC32 Ethernet MAC (opposite of setup). Used by tcpip_module_manager.

Preconditions

[DRV_ETHMAC_PIC32MACSetup](#) must have been called to setup the driver.

Parameters

Parameters	Description
pStackData	Pointer to Stack Data

Returns

TCPIP_MAC_RES_OK if initialization completed, error enumeration value otherwise.

Remarks

This function deinitializes the Ethernet controller, the MAC and the associated PHY. It should be called to be able to schedule any Ethernet transmit or receive operation.

Example**5.1.3.7.2 Get and Put Functions****5.1.3.7.3 Base Address and Buffer Size Functions****5.1.3.7.4 Receive Functions****5.1.3.7.4.1 DRV_ETHMAC_PIC32MACPacketRx Function****C**

```
TCPIP_MAC_PACKET* DRV_ETHMAC_PIC32MACPacketRx(
    TCPIP_MAC_HANDLE hMac,
    TCPIP_MAC_RES* pRes,
    const TCPIP_MAC_PACKET_RX_STAT** ppPktStat
);
```

Description

This is the MAC receive function.

Once a pending packet is available in the MAC driver internal RX queues this function will dequeue the packet and hand it over to the MAC driver's client - i.e. the stack - for further processing.

The flags for a RX packet have to be updated by the MAC driver:

- TCPIP_MAC_PKT_FLAG_RX has to be set If the MAC supports it, it should set:
 - TCPIP_MAC_PKT_FLAG_UNICAST has to be set if that packet is a unicast packet
 - TCPIP_MAC_PKT_FLAG_BCAST has to be set if that packet is a broadcast packet
 - TCPIP_MAC_PKT_FLAG_MCAST has to be set if that packet is a multicast packet
- TCPIP_MAC_PKT_FLAG_QUEUED has to be set
- TCPIP_MAC_PKT_FLAG_SPLIT has to be set if the packet has multiple data segments

Additional information about the packet is available by providing the pRes and ppPktStat fields.

Preconditions

[DRV_ETHMAC_PIC32MACSetup\(\)](#) should have been called.

Parameters

Parameters	Description
hMac	Ethernet MAC client handle
pRes	optional pointer to an address that will receive an additional result associated with the operation. Can be 0 if not needed.
ppPktStat	optional pointer to an address that will receive the received packet status. Note that this pointer cannot be used once the packet acknowledgement function was called. Can be 0 if not needed.

Returns

a valid pointer to an available RX packet 0 if no packet pending/available

Remarks

The MAC driver dequeues and return to the caller just one single packet.

Once the higher level layers in the stack are done with processing the RX packet, they have to call the corresponding packet acknowledgement function that tells the MAC driver that it can resume control of that packet.

Once the stack modules are done processing the RX packets and the acknowledge function is called the MAC driver will reuse the RX packets.

The MAC driver may use the [DRV_ETHMAC_PIC32MACProcess\(\)](#) for obtaining new RX packets if needed.

Example**5.1.3.7.4.2 DRV_ETHMAC_PIC32MACPacketTx Function****C**

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACPacketTx(
    TCPIP_MAC_HANDLE hMac,
    TCPIP_MAC_PACKET * ptrPacket
);
```

Description

This is the MAC transmit function. The MAC driver has to support internal queueing! A packet is to be rejected only if it's not properly formatted. Otherwise it has to be scheduled for transmission in an internal queue!

Once the packet is scheduled for transmission the MAC driver has to set the TCPIP_MAC_PKT_FLAG_QUEUED flag so that the stack is aware that this packet is under processing and cannot be modified!

Once the packet is transmitted, the TCPIP_MAC_PKT_FLAG_QUEUED has to be cleared, the proper packet acknowledgement result (ackRes) has to be set and the packet acknowledgement function (ackFunc) has to be called. It is implementation dependant if all these steps are implemented as part of the ackFunc itself or as discrete steps.

Preconditions

[DRV_ETHMAC_PIC32MACSetup\(\)](#) should have been called.

Parameters

Parameters	Description
hMac	Ethernet MAC client handle
ptrPacket	pointer to a TCPIP_MAC_PACKET that's completely formatted and ready to be transmitted over the network

Returns

TCPIP_MAC_RES_OK if the packet transmitted, [TCPIP_MAC_RES](#) errorcode for not properly formatted packets, etc.

Example**5.1.3.7.5 Transmit Functions****5.1.3.7.5.1 DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet Function****C**

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet(
```

```
TCPIP_MAC_HANDLE hMac,
TCPIP_MAC_ADDR* DestMACAddr
);
```

Description

This function calculates a CRC-32 using polynomial 0x4C11DB7 and then, using bits 28 through 23 of the CRC, sets the appropriate bit in the ETHHT0-ETHHT1 registers.

Preconditions

[DRV_ETHMAC_PIC32MACSetup\(\)](#) should have been called.

Parameters

Parameters	Description
hMac	Ethernet MAC client handle
DestMACAddr	6-byte group destination MAC address to allow through the Hash Table Filter. If DestMACAddr is set to '0000 0000 0000', the hash table will be cleared of all entries and the filter will be disabled.

Returns

TCPIP_MAC_RES_OK if success, an error code otherwise.

Remarks

Sets the appropriate bit in the ETHHT0/1 registers to allow packets sent to DestMACAddr to be received and enabled the Hash Table receive filter

There is no way to individually unset destination MAC addresses from the hash table since it is possible to have a hash collision and therefore multiple MAC addresses relying on the same hash table bit. The stack would have to individually store each 6 byte MAC address to support this feature, which would waste a lot of RAM and be unnecessary in most applications. As a simple compromise, you can call [DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet\(\)](#) using a 00-00-00-00-00-00 destination MAC address, which will clear the entire hash table and disable the hash table filter. This will allow you to then readd the necessary destination addresses.

Example

5.1.3.7.6 Event Functions

5.1.3.7.6.1 DRV_ETHMAC_PIC32MACTEventAcknowledge Function

C

```
bool DRV_ETHMAC_PIC32MACTEventAcknowledge(
    TCPIP_MAC_HANDLE hMac,
    TCPIP_MAC_EVENT tcpAckEv
);
```

Description

This function acknowledges and re-enables processed events. Multiple events can be orr-ed together as they are processed together. The events acknowledged by this function should be the events that have been retrieved from the stack by calling [DRV_ETHMAC_PIC32MACTEventPendingGet\(\)](#) or have been passed to the user by the stack using the notification handler ([PIC32MACTEventSetNotifyHandler\(\)](#)) and have been processed and have to be re-enabled.

Preconditions

[DRV_ETHMAC_PIC32MACSetup\(\)](#) should have been called.

Parameters

Parameters	Description
hMac	Ethernet MAC client handle
tcpAckEv	the events that the user processed and need to be re-enabled

Returns

true if events acknowledged false if no events to be acknowledged

Remarks

All events should be acknowledged, in order to be re-enabled.

Some events are fatal errors and should not be acknowledged (TCPIP_MAC_EV_RX_BUSERR, TCPIP_MAC_EV_TX_BUSERR). Stack re-initialization is needed under such circumstances.

Some events are just system/application behavior and they are intended only as simple info (TCPIP_MAC_EV_RX_OVFLOW, TCPIP_MAC_EV_RX_BUFNA, TCPIP_MAC_EV_TX_ABORT, TCPIP_MAC_EV_RX_ACT).

The TCPIP_MAC_EV_RX_FWMARK and TCPIP_MAC_EV_RX_EWMARK events are part of the normal flow control operation (if auto flow control was enabled). They should be enabled alternatively, if needed.

The events are persistent. They shouldn't be re-enabled unless they have been processed and the condition that generated them was removed. Re-enabling them immediately without proper processing will have dramatic effects on system performance.

Example

```
DRV_ETHMAC_PIC32MACEventAcknowledge( hMac, stackNewEvents );
```

5.1.3.7.6.2 DRV_ETHMAC_PIC32MACEventMaskSet Function**C**

```
bool DRV_ETHMAC_PIC32MACEventMaskSet(  
    TCPIP_MAC_HANDLE hMac,  
    TCPIP_MAC_EVENT macEvents,  
    bool enable  
);
```

Description

This function sets the enabled events. Multiple events can be orr-ed together. All events that are set will be added to the notification process. The old events will be disabled. The stack (or stack user) has to catch the events that are notified and process them:

- The stack should process the TCPIP_MAC_EV_RX_PKTEND/TCPIP_MAC_EV_RX_DONE, TCPIP_MAC_EV_TX_DONE transfer events
- Process the specific condition and acknowledge them calling [DRV_ETHMAC_PIC32MACEventAcknowledge\(\)](#) so that they can be re-enabled.

Preconditions

[DRV_ETHMAC_PIC32MACSetup\(\)](#) should have been called.

Parameters

Parameters	Description
hMac	Ethernet MAC client handle
macEvMask	events the user of the stack wants to add/delete for notification
enable	if true, the events will be enabled, else disabled

Returns

always true, operation succeeded.

Remarks

The event notification system enables the user of the TCPIP stack to call into the stack for processing only when there are relevant events rather than being forced to periodically call from within a loop.

If the notification events are null the interrupt processing will be disabled. Otherwise the event notification will be enabled and the interrupts relating to the requested events will be enabled.

Note that once an event has been caught by the stack ISR (and reported if a notification handler is in place) it will be disabled until the [DRV_ETHMAC_PIC32MACEventAcknowledge\(\)](#) is called.

Example

```
DRV_ETHMAC_PIC32MACEventMaskSet( hMac, TCPIP_MAC_EV_RX_OVFLOW | TCPIP_MAC_EV_RX_BUFNA,  
true );
```

5.1.3.7.6.3 DRV_ETHMAC_PIC32MACEventPendingGet Function**C**

```
TCPIP_MAC_EVENT DRV_ETHMAC_PIC32MACEventPendingGet(  
    TCPIP_MAC_HANDLE hMac  
);
```

Description

This function returns the currently pending events belonging to a group. Multiple events can be orr-ed together as they accumulate. The stack should be called for processing whenever a stack managed event (TCPIP_MAC_EV_RX_PKTEND, TCPIP_MAC_EV_TX_DONE) is present. The other, non critical events, may not be managed by the stack and passed to an user. They will have to be eventually acknowledged if re-enabling is needed.

Preconditions

[DRV_ETHMAC_PIC32MACSetup](#) should have been called.

Parameters

Parameters	Description
hMac	parameter identifying the intended MAC client

Returns

The currently stack pending events.

Remarks

This is the preferred method to get the current pending MAC events. The stack maintains a proper image of the events from their occurrence to their acknowledgement.

Even with a notification handler in place it's better to use this function to get the current pending events rather than using the events passed by the notification handler which could be stale.

The events are persistent. They shouldn't be re-enabled unless they have been processed and the condition that generated them was removed. Re-enabling them immediately without proper processing will have dramatic effects on system performance.

The returned value is just a momentary value. The pending events can change any time.

Example

```
TCPIP_MAC_EVENT currEvents = DRV_ETHMAC_PIC32MACEventPendingGet( hMac);
```

5.1.3.7.7 Library Functions

5.1.3.7.8 Other Functions

5.1.3.7.8.1 DRV_ETHMAC_PIC32MACPowerMode Function

C

```
bool DRV_ETHMAC_PIC32MACPowerMode(  
    TCPIP_MAC_HANDLE hMac,  
    TCPIP_MAC_POWER_MODE pwrMode  
);
```

Description

This function powers down the Ethernet MAC.

Preconditions

None.

Parameters

Parameters	Description
hMac	Ethernet MAC client handle
pwrMode	Power Mode (?)

Returns

None.

Example

5.1.3.7.8.2 DRV_ETHMAC_PIC32MACProcess Function

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACProcess(  
    TCPIP_MAC_HANDLE hMac  
);
```

Description

This is a function that allows for internal processing by the MAC driver. It is meant for processing that cannot be done from within ISR.

Some of the processing that this is intended for:

- the MAC driver can process its pending TX queues (although it should do that preferably from within the TX ISR)
- RX buffers replenishing. If the number of packets in the RX queue falls below a specified limit, the MAC driver can use this function to allocate some extra RX packets. Similarly, if there are too many allocated RX packets, the MAC driver can free some of them.

Normally this function will be called in response to an TX and/or RX event signalled by the driver. This is specified by the MAC driver at initialization time using [TCPIP_MAC_MODULE_CTRL](#).

An alternative approach is that the MAC driver uses a system service to create a timer signal that will call the [TCPIP_MAC_Process](#) on a periodic basis.

Preconditions

[DRV_ETHMAC_PIC32MACSetup\(\)](#) should have been called.

Parameters

Parameters	Description
hMac	Ethernet MAC client handle

Returns

TCPIP_MAC_RES_OK if all processing went on OK a [TCPIP_MAC_RES](#) error code if processing failed for some reason

Remarks

The MAC driver may use the DRV_ETHMAC_PIC32MACProcess() for obtaining new RX packets if needed.

Example

5.1.3.7.9 Data Types and Constants

5.1.3.7.9.1 ETH_CLOSE_FLAGS Enumeration

C

```
typedef enum {  
    ETH_CLOSE_GRACEFUL,  
    ETH_CLOSE_DEFAULT = (0)  
} ETH_CLOSE_FLAGS;
```

Description

Ethernet Close Flags

This enumeration defines the close capabilities of the Ethernet module.

Members

Members	Description
ETH_CLOSE_GRACEFUL	Wait for the current TX/RX op to finish
ETH_CLOSE_DEFAULT = (0)	Default close options

5.1.3.7.9.2 ETH_LINK_STATUS Enumeration

C

```
typedef enum {  
    ETH_LINK_ST_DOWN,  
    ETH_LINK_ST_UP,  
    ETH_LINK_ST_LP_NEG_UNABLE,  
    ETH_LINK_ST_REMOTE_FAULT,  
    ETH_LINK_ST_PDF,  
    ETH_LINK_ST_LP_PAUSE,  
    ETH_LINK_ST_LP_ASM_DIR,  
    ETH_LINK_ST_NEG_TMO,  
    ETH_LINK_ST_NEG_FATAL_ERR  
} ETH_LINK_STATUS;
```

Description

Ethernet Link Status Codes

This enumeration defines the flags describing the status of the Ethernet link.

Members

Members	Description
ETH_LINK_ST_DOWN	No connection to the LinkPartner
ETH_LINK_ST_UP	Link is up
ETH_LINK_ST_LP_NEG_UNABLE	LP non negotiation able
ETH_LINK_ST_REMOTE_FAULT	LP fault during negotiation
ETH_LINK_ST_PDF	Parallel Detection Fault encountered (when ETH_LINK_ST_LP_NEG_UNABLE)
ETH_LINK_ST_LP_PAUSE	LP supports symmetric pause
ETH_LINK_ST_LP_ASM_DIR	LP supports asymmetric TX/RX pause operation
ETH_LINK_ST_NEG_TMO	LP not there
ETH_LINK_ST_NEG_FATAL_ERR	An unexpected fatal error occurred during the negotiation

Remarks

Multiple flags can be set.

5.1.3.7.9.3 ETH_MODULE_STATUS Enumeration

C

```
typedef enum {
    ETH_ST_RXBUSY,
    ETH_ST_TXBUSY,
    ETH_ST_BUSY
} ETH_MODULE_STATUS;
```

Description

Ethernet Controller Status Codes

This enumeration defines the flags describing the status of the Ethernet controller.

Members

Members	Description
ETH_ST_RXBUSY	A packet is currently received
ETH_ST_TXBUSY	A packet is currently transmitted
ETH_ST_BUSY	Module is on or completing a transaction

5.1.3.7.9.4 ETH_OPEN_FLAGS Enumeration

C

```
typedef enum {
    ETH_OPEN_AUTO,
    ETH_OPEN_FDUPLEX,
    ETH_OPEN_HDUPLEX,
    ETH_OPEN_100,
    ETH_OPEN_10,
    ETH_OPEN_HUGE_PKTS,
    ETH_OPEN_MAC_LOOPBACK,
    ETH_OPEN_PHY_LOOPBACK,
    ETH_OPEN_MDIX_AUTO,
    ETH_OPEN_MDIX_NORM,
    ETH_OPEN_MDIX_SWAP,
    ETH_OPEN_RMII,
    ETH_OPEN_MII,
    ETH_OPEN_DEFAULT = (ETH_OPEN_AUTO|ETH_OPEN_FDUPLEX|ETH_OPEN_HDUPLEX|
    ETH_OPEN_100|ETH_OPEN_10|ETH_OPEN_MDIX_AUTO)
```



```
} ETH_OPEN_FLAGS;
```

Description

Ethernet Open Configuration Settings

This enumeration defines the various configuration options for the Ethernet module. These values can be ORed together to create a configuration mask passed to the EthMACOpen routine.

Members

Members	Description
ETH_OPEN_AUTO	Use auto negotiation. set the following flags to specify your choices
ETH_OPEN_FDUPLEX	Use full duplex or full duplex negotiation capability needed
ETH_OPEN_HDUPLEX	Use half duplex or half duplex negotiation capability needed
ETH_OPEN_100	Use 100MBps or 100MBps negotiation capability needed
ETH_OPEN_10	Use 10MBps or 10MBps negotiation capability needed
ETH_OPEN_HUGE_PKTS	Allow huge packets RX/TX
ETH_OPEN_MAC_LOOPBACK	Loopbacked at the MAC level
ETH_OPEN_PHY_LOOPBACK	When PHY is loopback-ed, negotiation will be disabled!
ETH_OPEN_MDIX_AUTO	Use Auto MDIX
ETH_OPEN_MDIX_NORM	Use normal MDIX when Auto MDIX disabled
ETH_OPEN_MDIX_SWAP	Use swapped MDIX when Auto MDIX disabled
ETH_OPEN_RMII	RMII connection
ETH_OPEN_MII	MII connection
ETH_OPEN_DEFAULT = (ETH_OPEN_AUTO ETH_OPEN_FDUPLEX ETH_OPEN_HDUPLEX ETH_OPEN_100 ETH_OPEN_10 ETH_OPEN_MDIX_AUTO)	All capabilities default

Remarks

When Auto-negotiation is specified:

- If multiple capability flags are set (ETH_OPEN_FDUPLEX, ETH_OPEN_HDUPLEX, ETH_OPEN_100, ETH_OPEN_10) they are all advertised as this link side capabilities.
- If no setting is passed, the lowest one is taken, i.e., ETH_OPEN_HDUPLEX and ETH_OPEN_10.
- Auto-MDIX requires Auto-Negotiation; ETH_OPEN_MDIX_NORM or ETH_OPEN_MDIX_SWAP setting irrelevant.

When No Auto-negotiation is specified:

- If multiple settings, the highest priority setting is taken, i.e. ETH_OPEN_FDUPLEX over ETH_OPEN_HDUPLEX and ETH_OPEN_100 over ETH_OPEN_10.
- If no setting, the lowest setting is taken, i.e., ETH_OPEN_HDUPLEX and ETH_OPEN_10.
- The MDIX is set based on the ETH_OPEN_MDIX_NORM/ETH_OPEN_MDIX_SWAP setting.

5.1.3.7.9.5 ETH_PAUSE_TYPE Enumeration

C

```
typedef enum {
    ETH_MAC_PAUSE_TYPE_NONE,
    ETH_MAC_PAUSE_TYPE_PAUSE,
    ETH_MAC_PAUSE_TYPE_ASM_DIR,
```

```

    ETH_MAC_PAUSE_TYPE_EN_TX,
    ETH_MAC_PAUSE_TYPE_EN_RX,
    ETH_MAC_PAUSE_ALL = (ETH_MAC_PAUSE_TYPE_PAUSE|ETH_MAC_PAUSE_TYPE_ASM_DIR|
    ETH_MAC_PAUSE_TYPE_EN_TX|ETH_MAC_PAUSE_TYPE_EN_RX),
    ETH_MAC_PAUSE_CPBL_MASK = ETH_MAC_PAUSE_ALL
} ETH_PAUSE_TYPE;

```

Description

Ethernet MAC Pause Types

This enumeration defines the pause capabilities of the Ethernet MAC.

Members

Members	Description
ETH_MAC_PAUSE_TYPE_NONE	No PAUSE capabilities
ETH_MAC_PAUSE_TYPE_PAUSE	Supports symmetric PAUSE
ETH_MAC_PAUSE_TYPE_ASM_DIR	Supports ASM_DIR
ETH_MAC_PAUSE_TYPE_EN_TX	Enable MAC TX pause support
ETH_MAC_PAUSE_TYPE_EN_RX	Enable MAC RX pause support
ETH_MAC_PAUSE_ALL = (ETH_MAC_PAUSE_TYPE_PAUSE ETH_MAC_PAUSE_TYPE_ASM_DIR ETH_MAC_PAUSE_TYPE_EN_TX ETH_MAC_PAUSE_TYPE_EN_RX)	All types of pause
ETH_MAC_PAUSE_CPBL_MASK = ETH_MAC_PAUSE_ALL	All pause capabilities our MAC supports

5.1.3.7.9.6 ETH_RESULT_CODE Enumeration**C**

```

typedef enum {
    ETH_RES_OK,
    ETH_RES_NO_PACKET,
    ETH_RES_PACKET_QUEUED,
    ETH_RES_OUT_OF_MEMORY,
    ETH_RES_NO_DESCRIPTOR,
    ETH_RES_USPACE_ERR,
    ETH_RES_RX_SIZE_ERR,
    ETH_RES_RX_PKT_SPLIT_ERR,
    ETH_RES_NEGOTIATION_UNABLE,
    ETH_RES_NEGOTIATION_INACTIVE,
    ETH_RES_NEGOTIATION_NOT_STARTED,
    ETH_RES_NEGOTIATION_ACTIVE,
    ETH_RES_NEGOTIATION_LINKDOWN,
    ETH_RES_DTCT_ERR,
    ETH_RES_CPBL_ERR,
    ETH_RES_CFG_ERR
} ETH_RESULT_CODE;

```

Description

Ethernet Operation Result Codes

This enumeration defines the possible results of any of the Ethernet library operations that have the possibility of failing. This result should be checked to ensure that the operation achieved the desired result.

Members

Members	Description
ETH_RES_OK	Everything ok
ETH_RES_NO_PACKET	No such packet exist
ETH_RES_PACKET_QUEUED	Packet is queued (not transmitted or received and not processed)

ETH_RES_OUT_OF_MEMORY	Some memory allocation failed
ETH_RES_NO_DESCRIPTORs	Not enough descriptors available
ETH_RES_USPACE_ERR	We don't support user space buffers.
ETH_RES_RX_SIZE_ERR	The size of the receive buffers too small
ETH_RES_RX_PKT_SPLIT_ERR	A received packet spans more buffers/descriptors than supplied
ETH_RES_NEGOTIATION_UNABLE	No negotiation support
ETH_RES_NEGOTIATION_INACTIVE	No negotiation active
ETH_RES_NEGOTIATION_NOT_STARTED	Negotiation not started yet
ETH_RES_NEGOTIATION_ACTIVE	Negotiation active
ETH_RES_NEGOTIATION_LINKDOWN	Link down after negotiation, negotiation failed
ETH_RES_DTCT_ERR	No Phy was detected or it failed to respond to reset command
ETH_RES_CPBL_ERR	No match between the capabilities: the Phy supported and the open requested ones
ETH_RES_CFG_ERR	Hardware configuration doesn't match the requested open mode

5.1.3.7.9.7 ETHPHY_CONFIG_FLAGS Enumeration

C

```
typedef enum {
    ETH_PHY_CFG_RMII,
    ETH_PHY_CFG_MII,
    ETH_PHY_CFG_ALTERNATE,
    ETH_PHY_CFG_DEFAULT,
    ETH_PHY_CFG_AUTO
} ETHPHY_CONFIG_FLAGS;
```

Description

flags for [DRV_ETHPHY_Setup\(\)](#) call

Members

Members	Description
ETH_PHY_CFG_RMII	RMII data interface in configuration fuses.
ETH_PHY_CFG_MII	MII data interface in configuration fuses.
ETH_PHY_CFG_ALTERNATE	Configuration fuses is ALT
ETH_PHY_CFG_DEFAULT	Configuration fuses is DEFAULT
ETH_PHY_CFG_AUTO	Use the fuses configuration to detect if you are RMII/MII and ALT/DEFAULT configuration

5.1.3.7.9.8 DRV_ETHMAC_INDEX_1 Macro

C

```
#define DRV_ETHMAC_INDEX_1 1
```

Description

This is macro DRV_ETHMAC_INDEX_1.

5.1.3.7.9.9 DRV_ETHPHY_INDEX_1 Macro

C

```
#define DRV_ETHPHY_INDEX_1 1
```

Description

This is macro DRV_ETHPHY_INDEX_1.

5.1.3.8 Files

Files

Name	Description
drv_ethernet_flags.h	Ethernet driver configuration flags file.
drv_ethmac.h	Ethernet MAC device driver interface file.
drv_ethmac_config.h	Ethernet MAC driver configuration definitions template.

Description

5.1.3.8.1 drv_ethernet_flags.h

Ethernet Drivers Configuration Flags

This file provides the definition of commonly-used configuration enumerations for use with the Ethernet PHY and Ethernet MAC Drivers.

Enumerations

	Name	Description
	ETH_CLOSE_FLAGS	Defines the possible disable codes of Ethernet controller "DRV_ETHMAC_LegacyClose" call.
	ETH_LINK_STATUS	Defines the possible status flags of Ethernet link.
	ETH_OPEN_FLAGS	Supported open configuration flags for the Ethernet module (EthMACOpen).
	ETH_PAUSE_TYPE	Defines the possible Ethernet MAC pause types.
	ETH_RESULT_CODE	Defines the possible results of Ethernet operations that can succeed or fail
	ETHPHY_CONFIG_FLAGS	flags for DRV_ETHPHY_Setup() call

Company

Microchip Technology Inc.

FileName: drv_ethernet_flags.h

5.1.3.8.2 drv_ethmac.h

Ethernet MAC Device Driver Interface

The Ethernet MAC device driver provides a simple interface to manage the Ethernet peripheral. This file defines the interface definitions and prototypes for the Ethernet MAC driver.

Enumerations

	Name	Description
	ETH_MODULE_STATUS	Defines the possible status codes of the Ethernet controller.

Functions

	Name	Description
	<u>DRV_ETHMAC_PIC32MACClose</u>	Closes a client instance of the PIC32 MAC Driver.
	<u>DRV_ETHMAC_PIC32MACEventAcknowledge</u>	This function acknowledges and re-enables processed events. Multiple events can be orr-ed together as they are processed together. The events acknowledged by this function should be the events that have been retrieved from the stack by calling <u>DRV_ETHMAC_PIC32MACEventPendingGet()</u> or have been passed to the user by the stack using the notification handler (<u>PIC32MACEventSetNotifyHandler()</u>) and have been processed and have to be re-enabled.
	<u>DRV_ETHMAC_PIC32MACEventMaskSet</u>	Enables the MAC events.
	<u>DRV_ETHMAC_PIC32MACEventPendingGet</u>	Returns the currently pending events.
	<u>DRV_ETHMAC_PIC32MACGetConfig</u>	Supports PIC32 Ethernet MAC by copying its configuration..
	<u>DRV_ETHMAC_PIC32MACLinkCheck</u>	Checks current link status.
	<u>DRV_ETHMAC_PIC32MACOpen</u>	Opens a client instance of the PIC32 MAC Driver.
	<u>DRV_ETHMAC_PIC32MACPacketRx</u>	A packet is returned if such a pending packet exists.
	<u>DRV_ETHMAC_PIC32MACPacketTx</u>	A packet is submitted to the MAC driver for transmission.
	<u>DRV_ETHMAC_PIC32MACPowerMode</u>	Powers down the Ethernet MAC.
	<u>DRV_ETHMAC_PIC32MACProcess</u>	MAC periodic processing function.
	<u>DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet</u>	Calculates a CRC-32 and sets the appropriate bit in the ETHHTx registers
	<u>DRV_ETHMAC_PIC32MACSetup</u>	Supports PIC32 Ethernet MAC setup.
	<u>DRV_ETHMAC_PIC32MACTeardown</u>	Supports PIC32 Ethernet MAC Teardown (opposite of setup).

Macros

	Name	Description
	<u>DRV_ETHMAC_INDEX_0</u>	Ethernet driver index definitions.
	<u>DRV_ETHMAC_INDEX_1</u>	This is macro <u>DRV_ETHMAC_INDEX_1</u> .
	<u>DRV_ETHMAC_INDEX_COUNT</u>	Number of valid Ethernet driver indices.

Company

Microchip Technology Inc.

FileName: drv_ethmac.h

5.1.3.8.3 drv_ethmac_config.h

ETHMAC Driver Configuration Definitions for the template version

These definitions statically define the driver's mode of operation.

Macros

	Name	Description
	<u>DRV_ETHMAC_CLIENTS_NUMBER</u>	Selects the maximum number of clients.
	<u>DRV_ETHMAC_INDEX</u>	Ethernet MAC static index selection.

	<u>DRV_ETHMAC_INSTANCES_NUMBER</u>	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	<u>DRV_ETHMAC_INTERRUPT_MODE</u>	Controls operation of the driver in the interrupt or polled mode.
	<u>DRV_ETHMAC_INTERRUPT_SOURCE</u>	Defines an override of the interrupt source in case of static driver.
	<u>DRV_ETHMAC_PERIPHERAL_ID</u>	Defines an override of the peripheral ID.
	<u>DRV_ETHMAC_POWER_STATE</u>	Defines an override of the power state of the Ethernet MAC driver.

File Name

drv_ethmac_config.h

Company

Microchip Technology Inc.

5.1.4 Ethernet PHY Driver Library

5.1.4.1 Introduction

Ethernet PHY Driver Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Ethernet PHY Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

This library provides a software abstraction for configuring external Ethernet PHY devices for use with the on-chip Ethernet Controller.

5.1.4.2 Release Notes

MPLAB Harmony Version: v0.70b **Ethernet PHY Driver Library Version :** 7.10 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.1.4.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.4.4 Using the Library

This topic describes the basic architecture of the Ethernet PHY Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_ethphy.h](#)

The interface to the Ethernet PHY Driver Library is defined in the "[drv_ethphy.h](#)" header file. This file is included by the "[drv_ethphy.h](#)" file. Any C language source (.c) file that uses the Ethernet PHY Driver Library should include "drv.h".

Library File:

The Ethernet PHY Driver Library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the library interacts with the framework.

5.1.4.4.1 Abstraction Model

Overview

To understand how this library works you must first understand how an external Ethernet PHY interfaces with the Ethernet Controller. As shown in Figure 1, the PHY has two interfaces, one for managing the PHY, known as the Serial Management Interface (SMI), for configuring the device and a second, known as the Reduced Media Independent Interface (RMII), for transmit and receive data.

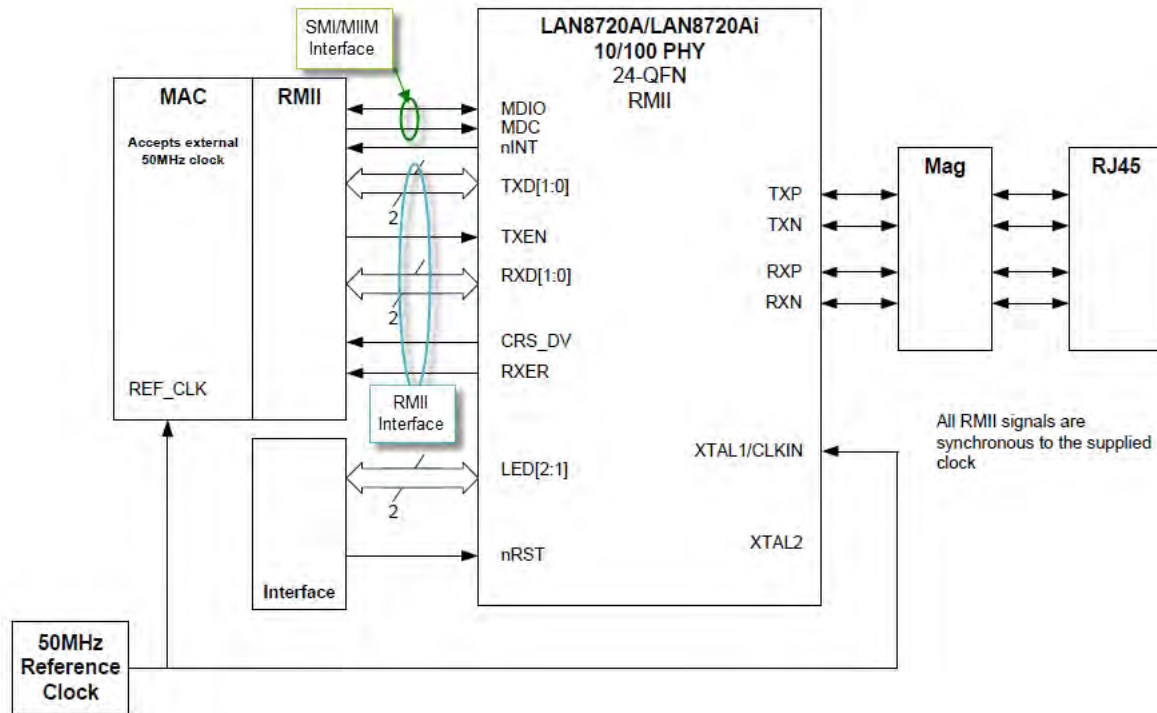


Figure 1: Typical External PHY Interface

The block diagram also shows an interrupt signal (nINT) going to a external interrupt pin on the host device and signals going to on-board LEDs to show link state and link activity.

The SMI interface is also known as the MII Management (MIIM) interface. This control interface is standardized for all PHY's by Clause 22 of the 802.3 standard. It provides up to 32 16-bit registers on the PHY. The table below provides a summary of all 32 registers. Consult the data sheet for the PHY device for the specific bit fields in each register.

Register Address	Register Name	Register Type
0	Control	Basic
1	Status	Basic
2,3	PHY Identifier	Extended
4	Auto-Negotiation Advertisement	Extended
5	Auto-Negotiation Link Partner Base Page Ability	Extended
6	Auto-Negotiation Expansion	Extended
7	Auto-Negotiation Next Page Transmit	Extended
8	Auto-Negotiation Link Partner Received Next Page	Extended

9	MASTER-SLAVE Control Register	Extended
10	MASTER-SLAVE Status Register	Extended
11-14	Reserved	Extended
15	Extended Status	Reserved
16-31	Vendor Specific	Extended

5.1.4.4.2 Library Overview

The Ethernet PHY Driver Library is divided into the following sections:

Library Interface Section	Description
System Level Functions	Routines that integrate the driver into the MPLAB Harmony system
Client Level Functions	Open, Close, Link Status, Auto Negotiation
SMI/MIIM Functions	SMI/MIIM Management Interface
External PHY Support Functions	Provides the API for PHY support routines that the driver will call when setting up the PHY. The driver library provides support for 4 PHYs.
Other Functions	Functions that provide software version information.
Data Types and Constants	C language typedefs and enums used by this library.

5.1.4.4.3 MPLAB Harmony vs. the Unified TCP/IP Stack

The following is a cross-reference of MPLAB Harmony functions and the equivalent Unified TCP/IP Stack function:

MPLAB Harmony Function:	Unified Stack Function:
DRV_ETHPHY_ClientStatus	-none-
DRV_ETHPHY_Close	-none-
DRV_ETHPHY_Deinitialize	-none-
DRV_ETHPHY_HWConfigFlagsGet	EthPhyGetHwConfigFlags
DRV_ETHPHY_Setup	EthPhyInit
DRV_ETHPHY_Initialize	-none-
DRV_ETHPHY_LinkScanRead	EthPhyScanLinkRead
DRV_ETHPHY_LinkScanStart	EthPhyScanLinkStart
DRV_ETHPHY_LinkScanStop	EthPhyScanLinkStop
DRV_ETHPHY_LinkStatusGet	EthPhyGetLinkStatus
DRV_ETHPHY_NegotiationIsComplete	EthPhyNegotiationComplete
DRV_ETHPHY_NegotiationResultGet	EthPhyGetNegotiationResult
DRV_ETHPHY_Open	-none-
DRV_ETHPHY_Reinitialize	-none-
DRV_ETHPHY_Reset	EthPhyReset
DRV_ETHPHY_RestartNegotiation	EthPhyRestartNegotiation
DRV_ETHPHY_SMIReadResultGet	EthMIIMReadResult
DRV_ETHPHY_SMIReadStart	EthMIIMReadStart
DRV_ETHPHY_SMIScanDataGet	EthMIIMScanResult
DRV_ETHPHY_SMIScanStart	EthMIIMScanStart
DRV_ETHPHY_SMIScanStatusGet	-none-
DRV_ETHPHY_SMIScanStop	-none-
DRV_ETHPHY_SMIWriteStart	EthMIIMWriteStart
DRV_ETHPHY_SMIIsBusy	-none-
DRV_ETHPHY_Status	-none-
DRV_ETHPHY_Tasks	-none-
DRV_ETHPHY_VersionGet	-none-
DRV_ETHPHY_VersionStrGet	-none-
DRV_ETHPHY_SMIClockSet	EthMIIMConfig

Unified Stack Function:	MPLAB Harmony Function:
EthMIIMConfig	DRV_ETHPHY_SMIClockSet
EthMIIMReadResult	DRV_ETHPHY_SMIReadResultGet
EthMIIMReadStart	DRV_ETHPHY_SMIReadStart
EthMIIMScanResult	DRV_ETHPHY_SMIScanDataGet
EthMIIMScanStart	DRV_ETHPHY_SMIScanStart
EthMIIMWriteStart	DRV_ETHPHY_SMIWriteStart
EthPhyGetHwConfigFlags	DRV_ETHPHY_HWConfigFlagsGet
EthPhyGetLinkStatus	DRV_ETHPHY_LinkStatusGet
EthPhyGetNegotiationResult	DRV_ETHPHY_NegotiationResultGet
EthPhyInit	DRV_ETHPHY_Setup
EthPhyNegotiationComplete	DRV_ETHPHY_NegotiationIsComplete
EthPhyReset	DRV_ETHPHY_Reset
EthPhyRestartNegotiation	DRV_ETHPHY_RestartNegotiation
EthPhyScanLinkRead	DRV_ETHPHY_LinkScanRead
EthPhyScanLinkStart	DRV_ETHPHY_LinkScanStart
EthPhyScanLinkStop	DRV_ETHPHY_LinkScanStop
-none-	DRV_ETHPHY_ClientStatus
-none-	DRV_ETHPHY_Close
-none-	DRV_ETHPHY_Deinitialize
-none-	DRV_ETHPHY_Initialize
-none-	DRV_ETHPHY_Open
-none-	DRV_ETHPHY_Reinitialize
-none-	DRV_ETHPHY_SMIScanStatusGet
-none-	DRV_ETHPHY_SMIScanStop
-none-	DRV_ETHPHY_SMIIsBusy
-none-	DRV_ETHPHY_Status
-none-	DRV_ETHPHY_Tasks
-none-	DRV_ETHPHY_VersionGet
-none-	DRV_ETHPHY_VersionStrGet

5.1.4.5 Configuring the Library

Macros

	Name	Description
	DRV_ETHPHY_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHPHY_INDEX	Ethernet PHY static index selection.
	DRV_ETHPHY_INDEX_0	Ethernet PHY driver index definitions.
	DRV_ETHPHY_INDEX_COUNT	Number of valid Ethernet PHY driver indices.
	DRV_ETHPHY_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHPHY_PERIPHERAL_ID	Defines an override of the peripheral ID.
	PHY_NEG_DONE_TMO	negotiation complete timeout, ms. based on IEEE 802.3 Clause 28 Table 28-9 autoneg_wait_timer value (max 1s)
	PHY_NEG_INIT_TMO	negotiation initiation timeout, ms.

	PHY_RESET_CLR_TMO	reset self clear timeout, ms. IEEE 802.3 Clause 22 Table 22-7 and paragraph "22.2.4.1.1 Reset" (max 0.5s)
	TCPIP_IF_PIC32INT	We have an internal PIC32 MAC

Description

The configuration of the Ethernet MAC Driver Library is based on the file sys_config.h

This header file contains the configuration selection for the Ethernet MAC Driver Library. Based on the selections made, the Ethernet MAC Driver Library will support or not support selected features. These configuration settings will apply to all instances of the Ethernet MAC Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.4.5.1 DRV_ETHPHY_CLIENTS_NUMBER Macro**C**

```
#define DRV_ETHPHY_CLIENTS_NUMBER 1
```

Description

Ethernet PHY Maximum Number of Clients

This definition select the maximum number of clients that the Ethernet PHY driver can support at run time. Not defining it means using a single client.

Remarks

None.

5.1.4.5.2 DRV_ETHPHY_INDEX Macro**C**

```
#define DRV_ETHPHY_INDEX DRV_ETHPHY_INDEX_1
```

Description

Ethernet PHY Static Index Selection

This definition selects the Ethernet PHY static index for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

5.1.4.5.3 DRV_ETHPHY_INDEX_0 Macro**C**

```
#define DRV_ETHPHY_INDEX_0 0
```

Description

Ethernet PHY Driver Module Index Numbers

These constants provide the Ethernet PHY driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the [DRV_ETHPHY_Initialize](#) and [DRV_ETHPHY_Open](#) routines to identify the driver instance in use.

5.1.4.5.4 DRV_ETHPHY_INDEX_COUNT Macro

C

```
#define DRV_ETHPHY_INDEX_COUNT 1
```

Description

Ethernet PHY Driver Module Index Count

This constant identifies the number of valid Ethernet PHY driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

5.1.4.5.5 DRV_ETHPHY_INSTANCES_NUMBER Macro

C

```
#define DRV_ETHPHY_INSTANCES_NUMBER 1
```

Description

Ethernet PHY hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver. Not defining it means using a static driver.

Remarks

None.

5.1.4.5.6 DRV_ETHPHY_PERIPHERAL_ID Macro

C

```
#define DRV_ETHPHY_PERIPHERAL_ID ETHPHY_ID_1
```

Description

Ethernet PHY Peripheral ID Selection

Defines an override of the peripheral ID, using macros.

Remarks

Some devices also support ETHPHY_ID_0

5.1.4.5.7 PHY_NEG_DONE_TMO Macro

C

```
#define PHY_NEG_DONE_TMO 2000 // negotiation complete timeout, ms.
```

Description

negotiation complete timeout, ms. based on IEEE 802.3 Clause 28 Table 28-9 autoneg_wait_timer value (max 1s)

5.1.4.5.8 PHY_NEG_INIT_TMO Macro

C

```
#define PHY_NEG_INIT_TMO 1 // negotiation initiation timeout, ms.
```

Description

negotiation initiation timeout, ms.

5.1.4.5.9 PHY_RESET_CLR_TMO Macro

C

```
#define PHY_RESET_CLR_TMO 500 // reset self clear timeout, ms.
```

Description

reset self clear timeout, ms. IEEE 802.3 Clause 22 Table 22-7 and paragraph "22.2.4.1.1 Reset" (max 0.5s)

5.1.4.5.10 TCPIP_IF_PIC32INT Macro

C

```
#define TCPIP_IF_PIC32INT
```

Description

We have an internal PIC32 MAC

5.1.4.6 Building the Library

This section list the files that are available in the \src of the Ethernet PHY Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.1.4.7 Library Interface

Data Types and Constants

	Name	Description
	DRV_ETHPHY_CLIENT_STATUS	Identifies the client-specific status of the Ethernet PHY driver.
	DRV_ETHPHY_INIT	Contains all the data necessary to initialize the Ethernet PHY device.
	SMI_SCAN_DATA_STATUS	Returns the status of the SMI of the Ethernet PHY.
	SYS_OBJ_HANDLE_INVALID	This is macro SYS_OBJ_HANDLE_INVALID.
	SYS_OBJ_HANDLE_STATIC	This is macro SYS_OBJ_HANDLE_STATIC.
	SYS_OBJ_HANDLE	SYS_MODULE_OBJ is badly named. It should be SYS_MODULE_OBJ_HANDLE or something shorter. For brevity, it is renamed to SYS_OBJ_HANDLE.

Client Level Functions

	Name	Description
	DRV_ETHPHY_ClientStatus	Gets the current client-specific status the Ethernet PHY driver.
	DRV_ETHPHY_Close	Closes an opened instance of the Ethernet PHY driver.

	DRV_ETHPHY_HWConfigFlagsGet	Returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags.
	DRV_ETHPHY_LinkScanRead	Returns current link status.
	DRV_ETHPHY_LinkScanStart	Starts an SMI scan of the link status.
	DRV_ETHPHY_LinkScanStop	Stops the SMI scan of the link status.
	DRV_ETHPHY_LinkStatusGet	Returns the current link status.
	DRV_ETHPHY_NegotiationIsComplete	Returns the results of a previously initiated Ethernet PHY negotiation.
	DRV_ETHPHY_NegotiationResultGet	Returns the link status after a completed negotiation.
	DRV_ETHPHY_Open	Opens the specified Ethernet PHY driver instance and returns a handle to it.
	DRV_ETHPHY_Reset	Immediately resets the Ethernet PHY.
	DRV_ETHPHY_RestartNegotiation	Restarts auto-negotiation of the Ethernet PHY link.
	DRV_ETHPHY_Setup	Initializes Ethernet PHY communication.

External PHY Support Functions

	Name	Description
	EXTPHY_MDIXCONFIGURE	Configures the MDIX mode for the Ethernet PHY.
	EXTPHY_MIICONFIGURE	Configures the Ethernet PHY in one of the MII/RMII operation modes.
	EXTPHY_SMIADDRESSGET	Returns the SMI/MIIM address of the Ethernet PHY.
	EXTPHY_SMICLOCKGET	Returns the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY.

Other Functions

	Name	Description
	DRV_ETHPHY_VersionGet	Gets the Ethernet PHY driver version in numerical format.
	DRV_ETHPHY_VersionStrGet	Gets the Ethernet PHY driver version in string format.

SMI/MIIM Functions

	Name	Description
	DRV_ETHPHY_SMIClockSet	Sets the SMI/MIIM interface clock.
	DRV_ETHPHY_SMIIsBusy	Returns a boolean 'true' if the SMI/MIIM interface is busy with a transaction.
	DRV_ETHPHY_SMIReadResultGet	Gets the result of the SMI/MIIM register read.
	DRV_ETHPHY_SMIReadStart	Initiates SMI/MIIM read transaction.
	DRV_ETHPHY_SMIScanDataGet	Gets the latest SMI/MIIM scan data result.
	DRV_ETHPHY_SMIScanStart	Starts the scan of the SMI/MIIM register.
	DRV_ETHPHY_SMIScanStatusGet	Gets the status of the SMI/MIIM scan data.
	DRV_ETHPHY_SMIScanStop	Stops the scan of the SMI/MIIM register.
	DRV_ETHPHY_SMIWriteStart	Initiates a SMI/MIIM write transaction.

System Level Functions

	Name	Description
	DRV_ETHPHY_Deinitialize	Deinitializes the specified instance of the Ethernet PHY driver module.
	DRV_ETHPHY_Initialize	Initializes the Ethernet PHY driver.
	DRV_ETHPHY_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings.
	DRV_ETHPHY_Status	Provides the current status of the Ethernet PHY driver module.
	DRV_ETHPHY_Tasks	Maintains the driver's state machine and implements its ISR.

Description

This section describes the Application Programming Interface (API) functions of the Ethernet PHY Driver Library.

Refer to each section for a detailed description.

5.1.4.7.1 System Level Functions

5.1.4.7.1.1 DRV_ETHPHY_Deinitialize Function

C

```
void DRV_ETHPHY_Deinitialize(  
    SYS_OBJ_HANDLE object  
)  
;
```

Description

This function deinitializes the specified instance of the Ethernet PHY driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Preconditions

The [DRV_ETHPHY_Initialize](#) function must have been called before calling this routine and a valid [SYS_OBJ_HANDLE](#) must have been returned.

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_ETHPHY_Initialize

Returns

None.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Example

```
SYS_OBJ_HANDLE    object;    // Returned from DRV_ETHPHY_Initialize  
SYS_STATUS        status;  
  
DRV_ETHPHY_Deinitialize(object);  
  
status = DRV_ETHPHY_Status(object);  
if (SYS_MODULE_DEINITIALIZED == status)  
{  
    // Check again later if you need to know  
    // when the driver is deinitialized.  
}
```

5.1.4.7.1.2 DRV_ETHPHY_Initialize Function

C

```
SYS_OBJ_HANDLE DRV_ETHPHY_Initialize(  
    const SYS_MODULE_INDEX index,  
    const SYS_MODULE_INIT * const init  
)  
;
```

Description

This function initializes the Ethernet PHY driver, making it ready for clients to open and use it.

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver. This pointer may be null if no data is required because static overrides have been provided.

Returns

If successful, returns a valid handle to a driver object. Otherwise, it returns [SYS_OBJ_HANDLE_INVALID](#). The returned object must be passed as argument to [DRV_ETHPHY_Reinitialize](#), [DRV_ETHPHY_Deinitialize](#), [DRV_ETHPHY_Tasks](#) and [DRV_ETHPHY_Status](#) routines.

Remarks

This function must be called before any other Ethernet PHY routine is called.

This function should only be called once during system initialization unless [DRV_ETHPHY_Deinitialize](#) is called to de-initialize the driver instance.

Example

```
DRV_ETHPHY_INIT    init;
SYS_OBJ_HANDLE    objectHandle;

// Populate the Ethernet PHY initialization structure
init.phyId = ETHPHY_ID_0;

// Populate the Ethernet PHY initialization structure
init.phyId = ETHPHY_ID_2;
init.sExtPHYFunctions.MyPHYMIIConfigure = &SMSC8720_MIIConfigure;
init.sExtPHYFunctions.MyPHYMDIXConfigure = &SMSC8720_MDIXConfigure;
init.sExtPHYFunctions.MyPHYSMIAddressGet = &SMSC8720_SMIAddressGet;
init.sExtPHYFunctions.MyPHYSMIClockGet = &SMSC8720_SMIClockGet;

// Do something

objectHandle = DRV_ETHPHY_Initialize(DRV_ETHPHY_INDEX_0, (SYS_MODULE_INIT*)&init);
if (SYS_OBJ_HANDLE_INVALID == objectHandle)
{
    // Handle error
}
```

5.1.4.7.1.3 DRV_ETHPHY_Reinitialize Function**C**

```
void DRV_ETHPHY_Reinitialize(
    SYS_OBJ_HANDLE object,
    const SYS_MODULE_INIT * const init
);
```

Description

This function reinitializes the driver and refreshes any associated hardware settings using the initialization data given, but it will not interrupt any ongoing operations.

Preconditions

The [DRV_ETHPHY_Initialize](#) function must have been called before calling this routine and a valid [SYS_OBJ_HANDLE](#) must have been returned.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_ETHPHY_Initialize routine
init	Pointer to the initialization data structure

Returns

None.

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

Example

```
DRV_ETHPHY_INIT    init;
SYS_OBJ_HANDLE     objectHandle;

// Populate the Ethernet PHY initialization structure
init.phyId = ETHPHY_ID_2;
init.sExtPHYFunctions.MyPHYMIIConfigure = &SMSC8720_MIIConfigure;
init.sExtPHYFunctions.MyPHYMDIXConfigure = &SMSC8720_MDIXConfigure;
init.sExtPHYFunctions.MyPHYSMIAddressGet = &SMSC8720_SMIAddressGet;
init.sExtPHYFunctions.MyPHYSMIClockGet = &SMSC8720_SMIClockGet;

DRV_ETHPHY_Reinitialize(objectHandle, (SYS_MODULE_INIT*)&init);

phyStatus = DRV_ETHPHY_Status(objectHandle);
if (SYS_STATUS_BUSY == phyStatus)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= phyStatus)
{
    // Handle error
}
```

5.1.4.7.1.4 DRV_ETHPHY_Status Function**C**

```
SYS_STATUS DRV_ETHPHY_Status(
    SYS_OBJ_HANDLE object
);
```

Description

This function provides the current status of the Ethernet PHY driver module.

Preconditions

The [DRV_ETHPHY_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_ETHPHY_Initialize

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to **accept** new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been de-initialized

This value is less than SYS_STATUS_ERROR.

The this operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Example

```
SYS_OBJ_HANDLE    object;    // Returned from DRV_ETHPHY_Initialize
SYS_STATUS        status;

status = DRV_ETHPHY_Status(object);
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

5.1.4.7.1.5 DRV_ETHPHY_Tasks Function**C**

```
void DRV_ETHPHY_Tasks(
    SYS_OBJ_HANDLE object
);
```

Description

This function is used to maintain the driver's internal state machine and implement its ISR for interrupt-driven implementations.

Preconditions

The **DRV_ETHPHY_Initialize** routine must have been called for the specified Ethernet PHY driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_ETHPHY_Initialize)

Returns

None

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (**SYS_Tasks**) or by the

appropriate raw ISR.

This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Example

```
SYS_OBJ_HANDLE    object;    // Returned from DRV_ETHPHY_Initialize

while (true)
{
    DRV_ETHPHY_Tasks (object);

    // Do other tasks
}
```

5.1.4.7.2 Client Level Functions

5.1.4.7.2.1 DRV_ETHPHY_ClientStatus Function

C

```
DRV_ETHPHY_CLIENT_STATUS DRV_ETHPHY_ClientStatus(
    DRV_HANDLE handle
);
```

Description

This function gets the client-specific status of the Ethernet PHY driver associated with the given handle.

Preconditions

The [DRV_ETHPHY_Initialize](#) routine must have been called.

[DRV_ETHPHY_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

A [DRV_ETHPHY_CLIENT_STATUS](#) value describing the current status of the driver.

Remarks

This function will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE phyHandle; // Returned from DRV_ETHPHY_Open
DRV_ETHPHY_CLIENT_STATUS phyClientStatus;

phyClientStatus = DRV_ETHPHY_ClientStatus(phyHandle);
if(DRV_ETHPHY_CLIENT_STATUS_ERROR >= phyClientStatus)
{
    // Handle the error
}
```

5.1.4.7.2.2 DRV_ETHPHY_Close Function

C

```
void DRV_ETHPHY_Close(
    DRV_HANDLE handle
);
```

Description

This function closes an opened instance of the Ethernet PHY driver, invalidating the handle.

Preconditions

The [DRV_ETHPHY_Initialize](#) routine must have been called for the specified Ethernet PHY driver instance.

[DRV_ETHPHY_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open routine

Returns

None

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling [DRV_ETHPHY_Open](#) before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the [DRV_ETHPHY_Status](#) operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Example

```
DRV_HANDLE handle; // Returned from DRV_ETHPHY_Open

DRV_ETHPHY_Close(handle);
```

5.1.4.7.2.3 DRV_ETHPHY_HWConfigFlagsGet Function**C**

```
ETHPHY_CONFIG_FLAGS DRV_ETHPHY_HWConfigFlagsGet(
    DRV_HANDLE handle
);
```

Description

This function returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags from the Device Configuration Fuse bits.

Preconditions

None.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

Ethernet PHY configuration flag, see [ETHPHY_CONFIG_FLAGS](#) enumeration for bit values.

Remarks

Please note,

- [ETHPHY_CONFIG_FLAGS](#) `DRV_ETHPHY_HWConfigFlagsGet( DRV\_HANDLE handle )`

replaces the legacy "Ethernet Controller Library" function:

- `eEthPhyCfgFlags EthPhyGetHwConfigFlags(void)`

Example

5.1.4.7.2.4 DRV_ETHPHY_LinkScanRead Function

C

```
ETH_LINK_STATUS DRV_ETHPHY_LinkScanRead(  
    DRV_HANDLE handle  
);
```

Description

This function returns the current link status.

Preconditions

[DRV_ETHPHY_LinkScanStart](#) must be called first.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

Current link status, see [ETH_LINK_STATUS](#) enumeration.

Remarks

See also [DRV_ETHPHY_LinkStatusGet](#).

Please note,

- [ETH_LINK_STATUS](#) `DRV_ETHPHY_LinkScanRead( DRV\_HANDLE handle )`

replaces the legacy "Ethernet Controller Library" function:

- `eEthLinkStat EthPhyScanLinkRead(void)`

Example

5.1.4.7.2.5 DRV_ETHPHY_LinkScanStart Function

C

```
void DRV_ETHPHY_LinkScanStart(  
    DRV_HANDLE handle  
);
```

Description

This function starts an SMI scan of the link status.

Preconditions

Communication with the Ethernet PHY was already established.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

None.

Remarks

This function starts an SMI scan of the Ethernet PHY link status register. It is a more efficient way of having access to the current link status, since the normal SMI frame read operation is slow.

Please note,

- void DRV_ETHPHY_LinkScanStart(DRV_HANDLE handle)
replaces the legacy "Ethernet Controller Library" function:
- void EthPhyScanLinkStart(void)

Example

5.1.4.7.2.6 DRV_ETHPHY_LinkScanStop Function

C

```
void DRV_ETHPHY_LinkScanStop(  
    DRV_HANDLE handle  
);
```

Description

This function stops the SMI scan of the ink status.

Preconditions

Communication with the Ethernet PHY was already established.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

None.

Remarks

Please note,

- void DRV_ETHPHY_LinkScanStop(DRV_HANDLE handle)
replaces the legacy "Ethernet Controller Library" function:
- void EthPhyScanLinkStop(void)

Example

5.1.4.7.2.7 DRV_ETHPHY_LinkStatusGet Function

C

```
ETH_LINK_STATUS DRV_ETHPHY_LinkStatusGet(  
    DRV_HANDLE handle,  
    bool refresh  
);
```

Description

This function returns the urrent link status.

Preconditions

[DRV_ETHPHY_Setup](#) should have been called.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
refresh	Boolean flag, true to specify that a double read is needed

Returns

Current link status, see [ETH_LINK_STATUS](#) enumeration.

Remarks

This function reads the Ethernet PHY to get current link status. If refresh is specified then, if the link is down a second read will be performed to return the current link status.

Please note,

- [ETH_LINK_STATUS](#) `DRV_ETHPHY_LinkStatusGet( DRV\_HANDLE handle, bool refresh )`

replaces the legacy "Ethernet Controller Library" function:

- `eEthLinkStat EthPhyGetLinkStatus(int refresh)`

Example**5.1.4.7.2.8 DRV_ETHPHY_NegotiationIsComplete Function****C**

```
ETH_RESULT_CODE DRV_ETHPHY_NegotiationIsComplete(
    DRV_HANDLE handle,
    bool waitComplete
);
```

Description

This function returns the results of a previously initiated Ethernet PHY negotiation.

Preconditions

[DRV_ETHPHY_Setup](#) (and [DRV_ETHPHY_RestartNegotiation](#)) should have been called.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
waitComplete	Boolean flag, true if wait for completion is required

Returns

`ETH_RES_OK` if negotiation is done, `ETH_RES_NEGOTIATION_INACTIVE` if no negotiation in progress, `ETH_RES_NEGOTIATION_NOT_STARTED` if negotiation not yet started yet (means tmo if waitComplete was requested), or `ETH_RES_NEGOTIATION_ACTIVE` if negotiation ongoing (means tmo if waitComplete was requested).

Remarks

See also [DRV_ETHPHY_NegotiationResultGet](#).

Please note,

- [ETH_RESULT_CODE](#) `DRV_ETHPHY_NegotiationIsComplete( DRV\_HANDLE handle, bool waitComplete )`

replaces the legacy "Ethernet Controller Library" function:

- eEthRes EthPhyNegotiationComplete(int waitComplete)

Example

5.1.4.7.2.9 DRV_ETHPHY_NegotiationResultGet Function

C

```
ETH_LINK_STATUS DRV_ETHPHY_NegotiationResultGet(
    DRV_HANDLE handle,
    ETH_OPEN_FLAGS* pFlags,
    ETH_PAUSE_TYPE* pPauseType
);
```

Description

This function returns the link status after a completed negotiation.

Preconditions

[DRV_ETHPHY_Setup](#), [DRV_ETHPHY_RestartNegotiation](#), and [DRV_ETHPHY_NegotiationIsComplete](#) should have been called.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
pFlags	Address to store the negotiation result
pPauseType	Address to store the pause type supported by the link partner

Returns

Link status after the (completed) negotiation, see [ETH_LINK_STATUS](#) enumeration.

Remarks

If no negotiation possible/active/failed, most likely the flags are invalid!

Please note,

- [ETH_LINK_STATUS](#) DRV_ETHPHY_NegotiationResultGet([DRV_HANDLE](#) handle, [ETH_OPEN_FLAGS](#)* pFlags, [ETH_PAUSE_TYPE](#)* pPauseType)

replaces the legacy "Ethernet Controller Library" function:

- eEthLinkStat EthPhyGetNegotiationResult(eEthOpenFlags* pFlags, eEthMacPauseType* pPauseType)

Example

5.1.4.7.2.10 DRV_ETHPHY_Open Function

C

```
DRV_HANDLE DRV_ETHPHY_Open(
    const SYS_MODULE_INDEX drvIndex,
    const DRV_IO_INTENT intent
);
```

Description

This function opens the specified Ethernet PHY driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Preconditions

The [DRV_ETHPHY_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is [DRV_HANDLE_INVALID](#).

Remarks

The handle returned is valid until the [DRV_ETHPHY_Close](#) routine is called.

This function will NEVER block waiting for hardware.

If the [DRV_IO_INTENT_BLOCKING](#) is requested and the driver was built appropriately to support blocking behavior, other client-level operations may block waiting on hardware until they are complete.

If [DRV_IO_INTENT_NON_BLOCKING](#) is requested the driver client can call the [DRV_ETHPHY_ClientStatus](#) operation to find out when the module is in the ready state.

If the requested intent flags are not supported, the function will return [DRV_HANDLE_INVALID](#).

Example

```
DRV_HANDLE  handle;

handle = DRV_ETHPHY_Open(DRV_ETHPHY_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

5.1.4.7.2.11 DRV_ETHPHY_Reset Function**C**

```
bool DRV_ETHPHY_Reset(
    DRV_HANDLE handle,
    bool waitComplete
);
```

Description

This function immediately resets the Ethernet PHY, optionally waiting for a reset to complete.

Preconditions

Communication with the Ethernet PHY was already established.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
waitComplete	boolean flag, if true the procedure will wait for reset to complete

Returns

True - If PHY reset procedure completed (or completion not required) False - Otherwise

Remarks

Please note,

- bool DRV_ETHPHY_Reset([DRV_HANDLE](#) handle, bool waitComplete)

replaces the legacy "Ethernet Controller Library" function:

- int EthPhyReset(int waitComplete)

Example

5.1.4.7.2.12 DRV_ETHPHY_RestartNegotiation Function

C

```
void DRV_ETHPHY_RestartNegotiation(  
    DRV_HANDLE handle  
)  
;
```

Description

This function restarts auto-negotiation of the Ethernet PHY link.

Preconditions

The Ethernet PHY should have been initialized with the proper duplex/speed mode.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

None.

Remarks

Please note,

- void DRV_ETHPHY_RestartNegotiation([DRV_HANDLE](#) handle)

replaces the legacy "Ethernet Controller Library" function:

- eEthRes EthPhyRestartNegotiation(void)

Example

5.1.4.7.2.13 DRV_ETHPHY_Setup Function

C

```
ETH_RESULT_CODE DRV_ETHPHY_Setup(  
    DRV_HANDLE handle,  
    ETH_OPEN_FLAGS oFlags,  
    ETHPHY_CONFIG_FLAGS cFlags,  
    ETH_OPEN_FLAGS * pResFlags  
)  
;
```

Description

This function initializes the Ethernet PHY communication. It tries to detect the external Ethernet PHY, to read the capabilities and find a match with the requested features. Then it programs the Ethernet PHY accordingly.

Preconditions

DRV_ETHMAC_LegacyInit should have been called to initialize the Ethernet MAC.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
oFlags	The requested open flags
cFlags	Ethernet PHY MII/RMII configuration flags
pResFlags	Address to store the initialization results

Returns

ETH_RES_OK for success; otherwise, an error code.

Remarks

Please note,

- [ETH_RESULT_CODE](#) DRV_ETHPHY_Setup([DRV_HANDLE](#) handle, [ETH_OPEN_FLAGS](#) oFlags, [ETHPHY_CONFIG_FLAGS](#) cFlags, [ETH_OPEN_FLAGS](#)* pResFlags)

replaces the legacy "Ethernet Controller Library" function:

- eEthRes EthPhyInit(eEthOpenFlags oFlags, eEthPhyCfgFlags cFlags, eEthOpenFlags* pResFlags)

Example

5.1.4.7.3 SMI/MIIM Functions

5.1.4.7.3.1 DRV_ETHPHY_SMIClockSet Function

C

```
void DRV_ETHPHY_SMIClockSet(
    DRV_HANDLE handle,
    uint32_t hostClock,
    uint32_t maxSMIClock
);
```

Description

This function sets SMI/MIIM interface clock base on host clock and maximum supported SMI/MIIM interface clock speed.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
hostClock	Host clock speed in Hz
maxSMIClock	Maximum supported SMI/MIIM clock speed in Hz

Returns

None.

Remarks

Please note,

- void DRV_ETHPHY_SMIClockSet([DRV_HANDLE](#) handle, uint16_t hostClock, uint16_t maxSMIClock)

replaces the legacy "Ethernet Controller Library" function:

- void EthMIIMConfig(unsigned int hostClock, unsigned int miimClock)

Example

5.1.4.7.3.2 DRV_ETHPHY_SMIisBusy Function

C

```
bool DRV_ETHPHY_SMIisBusy(  
    DRV_HANDLE handle  
)  
;
```

Description

This function returns a boolean 'true' if the SMI/MIIM interface is busy with a transaction.

Preconditions

MyPHYSMIClockSet() called to setup SMI/MIIM clock.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

- true - If the SMI/MIIM is busy
- false - If the SMI/MIIM is not busy

Example

5.1.4.7.3.3 DRV_ETHPHY_SMIReadResultGet Function

C

```
uint16_t DRV_ETHPHY_SMIReadResultGet(  
    DRV_HANDLE handle  
)  
;
```

Description

This function gets the result of the SMI/MIIM register read.

Preconditions

MyPHYSMIClockSet was called to set up the SMI/MIIM clock and [DRV_ETHPHY_SMIReadStart](#) was called to initiate a SMI/MIIM register read. [DRV_ETHPHY_SMIisBusy](#) should return false.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

Result of the SMI/MIIM register read previously scheduled.

Remarks

Please note,

- uint16_t DRV_ETHPHY_SMIReadResultGet([DRV_HANDLE](#) handle)
replaces the legacy "Ethernet Controller Library" function:
 - unsigned short EthMIIMReadResult (void)

Example

5.1.4.7.3.4 DRV_ETHPHY_SMIReadStart Function

C

```
void DRV_ETHPHY_SMIReadStart(  
    DRV_HANDLE handle,  
    unsigned int rIx,  
    unsigned int phyAdd  
);
```

Description

This function initiates SMI/MIIM read transaction for a given PHY address and register.

Preconditions

MyPHYSMIClockSet was called to set up the SMI/MIIM clock.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
rlx	PHY register to be accessed
phyAdd	Address of PHY to be accessed

Returns

None.

Remarks

Please note,

- void DRV_ETHPHY_SMIReadStart([DRV_HANDLE](#) handle, unsigned int rIx, unsigned int phyAdd) replaces the legacy "Ethernet Controller Library" function:
- void EthMIIMReadStart(unsigned int rIx, unsigned int phyAdd)

Example

5.1.4.7.3.5 DRV_ETHPHY_SMIScanDataGet Function

C

```
uint16_t DRV_ETHPHY_SMIScanDataGet(  
    DRV_HANDLE handle  
);
```

Description

This function gets the latest SMI/MIIM scan data result.

Preconditions

[DRV_ETHPHY_SMIScanStart\(\)](#) has been called. [DRV_ETHPHY_SMIScanStatusGet\(\)](#) should return SMI_SCAN_DATA_VALID.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

Current scan result.

Remarks

Scan data status must be SMI_SCAN_DATA_VALID.

Please note,

- uint16_t DRV_ETHPHY_SMI ScanDataGet([DRV_HANDLE](#) handle)
replaces the legacy "Ethernet Controller Library" function:
- unsigned short EthMIIMScanResult(void)

Example**5.1.4.7.3.6 DRV_ETHPHY_SMI ScanStart Function****C**

```
void DRV_ETHPHY_SMI ScanStart(  
    DRV_HANDLE handle,  
    unsigned int rIx,  
    unsigned int phyAdd  
)  
;
```

Description

This function starts the scan of the SMI/MIIM register.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
rlx	PHY register to be accessed, 0-31
phyAdd	PHY address, 0-31

Returns

None.

Remarks

Please note,

- void DRV_ETHPHY_SMI ScanStart([DRV_HANDLE](#) handle, unsigned int rIx, unsigned int phyAdd)
replaces the legacy "Ethernet Controller Library" function:
- void EthMIIMScanStart(unsigned int rIx, unsigned int phyAdd)

Example**5.1.4.7.3.7 DRV_ETHPHY_SMI ScanStatusGet Function****C**

```
SMI_SCAN_DATA_STATUS DRV_ETHPHY_SMI ScanStatusGet(  
    DRV_HANDLE handle  
)  
;
```

Description

This function gets the status of the SMI/MIIM scan data.

Preconditions

[DRV_ETHPHY_SMI ScanStart\(\)](#) has been called.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

SMI_SCAN_DATA_NOTVALID or SMI_SCAN_DATA_VALID

Remarks

None.

Example**5.1.4.7.3.8 DRV_ETHPHY_SMIScanStop Function****C**

```
void DRV_ETHPHY_SMIScanStop(  
    DRV_HANDLE handle  
)  
;
```

Description

This function stops the scan of the SMI/MIIM register.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)

Returns

None.

Remarks

Stops a scan transaction on the SMI interface.

Please note,

- void DRV_ETHPHY_SMIScanStop([DRV_HANDLE](#) handle)
replaces the legacy "Ethernet Controller Library" function:
- void EthMIIMScanStop (void)

Example**5.1.4.7.3.9 DRV_ETHPHY_SMIWriteStart Function****C**

```
void DRV_ETHPHY_SMIWriteStart(  
    DRV_HANDLE handle,  
    unsigned int rIx,  
    unsigned int phyAdd,  
    uint16_t wData  
)  
;
```

Description

This function initiates SMI/MIIM write transaction for a given PHY address and register.

Preconditions

MyPHYSMIClockSet was called to set up the SMI/MIIM clock.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
rlx	PHY register to be accessed
phyAdd	Address of PHY to be accessed
wData	Data to be written

Returns

None.

Remarks

Please note,

- void DRV_ETHPHY_SMIWriteStart([DRV_HANDLE](#) handle, unsigned int rlx, unsigned int phyAdd, uint16_t wData)
replaces the legacy "Ethernet Controller Library" function:
- void EthMIIMWriteStart(unsigned int rlx, unsigned int phyAdd, unsigned short wData)

Example

5.1.4.7.4 External PHY Support Functions

5.1.4.7.4.1 EXTPHY_MDIXCONFIGURE Type

C

```
typedef ETH_RESULT_CODE (* EXTPHY_MDIXCONFIGURE)(DRV_HANDLE handle, ETH_OPEN_FLAGS oFlags);
```

Description

Pointer To Function: [ETH_RESULT_CODE](#) EXTPHY_MDIXConfigure([DRV_HANDLE](#) handle, [ETH_OPEN_FLAGS](#) oFlags)

Configures the MDIX mode for the Ethernet PHY.

Preconditions

Communication to the PHY should have been established.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_PHY_Open)
oFlags	Requested open flags: ETH_OPEN_MDIX_AUTO , ETH_OPEN_MDIX_NORM , or ETH_OPEN_MDIX_NORM ETH_OPEN_MDIX_SWAP

Returns

[ETH_RES_OK](#) - if success; otherwise, [ETH_RES_CFG_ERR](#).

Remarks

For some external PHYs, the MDIX configuration is set by pins on the device. In this case, the function will return [ETH_RES_OK](#) if the cFlags requests the already configured interface; otherwise, [ETH_RES_CFG_ERR](#).

5.1.4.7.4.2 EXTPHY_MIICONFIGURE Type

C

```
typedef ETH_RESULT_CODE (* EXTPHY_MIICONFIGURE)(DRV_HANDLE handle, EXTPHY_CONFIG_FLAGS cFlags);
```

Description

Pointer To Function: `ETH_RESULT_CODE` `EXTPHY_MIIConfigure( DRV_HANDLE handle, ETHPHY_CONFIG_FLAGS cFlags )`
 Configures the Ethernet PHY in one of the MII/RMII operation modes.

Preconditions

Communication to the PHY should have been established.

Parameters

Parameters	Description
handle	Client's driver handle (returned from <code>DRV_PHY_Open</code>)
cFlags	Requested configuration flags: <code>ETH_PHY_CFG_RMII</code> or <code>ETH_PHY_CFG_MII</code>

Returns

`ETH_RES_OK` - if success; otherwise, `ETH_RES_CFG_ERR`.

Remarks

For some external PHYs, the data interface is set by pins on the device or fuses in the host. In this case, the function will return `ETH_RES_OK` if the cFlags requests the already configured interface; otherwise, `ETH_RES_CFG_ERR`.

5.1.4.7.4.3 EXTPHY_SMIADDRESSGET Type**C**

```
typedef unsigned int (* EXTPHY_SMIADDRESSGET)(DRV_HANDLE handle);
```

Description

Pointer To Function: `uint16_t` `EXTPHY_SMIAddressGet( DRV_HANDLE handle );`
 Returns the SMI/MIIM address of the Ethernet PHY.

Preconditions

Communication to the PHY should have been established.

Parameters

Parameters	Description
handle	Client's driver handle (returned from <code>DRV_PHY_Open</code>)

Returns

The SMI/MIIM Ethernet PHY address as an unsigned 16-bit integer.

Remarks

For some external PHYs, the MDIX configuration is set by pins on the device. In this case, the function will return `ETH_RES_OK` if the cFlags requests the already configured interface; otherwise, `ETH_RES_CFG_ERR`.

5.1.4.7.4.4 EXTPHY_SMICLOCKGET Type**C**

```
typedef unsigned int (* EXTPHY_SMICLOCKGET)(DRV_HANDLE handle);
```

Description

Pointer to Function: `uint16_t` `EXTPHY_SMIClockGet( DRV_HANDLE handle );`
 Returns the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY.

Preconditions

Communication to the PHY should have been established.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_PHY_Open)

Returns

The maximum SMI/MIIM clock speed as an unsigned 16-bit integer.

Remarks

For some external PHYs, the MDIX configuration is set by pins on the device. In this case, the function will return ETH_RES_OK if the cFlags requests the already configured interface; otherwise, ETH_RES_CFG_ERR.

5.1.4.7.5 Other Functions

5.1.4.7.5.1 DRV_ETHPHY_VersionGet Function

C

```
unsigned int DRV_ETHPHY_VersionGet(  
    const SYS_MODULE_INDEX drvIndex  
);
```

Description

This function gets the Ethernet PHY driver version. The version is encoded as major * 10000 + minor * 100 + patch. The stringed version can be obtained using [DRV_ETHPHY_VersionStrGet\(\)](#)

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for

Returns

Current driver version in numerical format.

Remarks

None.

Example

```
unsigned int version;  
  
version = DRV_ETHPHY_VersionGet( DRV_ETHPHY_INDEX_1 );  
  
if(version < 110200)  
{  
    // Do Something  
}
```

5.1.4.7.5.2 DRV_ETHPHY_VersionStrGet Function

C

```
char * DRV_ETHPHY_VersionStrGet(  
    const SYS_MODULE_INDEX drvIndex  
);
```

Description

This function gets the Ethernet PHY driver version. The version is returned as major.minor.path[type], where type is optional. The numerical version can be obtained using [DRV_ETHPHY_VersionGet\(\)](#)

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for.

Returns

Current Ethernet PHY driver version in the string format.

Remarks

None.

Example

```
char *version;  
  
version = DRV_ETHPHY_VersionStrGet( DRV_ETHPHY_INDEX_1 );  
  
printf("%s", version);
```

5.1.4.7.6 Data Types and Constants

5.1.4.7.6.1 DRV_ETHPHY_CLIENT_STATUS Enumeration

C

```
typedef enum {  
    DRV_ETHPHY_CLIENT_STATUS_ERROR,  
    DRV_ETHPHY_CLIENT_STATUS_CLOSED,  
    DRV_ETHPHY_CLIENT_STATUS_BUSY,  
    DRV_ETHPHY_CLIENT_STATUS_READY  
} DRV_ETHPHY_CLIENT_STATUS;
```

Description

Ethernet PHY Driver Client Status

This enumeration identifies the client-specific status of the Ethernet PHY driver.

Members

Members	Description
DRV_ETHPHY_CLIENT_STATUS_ERROR	Unspecified error condition
DRV_ETHPHY_CLIENT_STATUS_CLOSED	Client is not open
DRV_ETHPHY_CLIENT_STATUS_BUSY	An operation is currently in progress
DRV_ETHPHY_CLIENT_STATUS_READY	Up and running, no operations running

Remarks

None.

5.1.4.7.6.2 DRV_ETHPHY_INIT Structure

C

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    ETH_MODULE_ID ethphyId;
    EXTPHY_MIICONFIGURE MyPHYMIIConfigure;
    EXTPHY_MDIXCONFIGURE MyPHYMDIXConfigure;
    EXTPHY_SMIADDRESSGET MyPHYSMIAAddressGet;
    EXTPHY_SMICLOCKGET MyPHYSMIClockGet;
} DRV_ETHPHY_INIT;
```

Description

Ethernet PHY Device Driver Initialization Data

This structure contains all the data necessary to initialize the Ethernet PHY device.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
ETH_MODULE_ID ethphyId;	Identifies peripheral (PLIB-level) ID
EXTPHY_MIICONFIGURE MyPHYMIIConfigure;	Select MII or RMII data interface
EXTPHY_MDIXCONFIGURE MyPHYMDIXConfigure;	AutoMDIX or Manual MDIX
EXTPHY_SMIADDRESSGET MyPHYSMIAAddressGet;	Returns PHY's SMI Address
EXTPHY_SMICLOCKGET MyPHYSMIClockGet;	Returns PHY's clock speed

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the [DRV_ETHPHY_Initialize](#) routine.

5.1.4.7.6.3 SMI_SCAN_DATA_STATUS Enumeration

C

```
typedef enum {
    SMI_SCAN_DATA_NOTVALID,
    SMI_SCAN_DATA_VALID
} SMI_SCAN_DATA_STATUS;
```

Description

SMI Interface Scan Data Status Enumeration

This enumeration returns the status of the SMI of the Ethernet PHY.

Members

Members	Description
SMI_SCAN_DATA_NOTVALID	SMI interface ready for new scan/read/write
SMI_SCAN_DATA_VALID	SMI interface ready for new scan/read/write

Remarks

None.

5.1.4.7.6.4 SYS_OBJ_HANDLE_INVALID Macro**C**

```
#define SYS_OBJ_HANDLE_INVALID ( (SYS_OBJ_HANDLE) -1 )
```

Description

This is macro SYS_OBJ_HANDLE_INVALID.

5.1.4.7.6.5 SYS_OBJ_HANDLE_STATIC Macro**C**

```
#define SYS_OBJ_HANDLE_STATIC ( (SYS_OBJ_HANDLE) 0 )
```

Description

This is macro SYS_OBJ_HANDLE_STATIC.

5.1.4.7.6.6 SYS_OBJ_HANDLE Type**C**

```
typedef uintptr_t SYS_OBJ_HANDLE;
```

Description

[SYS_MODULE_OBJ](#) Rename

[SYS_MODULE_OBJ](#) is badly named. It should be SYS_MODULE_OBJ_HANDLE or something shorter. For brevity, it is renamed to SYS_OBJ_HANDLE.

Remarks

None.

5.1.4.8 Files**Files**

Name	Description
drv_ethphy.h	Ethernet ETHPHY Device Driver Interface File
drv_ethphy_config.h	Ethernet PHY driver configuration definitions template.

Description**5.1.4.8.1 drv_ethphy.h**

Ethernet ETHPHY Device Driver Interface

The Ethernet ETHPHY device driver provides a simple interface to manage an Ethernet ETHPHY peripheral using MIIM (or SMI) interface. This file defines the interface definitions and prototypes for the Ethernet ETHPHY driver.

Enumerations

	Name	Description
	DRV_ETHPHY_CLIENT_STATUS	Identifies the client-specific status of the Ethernet PHY driver.
	SMI_SCAN_DATA_STATUS	Returns the status of the SMI of the Ethernet PHY.

Functions

	Name	Description
	DRV_ETHPHY_ClientStatus	Gets the current client-specific status the Ethernet PHY driver.
	DRV_ETHPHY_Close	Closes an opened instance of the Ethernet PHY driver.
	DRV_ETHPHY_Deinitialize	Deinitializes the specified instance of the Ethernet PHY driver module.
	DRV_ETHPHY_HWConfigFlagsGet	Returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags.
	DRV_ETHPHY_Initialize	Initializes the Ethernet PHY driver.
	DRV_ETHPHY_LinkScanRead	Returns current link status.
	DRV_ETHPHY_LinkScanStart	Starts an SMI scan of the link status.
	DRV_ETHPHY_LinkScanStop	Stops the SMI scan of the link status.
	DRV_ETHPHY_LinkStatusGet	Returns the current link status.
	DRV_ETHPHY_NegotiationIsComplete	Returns the results of a previously initiated Ethernet PHY negotiation.
	DRV_ETHPHY_NegotiationResultGet	Returns the link status after a completed negotiation.
	DRV_ETHPHY_Open	Opens the specified Ethernet PHY driver instance and returns a handle to it.
	DRV_ETHPHY_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings.
	DRV_ETHPHY_Reset	Immediately resets the Ethernet PHY.
	DRV_ETHPHY_RestartNegotiation	Restarts auto-negotiation of the Ethernet PHY link.
	DRV_ETHPHY_Setup	Initializes Ethernet PHY communication.
	DRV_ETHPHY_SMIClockSet	Sets the SMI/MIIM interface clock.
	DRV_ETHPHY_SMIIsBusy	Returns a boolean 'true' if the SMI/MIIM interface is busy with a transaction.
	DRV_ETHPHY_SMIReadResultGet	Gets the result of the SMI/MIIM register read.
	DRV_ETHPHY_SMIReadStart	Initiates SMI/MIIM read transaction.
	DRV_ETHPHY_SMIScanDataGet	Gets the latest SMI/MIIM scan data result.
	DRV_ETHPHY_SMIScanStart	Starts the scan of the SMI/MIIM register.
	DRV_ETHPHY_SMIScanStatusGet	Gets the status of the SMI/MIIM scan data.
	DRV_ETHPHY_SMIScanStop	Stops the scan of the SMI/MIIM register.
	DRV_ETHPHY_SMIWriteStart	Initiates a SMI/MIIM write transaction.
	DRV_ETHPHY_Status	Provides the current status of the Ethernet PHY driver module.
	DRV_ETHPHY_Tasks	Maintains the driver's state machine and implements its ISR.
	DRV_ETHPHY_VersionGet	Gets the Ethernet PHY driver version in numerical format.
	DRV_ETHPHY_VersionStrGet	Gets the Ethernet PHY driver version in string format.

Macros

	Name	Description
	DRV_ETHPHY_INDEX_0	Ethernet PHY driver index definitions.
	DRV_ETHPHY_INDEX_1	This is macro DRV_ETHPHY_INDEX_1.
	DRV_ETHPHY_INDEX_COUNT	Number of valid Ethernet PHY driver indices.

Structures

	Name	Description
	DRV_ETHPHY_INIT	Contains all the data necessary to initialize the Ethernet PHY device.

Types

	Name	Description
	EXTPHY_MDIXCONFIGURE	Configures the MDIX mode for the Ethernet PHY.
	EXTPHY_MIICONFIGURE	Configures the Ethernet PHY in one of the MII/RMII operation modes.
	EXTPHY_SMIADDRESSGET	Returns the SMI/MIIM address of the Ethernet PHY.
	EXTPHY_SMICLOCKGET	Returns the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY.

Company

Microchip Technology Incorporated

FileName: drv_ethphy.h

5.1.4.8.2 drv_ethphy_config.h

Ethernet PHY Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

Macros

	Name	Description
	DRV_ETHPHY_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHPHY_INDEX	Ethernet PHY static index selection.
	DRV_ETHPHY_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHPHY_PERIPHERAL_ID	Defines an override of the peripheral ID.
	PHY_NEG_DONE_TMO	negotiation complete timeout, ms. based on IEEE 802.3 Clause 28 Table 28-9 autoneg_wait_timer value (max 1s)
	PHY_NEG_INIT_TMO	negotiation initiation timeout, ms.
	PHY_RESET_CLR_TMO	reset self clear timeout, ms. IEEE 802.3 Clause 22 Table 22-7 and paragraph "22.2.4.1.1 Reset" (max 0.5s)
	TCPIP_IF_PIC32INT	We have an internal PIC32 MAC

File Name

drv_ethphy_config.h

Company

Microchip Technology Inc.

5.1.4.8.3 drv_ethernet_flags.h

This file can be accessed through the following link: [drv_ethernet_flags.h](#)

5.1.5 Graphics (GFX) Driver Library

5.1.5.1 Introduction

Graphics (GFX) Driver Library

for

Microchip Microcontrollers

The Graphics (GFX) Driver Layer Library is the GFX library stack available for the Microchip family of microcontrollers.

Description

The MPLAB Harmony Graphics (GFX) Library is highly modular and is optimized for Microchip's 32-bit microcontrollers. Additionally, the library is free for Microchip customers, easy to use, and has an open documented interface for new driver support, which requires creation of only one C file.

5.1.5.2 Release Notes

MPLAB Harmony Version: v0.70b **Graphics (GFX) Driver Layer Library Version :** 4.00b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.1.5.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.5.4 Using the Library

5.1.5.4.1 Abstraction Model

The Graphics Driver Library structure consists of the following:

- Application Layer – This is the program that utilizes the Graphics Driver Library
- User Message Interface- This layer should be implemented by user to provide messages for the library
- Graphics Object Layer – This layer renders the widgets controls such as button, slider, window, and so on
- Graphics Primitives Layer – This layer implements the primitive drawing functions
- Device Display Driver – This layer is dependent on the display device being used
- Graphics Display Module – This is the display device being used

The library provides two configurations (Blocking and Non-Blocking).

For a Blocking configuration, all draw functions are blocking calls that delay the execution of program until rendering is done.

For a Non-Blocking configuration, draw functions do not wait for the drawing completion and release control to the program.

In this configuration, a draw function should be called repeatedly until the rendering of that particular draw function is complete. This allows efficient use of microcontroller CPU time since it can perform other tasks if the rendering is not yet done.

5.1.5.4.2 Library Overview

This layer informs the GFX library on which graphics controller is being used along with the LCD. It takes input only from the primitive layer. This layer contains all the hardware specific functions the primitive layer needs to render primitives onto the LCD. Each driver contained in the GFX library has a build time structure associated with it that informs the primitive layer which hardware specific functions are available. Thus when the primitive layer calls a specific rendering task it gets passed a pointer from the driver of that task. If the task does not exist the driver passes back a NULL and the primitive layer and either render the primitive without hardware support or let the application know that this feature is not available.

5.1.5.5 Configuring the Library

The configuration of the GFX driver is based on the file `drv_gfx_config.h`

This header file contains the configuration selection for the GFX Driver. Based on the selections made, the GFX Driver will support or not support selected features. These configuration settings will apply to all instances of the GFX Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

The configuration of the GFX driver layer is mainly dependent on making sure the needed driver has the core driver functions inside of it. The main rendering one being a PixelPut type function for the GFX primitive layer.

Once the GFX driver structure ([GFX_DRV_DATA](#)) is defined along with its primitive function rendering structure ([GFX_DRV_FUNCTIONS](#)) it can be used by the GFX primitive layer.

5.1.5.6 Building the Library

This section lists the files that are available in the /src folder of the GFX driver. It lists which files need to be included in the build based on either a hardware feature present on the board or a configuration option selected by the system.

5.1.5.7 Library Interface

Data Types and Constants

	Name	Description
	GFX_DRV_DATA	Structure containing the driver graphics controller data.
	GFX_DRV_FUNCTIONS	Structure containing the driver functions used by the primitive layer.
	LAYER_TYPE	types of Layers supported by the GFX Library
	PAGE_TYPE	types of pages supported by the GFX Library
	NUM_ALPHA_LEVELS	Specific to device
	ALPHA_DELTA	This is macro ALPHA_DELTA.
	GFX_CONFIG_DRIVER_COUNT	This is macro GFX_CONFIG_DRIVER_COUNT.
	GFX_ALPHA_PARAMS	This is type GFX_ALPHA_PARAMS.
	GFX_LAYER_PARAMS	This is type GFX_LAYER_PARAMS.

System Functions

	Name	Description
	Percentage2Alpha	
	DriverInterfaceInit	This is function DriverInterfaceInit.

Description

All of the drivers in the GFX library must follow certain guidelines to be used by the GFX Primitive. These guidelines are defined in `gfx_gol_display_driver.h`.

The GFX primitive calls function pointers to reference a particular GFX driver function. If the function does not exist for a particular driver (ie Alpha Blending) then the driver defines the function as NULL at buildtime. The function structure for a driver file can be seen in "Basic Rendering Routines".

5.1.5.7.1 System Functions

5.1.5.7.1.1 Percentage2Alpha Function

C

```
inline uint8_t Percentage2Alpha(
    uint8_t alphaPercentage
);
```

5.1.5.7.1.2 DriverInterfaceInit Function

C

```
inline void DriverInterfaceInit(
    DRV_HANDLE * pmphandle,
    DRV_PMP_MODE_CONFIG * config
);
```

Description

This is function DriverInterfaceInit.

5.1.5.7.2 Data Types and Constants**5.1.5.7.2.1 GFX_DRV_DATA Structure**

C

```
typedef struct {
    uint16_t orientation;
    uint16_t horizontalResolution;
    uint16_t verticalResolution;
    uint16_t dataWidth;
    uint16_t horizontalPulseWidth;
    uint16_t horizontalBackPorch;
    uint16_t horizontalFrontPorch;
    uint16_t verticalPulseWidth;
    uint16_t verticalBackPorch;
    uint16_t verticalFrontPorch;
    uint8_t logicShift;
    enum {
        LCD_TFT = 1,
        LCD_MSTN,
        LCD_CSTN
    } LCDType;
    uint8_t colorType;
    uint8_t timingControl;
    GFX_DRV_FUNCTIONS PrimitiveFunction;
    uint8_t driverBusy;
    uint8_t ready;
    GFX_COLOR color;
    volatile uint8_t activePage;
    volatile uint8_t visualPage;
} GFX_DRV_DATA;
```

Description

Structure: GFX_DRV_DATA

Each driver in order to be registered with the primitive layer needs these values. they are mainly dependent on the chosen LCD.

Members

Members	Description
uint16_t orientation;	Orientation of the display (given in degrees of 0,90,180,270)
uint16_t horizontalResolution;	Horizontal Resolution of the displayed orientation in Pixels
uint16_t verticalResolution;	Vertical resolution of the displayed orientation in pixels
uint16_t horizontalPulseWidth;	Horizontal Pulse Width of the LCD
uint16_t horizontalBackPorch;	Horizontal BackPorch of the LCD
uint16_t horizontalFrontPorch;	Horizontal FrontPorch of the LCD
uint16_t verticalPulseWidth;	Vertical Pulse width of the LCD
uint16_t verticalBackPorch;	Vertical BackPorch of the LCD
uint16_t verticalFrontPorch;	Vertical FrontPorch of the LCD
uint8_t logicShift;	Rising/Falling edge indicator of the LCD pixel clock

enum { LCD_TFT = 1, LCD_MSTN, LCD_CSTN } LCDType;	LCD type
uint8_t colorType;	color depth (18BPP, 24BPP, 8BPP, palette)
uint8_t timingControl;	flag to
GFX_DRV_FUNCTIONS PrimitiveFunction;	List of Primitive Renderring functions available by the driver
uint8_t driverBusy;	Flag to indicate driver is currently busy
uint8_t ready;	Flag to indicate driver has been initialized and recognized by the primitive layer
GFX_COLOR color;	current Color set for the display driver
volatile uint8_t activePage;	current activepage set for the display driver
volatile uint8_t visualPage;	current visualPage set for the display driver

5.1.5.7.2.2 GFX_DRV_FUNCTIONS Structure

C

```
typedef struct {
    uint16_t (* PixelsPut)(short,short,uint16_t, uint16_t);
    uint16_t (* BarFill)(short,short,short,short);
    uint16_t* (* PixelArrayPut)(uint16_t*,short,short,uint16_t, uint16_t);
    uint16_t* (* PixelArrayGet)(uint16_t*,short,short,uint16_t);
    uint16_t (* PixelPut)(short,short);
    void (* ColorSet)(GFX_COLOR);
    void (* InstanceSet)(uint8_t);
    uint16_t (* PageSet)(uint8_t, uint8_t);
    uint16_t* (* Layer)(uint8_t, GFX_LAYER_PARAMS*);
    uint16_t (* PixelGet)(short,short);
    uint16_t* (* AlphaBlendWindow)(GFX_ALPHA_PARAMS*, uint16_t, uint16_t, uint8_t);
} GFX_DRV_FUNCTIONS;
```

Description

Structure: GFX_DRV_FUNCTIONS

This structure needs to be defined in the driver file to inform the primitive layer as to what functions are available.

Members

Members	Description
uint16_t (* PixelsPut)(short,short,uint16_t, uint16_t);	This function sends pixels of an X,Y location to the LCD controller of a given count and linecount.
uint16_t (* BarFill)(short,short,short,short);	This function sends a 2D accelerated Bar command to the LCD controller
uint16_t* (* PixelArrayPut)(uint16_t*,short,short,uint16_t, uint16_t);	This function sends an array of pixels at an X,Y location to the LCD controller of a given count and linecount from a memory location.
uint16_t* (* PixelArrayGet)(uint16_t*,short,short,uint16_t);	This function gets an array of pixels at an X,Y location to the LCD controller of a given count and linecount from a memory location.
uint16_t (* PixelPut)(short,short);	This function sends a pixel of an X,Y location to the LCD controller.
void (* ColorSet)(GFX_COLOR);	This function sets the global color for the display driver
void (* InstanceSet)(uint8_t);	This function sets the global color for the display driver
uint16_t (* PageSet)(uint8_t, uint8_t);	See Primitive PageSet Defition
uint16_t* (* Layer)(uint8_t, GFX_LAYER_PARAMS*);	See Primitive Layer Definition
uint16_t (* PixelGet)(short,short);	This function gets a pixel of an X,Y location to the LCD controller.

uint16_t* (*AlphaBlendWindow)(GFX_ALPHA_PARAMS*, uint16_t, uint16_t, uint8_t);	See Primitive AlphaBlendWindow Definition

5.1.5.7.2.3 LAYER_TYPE Enumeration

C

```
typedef enum {
    PIP1 = 0,
    PIP2
} LAYER_TYPE;
```

Description

Enumeration: LAYER_TYPE

5.1.5.7.2.4 PAGE_TYPE Enumeration

C

```
typedef enum {
    ACTIVE_PAGE = 0,
    VISUAL_PAGE,
    BACKGROUND_PAGE,
    FOREGROUND_PAGE,
    DESTINATION_PAGE,
    TRANSITION_PAGE
} PAGE_TYPE;
```

Description

Enumeration: PAGE_TYPE

Members

Members	Description
TRANSITION_PAGE	Page that displays the transition

5.1.5.7.2.5 NUM_ALPHA_LEVELS Macro

C

```
#define NUM_ALPHA_LEVELS 0x20 // Specific to device
```

Description

Specific to device

5.1.5.7.2.6 ALPHA_DELTA Macro

C

```
#define ALPHA_DELTA ((NUM_ALPHA_LEVELS) << 5)
```

Description

This is macro ALPHA_DELTA.

5.1.5.7.2.7 GFX_CONFIG_DRIVER_COUNT Macro

C

```
#define GFX_CONFIG_DRIVER_COUNT 1
```

Description

This is macro GFX_CONFIG_DRIVER_COUNT.

5.1.5.7.2.8 GFX_ALPHA_PARAMS Structure**C**

```
typedef struct {  
    uint8_t foregroundPage;  
    short foregroundLeft;  
    short foregroundTop;  
    uint8_t backgroundPage;  
    short backgroundLeft;  
    short backgroundTop;  
    uint8_t destinationPage;  
    short destinationLeft;  
    short destinationTop;  
    uint8_t alpha;  
    GFX_COLOR prevAlphaColor;  
} GFX_ALPHA_PARAMS;
```

Description

This is type GFX_ALPHA_PARAMS.

5.1.5.7.2.9 GFX_LAYER_PARAMS Structure**C**

```
typedef struct {  
    uint8_t type;  
    uint8_t on;  
    uint8_t page;  
    short left;  
    short top;  
    uint16_t width;  
    uint16_t height;  
    short layerLeft;  
    short layerTop;  
} GFX_LAYER_PARAMS;
```

Description

This is type GFX_LAYER_PARAMS.

5.1.5.8 Files**Files**

Name	Description
drv_gfx_display.h	GFX Display Driver definitions
drv_gfx_config_template.h	This is file drv_gfx_config_template.h.

Description**5.1.5.8.1 drv_gfx_display.h**

GFX Display Driver definitions

This file describes the GFX Display Driver specific definitions.

Enumerations

	Name	Description
	LAYER_TYPE	types of Layers supported by the GFX Library
	PAGE_TYPE	types of pages supported by the GFX Library

Files

Name	Description
drv_gfx_lcc.h	Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal
drv_gfx_s1d13517.h	Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal
drv_gfx_ssd1926.h	Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal

Functions

	Name	Description
	DriverInterfaceInit	This is function DriverInterfaceInit.
	Percentage2Alpha	

Macros

	Name	Description
	ALPHA_DELTA	This is macro ALPHA_DELTA.
	GFX_CONFIG_DRIVER_COUNT	This is macro GFX_CONFIG_DRIVER_COUNT.
	NUM_ALPHA_LEVELS	Specific to device

Structures

	Name	Description
	GFX_ALPHA_PARAMS	This is type GFX_ALPHA_PARAMS.
	GFX_DRV_DATA	Structure containing the driver graphics controller data.
	GFX_DRV_FUNCTIONS	Structure containing the driver functions used by the primitive layer.
	GFX_LAYER_PARAMS	This is type GFX_LAYER_PARAMS.

Company

Microchip Technology Incorporated

FileName: gfx_display_driver.h

5.1.5.8.1.1 drv_gfx_lcc.h

None

Enumerations

	Name	Description
	DMA_ISR_TASK	This is type DMA_ISR_TASK.
	LCC_TASK	This is type LCC_TASK.

Functions

	Name	Description
	GFX_DRV_lcc_BrightnessSet	Sets the brightness of the display backlight.

	GFX_DRV_Icc_Close	closes an instance of the graphics controller
	GFX_DRV_Icc_Initialize	resets LCD, initializes PMP
	GFX_DRV_Icc_Layer	This is function GFX_DRV_Icc_Layer.
	GFX_DRV_Icc_Open	opens an instance of the graphics controller
	GFX_DRV_Icc_PixelArrayGet	gets an array of pixels of length count starting at *color
	GFX_DRV_Icc_PixelArrayPut	outputs an array of pixels of length count starting at *color
	GFX_DRV_Icc_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
	GFX_DRV_Icc_PixelsPut	outputs one pixel into the frame buffer at the x,y coordinate given
	GFX_DRV_Icc_SetColor	Sets the color for the driver instance
	GFX_DRV_Icc_SetInstance	Sets the instance for the driver
	GFX_DRV_Icc_SetPage	This is function GFX_DRV_Icc_SetPage.
	GFX_DRV_Icc_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer
	GFX_PRIM_SetPIPWindow	This is function GFX_PRIM_SetPIPWindow.

Macros

	Name	Description
	GFX_DRV_Icc_COMMANDQUEUESIZE	This is macro GFX_DRV_Icc_COMMANDQUEUESIZE.
	PIP_BUFFER	This is macro PIP_BUFFER.
	USE_LCC_SCROLLING	This is macro USE_LCC_SCROLLING.
	USE_PIP	This is macro USE_PIP.

Structures

	Name	Description
	GFX_DRV_Icc_COMMAND	Structure for the commands in the driver queue.

File Name

drv_gfx_licc.h

Company

Microchip Technology Incorporated

5.1.5.8.1.2 drv_gfx_s1d13517.h

None

Functions

	Name	Description
	GFX_DRV_S1D13517_AlphaBlendWindow	SEE primitive layer alphablendWindow definition
	GFX_DRV_S1D13517_BrightnessSet	Sets the brightness of the display backlight.
	GFX_DRV_S1D13517_Close	closes an instance of the graphics controller
	GFX_DRV_S1D13517_GetReg	returns graphics controller register value (byte access)
	GFX_DRV_S1D13517_Initialize	resets LCD, initializes PMP
	GFX_DRV_S1D13517_Layer	Updates a Layer depending on the layer parameters.
	GFX_DRV_S1D13517_Open	opens an instance of the graphics controller
	GFX_DRV_S1D13517_PixelArrayPut	outputs an array of pixels of length count starting at *color
	GFX_DRV_S1D13517_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
	GFX_DRV_S1D13517_PixelsPut	outputs one pixel into the frame buffer at the x,y coordinate given

	GFX_DRV_S1D13517_SetColor	Sets the color for the driver instance
	GFX_DRV_S1D13517_SetInstance	Sets the instance for the driver
	GFX_DRV_S1D13517_SetPage	Sets the page of a certain page type
	GFX_DRV_S1D13517_SetReg	updates graphics controller register value (byte access)
	GFX_DRV_S1D13517_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

Structures

	Name	Description
	GFX_DRV_S1D13517_COMMAND	Structure for the commands in the driver queue.
	LAYER_REGISTERS	This structure is used to describe layer registers.

File Name

S1D13517.h

Company

Microchip Technology Incorporated

5.1.5.8.1.3 drv_gfx_ssd1926.h

None

Functions

	Name	Description
	GFX_DRV_SSD1926_BarFill	Hardware accelerated barfill function
	GFX_DRV_SSD1926_BrightnessSet	Sets the brightness of the display backlight.
	GFX_DRV_SSD1926_Busy	Returns non-zero if LCD controller is busy (previous drawing operation is not completed).
	GFX_DRV_SSD1926_Close	closes an instance of the graphics controller
	GFX_DRV_SSD1926_GetReg	returns graphics controller register value (byte access)
	GFX_DRV_SSD1926_Initialize	resets LCD, initializes PMP
	GFX_DRV_SSD1926_Open	opens an instance of the graphics controller
	GFX_DRV_SSD1926_PixelArrayGet	gets an array of pixels of length count starting at *color
	GFX_DRV_SSD1926_PixelArrayPut	outputs an array of pixels of length count starting at *color
	GFX_DRV_SSD1926_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
	GFX_DRV_SSD1926_PixelsPut	outputs one pixel into the frame buffer at the x,y coordinate given
	GFX_DRV_SSD1926_SetColor	Sets the color for the driver instance
	GFX_DRV_SSD1926_SetInstance	Sets the instance for the driver
	GFX_DRV_SSD1926_SetReg	updates graphics controller register value (byte access)
	GFX_DRV_SSD1926_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

Structures

	Name	Description
	GFX_DRV_SSD1926_COMMAND	Structure for the commands in the driver queue.
	GFX_DRV_SSD1926_TASK	Structure for the task machine

File Name

SSD1926.h

Company

Microchip Technology Incorporated

5.1.5.8.2 drv_gfx_config_template.h

This is file drv_gfx_config_template.h.

5.1.5.9 Software Drivers

5.1.5.9.1 Low-Cost Controllerless (LCC) Graphics Driver Library

5.1.5.9.1.1 Introduction

**Low-Cost Controllerless Graphics Driver Library
for
Microchip Microcontrollers**

This library provides a low-level abstraction of the Low-Cost Controllerless Graphics Controller Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.1.5.9.1.2 Release Notes

MPLAB Harmony Version: v0.70b **Low-Cost Controllerless Graphics Driver Library Version :** 4.00b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

DMA Channel 1 is non-configurable. The LCC driver will require DMA Channel 1 and not allow configuration via Harmony system services.

5.1.5.9.1.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER

EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.5.9.1.4 Using the Library

This topic describes the basic architecture of the Low-Cost Controllerless (LCC) Graphics Driver Library and provides information and examples on how to use it.

Interface Header File: gfx_drv_lcc.h

The interface to the Low-Cost Controllerless (LCC) Graphics Driver library is defined in the "gfx_drv_lcc.h" header file. This file is included by the "gfx_drv_lcc.h" file. Any C language source (.c) file that uses the Low-Cost Controllerless (LCC) Graphics Driver Library should include "gfx_drv_lcc.h".

Library File:

The GFX Low-Cost Controller library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the Low-Cost Controllerless (LCC) Graphics Driver Library interacts with the framework.

5.1.5.9.1.4.1 Library Overview

Refer to the section Driver Overview for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Low-Cost Controllerless (LCC) Graphics Controller Driver module.

Library Interface Section	Description
Functions	This section lists the API functions available in the library.

5.1.5.9.1.4.2 Abstraction Model

This library provides the low-level abstraction of the Low-Cost Controllerless (LCC) Graphics Controller Driver module on the Microchip family of graphic display controllers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Low-Cost Controllerless (LCC) Graphics Controller Driver Abstraction Block Diagram

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.1.5.9.1.5 Configuring the Library

The configuration of the Low-Cost Controllerless (LCC) Graphics Driver Library is based on the file system_config.h.

This header file contains the configuration selection for the GFX Low-Cost Controller. Based on the selections made, the

Low-Cost Controllerless (LCC) Graphics Driver Library will support or not support selected features. These configuration settings will apply to all instances of the Low-Cost Controllerless (LCC) Graphics Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.5.9.1.6 Building the Library

This section list the files that are available in the \src of the Low-Cost Controllerless (LCC) Graphics Driver Library. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- gfx_drv_lcc.c

5.1.5.9.1.7 Library Interface

This section describes the Application Programming Interface (API) functions of the Low-Cost Controllerless (LCC) Graphics Driver Library.

Refer to each section for a detailed description.

5.1.5.9.1.7.1 Functions

Functions

	Name	Description
≡	GFX_DRV_lcc_Close	closes an instance of the graphics controller
≡	GFX_DRV_lcc_Initialize	resets LCD, initializes PMP
≡	GFX_DRV_lcc_Open	opens an instance of the graphics controller
≡	GFX_DRV_lcc_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer
≡	GFX_DRV_lcc_BrightnessSet	Sets the brightness of the display backlight.
≡	GFX_DRV_lcc_PixelArrayGet	gets an array of pixels of length count starting at *color
≡	GFX_DRV_lcc_PixelArrayPut	outputs an array of pixels of length count starting at *color
≡	GFX_DRV_lcc_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
≡	GFX_DRV_lcc_PixelsPut	outputs one pixel into the frame buffer at the x,y coordinate given
≡	GFX_DRV_lcc_SetColor	Sets the color for the driver instance
≡	GFX_DRV_lcc_SetInstance	Sets the instance for the driver
≡	GFX_DRV_lcc_Layer	This is function GFX_DRV_lcc_Layer.
≡	GFX_DRV_lcc_SetPage	This is function GFX_DRV_lcc_SetPage.
≡	GFX_PRIM_SetPIPWindow	This is function GFX_PRIM_SetPIPWindow.

Description

5.1.5.9.1.7.1.1 GFX_DRV_lcc_Close Function

C

```
uint16_t GFX_DRV_lcc_Close(
    uint8_t instance
);
```

Description

none

Returns

0 - instance closed 2 - instance doesn't exist 3 - instance already closed

5.1.5.9.1.7.1.2 GFX_DRV_lcc_Initialize Function**C**

```
uint16_t GFX_DRV_lcc_Initialize(  
    uint8_t instance  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

5.1.5.9.1.7.1.3 GFX_DRV_lcc_Open Function**C**

```
uint16_t GFX_DRV_lcc_Open(  
    uint8_t instance  
);
```

Description

none

Returns

1 - driver not initialised 2 - instance doesn't exist 3 - instance already open instance to driver when successful

5.1.5.9.1.7.1.4 GFX_DRV_lcc_Tasks Function**C**

```
void GFX_DRV_lcc_Tasks();
```

5.1.5.9.1.7.1.5 GFX_DRV_lcc_BrightnessSet Function**C**

```
void GFX_DRV_lcc_BrightnessSet(  
    uint8_t instance,  
    uint16_t level  
);
```

Description

none

Parameters

Parameters	Description

5.1.5.9.1.7.1.6 GFX_DRV_lcc_PixelArrayGet Function

C

```
uint16_t* GFX_DRV_lcc_PixelArrayGet(  
    uint16_t * color,  
    short x,  
    short y,  
    uint16_t count  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels

Returns

NULL - call not successful (lcc driver busy) !NULL - address of the display driver queue command

5.1.5.9.1.7.1.7 GFX_DRV_lcc_PixelArrayPut Function

C

```
uint16_t* GFX_DRV_lcc_PixelArrayPut(  
    uint16_t * color,  
    short x,  
    short y,  
    uint16_t count,  
    uint16_t lineCount  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels

Returns

NULL - call not successful !NULL - handle to the number of pixels remaining

5.1.5.9.1.7.1.8 GFX_DRV_lcc_PixelPut Function

C

```
uint16_t GFX_DRV_lcc_PixelPut(  
    short x,
```

```
    short y  
);
```

Description

none

Parameters

Parameters	Description
x,y	pixel coordinates

Returns

NULL - call not successful (lcc driver busy) !NULL - address of the display driver queue command

5.1.5.9.1.7.1.9 GFX_DRV_lcc_PixelsPut Function**C**

```
uint16_t GFX_DRV_lcc_PixelsPut(  
    short x,  
    short y,  
    uint16_t count,  
    uint16_t lineCount  
);
```

Description

none

Parameters

Parameters	Description
x,y	pixel coordinates

Returns

NULL - call not successful (lcc driver busy) !NULL - address of the display driver queue command

5.1.5.9.1.7.1.10 GFX_DRV_lcc_SetColor Function**C**

```
void GFX_DRV_lcc_SetColor(  
    GFX_COLOR color  
);
```

Returns

none

5.1.5.9.1.7.1.11 GFX_DRV_lcc_SetInstance Function**C**

```
void GFX_DRV_lcc_SetInstance(  
    uint8_t instance  
);
```

Returns

none

5.1.5.9.1.7.1.12 GFX_DRV_lcc_Layer Function**C**

```
uint16_t* GFX_DRV_lcc_Layer(
    uint8_t instance,
    GFX_LAYER_PARAMS* layer
);
```

Description

This is function GFX_DRV_lcc_Layer.

5.1.5.9.1.7.1.13 GFX_DRV_lcc_SetPage Function**C**

```
uint16_t GFX_DRV_lcc_SetPage(
    uint8_t pageType,
    uint8_t page
);
```

Description

This is function GFX_DRV_lcc_SetPage.

5.1.5.9.1.7.1.14 GFX_PRIM_SetPIPWindow Function**C**

```
void GFX_PRIM_SetPIPWindow(
    uint16_t left,
    uint16_t top,
    uint16_t hlength,
    uint16_t vlength,
    uint16_t pipx,
    uint16_t pipy
);
```

Description

This is function GFX_PRIM_SetPIPWindow.

5.1.5.9.1.7.2 Data Types and Constants**Enumerations**

	Name	Description
	LCC_TASK	This is type LCC_TASK.
	DMA_ISR_TASK	This is type DMA_ISR_TASK.

Macros

	Name	Description
	USE_LCC_SCROLLING	This is macro USE_LCC_SCROLLING.
	USE_PIP	This is macro USE_PIP.
	PIP_BUFFER	This is macro PIP_BUFFER.
	GFX_DRV_lcc_COMMANDQUEUESIZE	This is macro GFX_DRV_lcc_COMMANDQUEUESIZE.

Structures

	Name	Description
	GFX_DRV_lcc_COMMAND	Structure for the commands in the driver queue.

Description**5.1.5.9.1.7.2.1 USE_LCC_SCROLLING Macro****C****#define USE_LCC_SCROLLING****Description**

This is macro USE_LCC_SCROLLING.

5.1.5.9.1.7.2.2 USE_PIP Macro**C****#define USE_PIP****Description**

This is macro USE_PIP.

5.1.5.9.1.7.2.3 GFX_DRV_lcc_COMMAND Structure**C**

```
typedef struct {
    uint8_t instance;
    union {
        uint32_t Val;
        struct {
            uint8_t b0 : 1;
            uint8_t b1 : 1;
            uint8_t b2 : 1;
            uint8_t b3 : 1;
            uint8_t b4 : 1;
            uint8_t b5 : 1;
            uint8_t b6 : 1;
            uint8_t b7 : 1;
            uint8_t b8 : 1;
            uint8_t b9 : 1;
            uint8_t b10 : 1;
            uint8_t b11 : 1;
            uint8_t b12 : 1;
            uint8_t b13 : 1;
            uint8_t b14 : 1;
            uint8_t b15 : 1;
            uint8_t b16 : 1;
            uint8_t b17 : 1;
            uint8_t b18 : 1;
            uint8_t b19 : 1;
            uint8_t b20 : 1;
            uint8_t b21 : 1;
            uint8_t b22 : 1;
            uint8_t b23 : 1;
            uint8_t b24 : 1;
            uint8_t b25 : 1;
            uint8_t b26 : 1;
            uint8_t b27 : 1;
            uint8_t b28 : 1;
            uint8_t b29 : 1;
            uint8_t b30 : 1;
            uint8_t b31 : 1;
        } bits;
    }
};
```

```

    } address;
    uint16_t * array;
    uint16_t data;
    uint16_t count;
    uint16_t lineCount;
    uint8_t task;
} GFX_DRV_lcc_COMMAND;

```

Description

Structure: GFX_DRV_lcc_COMMAND

None

5.1.5.9.1.7.2.4 PIP_BUFFER Macro

C

```
#define PIP_BUFFER (3)
```

Description

This is macro PIP_BUFFER.

5.1.5.9.1.7.2.5 LCC_TASK Enumeration

C

```

typedef enum {
    UPDATE_CLOCK = 0,
    INITIALIZE,
    WAIT_TRANSMIT_PIXELS,
    FINISH_TRANSMIT_PIXELS,
    WAIT_TRANSMIT_ARRAY,
    FINISH_TRANSMIT_ARRAY,
    WAIT_RECEIVE,
    READ_RECEIVE,
    PUT_ARRAY,
    PUT_PIXELS,
    GET_PIXELS,
    COPY_PIXELS,
    PAGE,
    LAYERS
} LCC_TASK;

```

Description

This is type LCC_TASK.

5.1.5.9.1.7.2.6 GFX_DRV_lcc_COMMANDQUEUEUSIZE Macro

C

```
#define GFX_DRV_lcc_COMMANDQUEUEUSIZE 480
```

Description

This is macro GFX_DRV_lcc_COMMANDQUEUEUSIZE.

5.1.5.9.1.7.2.7 DMA_ISR_TASK Enumeration

C

```

typedef enum {
    ACTIVE_PERIOD = 0,
    BLANKING_PERIOD,
    FINISH_LINE,

```

```

OVERFLOW,
PIP,
SCROLL
} DMA_ISR_TASK;

```

Description

This is type DMA_ISR_TASK.

5.1.5.9.1.8 Files

Files

Name	Description

5.1.5.9.1.8.1 drv_gfx_lcc_config_template.h

- Module for Microchip Graphics Library
- This file contains compile time options for the Graphics Library.

- FileName: GraphicsConfig.h
- Dependencies: none
- Processor: PIC24F, PIC24H, dsPIC, PIC32
- Compiler: C30 V3.00/C32
- Company: Microchip Technology, Inc.
- *
- Software License Agreement
- *
- Copyright © 2008 Microchip Technology Inc. All rights reserved.
- Microchip licenses to you the right to use, modify, copy and distribute
- Software only when embedded on a Microchip microcontroller or digital
- signal controller, which is integrated into your product or third party
- product (pursuant to the sublicense terms in the accompanying license
- agreement).
- *
- You should refer to the license agreement accompanying this Software
- for additional information regarding your rights and obligations.
- *
- SOFTWARE AND DOCUMENTATION ARE PROVIDED ?AS IS? WITHOUT WARRANTY OF ANY
- KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
- OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
- PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
- OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
- BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
- DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,

- INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
- COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
- CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
- OR OTHER SIMILAR COSTS.

*

- Author Date Comment

*-----

- Anton Alkhimenok 10/28/2007 Initial Version
- Pradeep Budagutta 10/28/2007 Display related defines
- moved to DisplayConfig.h

5.1.5.10 Hardware Drivers

5.1.5.10.1 OTM2201A Graphics Controller Driver Library

5.1.5.10.1.1 Introduction

OTM2201A Graphics Controller Driver Library for Microchip Microcontrollers

This library provides a low-level abstraction of the OTM2201A Graphics Controller Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.1.5.10.1.2 Release Notes

MPLAB Harmony Version: v0.70b **OTM2201A Graphics Controller Driver Library Version :** 4.00b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.1.5.10.1.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.5.10.1.4 Using the Library

This topic describes the basic architecture of the OTM2201A Graphics Controller Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_gfx_otm2201a.h](#)

The interface to the OTM2201A Graphics Controller Driver library is defined in the "[drv_gfx_otm2201a.h](#)" header file. This file is included by the "[gfx_primitive.h](#)" file. Any C language source (.c) file that uses the OTM2201A Graphics Controller Driver library should include "[gfx_primitive.h](#)".

Library File:

The OTM2201A Graphics Controller Driver library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the OTM2201A Graphics Controller Driver interacts with the framework.

5.1.5.10.1.4.1 Abstraction Model

This library provides the low-level abstraction of the OTM2201A Graphics Controller Driver module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

OTM2201A Graphics Controller Driver Abstraction Block Diagram

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.1.5.10.1.4.2 Library Overview

Refer to the section Driver Overview for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the OTM2201 Graphics Controller Driver module.

Library Interface Section	Description
Functions	This section lists the API functions available in the library.

5.1.5.10.1.5 Configuring the Driver Library

The configuration of the OTM2201 Graphics Controller Driver is based on the file sys_config.h

This header file contains the configuration selection for the OTM2201 Graphics Controller Driver. Based on the selections made, the OTM2201 Graphics Controller Driver will support or not support selected features. These configuration settings will apply to all instances of the OTM2201 Graphics Controller Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.5.10.1.6 Building the Library

This section list the files that are available in the \src of the OTM2201A Graphics Controller Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- drv_gfx_otm2201a.c

5.1.5.10.1.7 Library Interface

This section describes the Application Programming Interface (API) functions of the OTM2201A Graphics Controller Driver Library.

Refer to each section for a detailed description.

5.1.5.10.1.7.1 Functions

Functions

	Name	Description
≡	GFX_DRV_OTM2201A_AddressSet	Sets the start GRAM address where pixel data to be written
≡	GFX_DRV_OTM2201A_BrightnessSet	Sets the brightness of the display backlight.
≡	GFX_DRV_OTM2201A_Busy	Returns non-zero value if LCD controller is busy (previous drawing operation is not completed).
≡	GFX_DRV_OTM2201A_Close	Closes an instance of the graphics controller
≡	GFX_DRV_OTM2201A_ColorSet	Sets the color for the driver instance
≡	GFX_DRV_OTM2201A_Initialize	Intialize OTM2201A device
≡	GFX_DRV_OTM2201A_InstanceSet	Sets the instance for the driver
≡	GFX_DRV_OTM2201A_Open	Opens an instance of the graphics controller
≡	GFX_DRV_OTM2201A_PixelArrayGet	Gets an array of pixels of length count into an array starting at *color
≡	GFX_DRV_OTM2201A_PixelArrayPut	Outputs an array of pixels of length count starting at *color
≡	GFX_DRV_OTM2201A_PixelPut	Outputs one pixel into the frame buffer at the x,y coordinate given
≡	GFX_DRV_OTM2201A_PixelsPut	Outputs count number of pixels into the frame buffer from the given x,y coordinate.
≡	GFX_DRV_OTM2201A_RegGet	Returns graphics controller register value (byte access)
≡	GFX_DRV_OTM2201A_RegSet	Updates graphics controller register value (byte access)
≡	GFX_DRV_OTM2201A_Tasks	Task machine that renders the driver calls for the graphics library it must be called peridically to output the contents of its circular buffer

Description

5.1.5.10.1.7.1.1 GFX_DRV_OTM2201A_AddressSet Function

C

```
uint16_t GFX_DRV_OTM2201A_AddressSet(  
    uint32_t address  
);
```

Description

Address consists of Lower 8 bit at Register REG_RAM_ADDR_LOW and Higher 8 bit at Register REG_RAM_ADDR_HIGH

Parameters

Parameters	Description
address	pixel address

Returns

DRV_OTM2201A_ERROR_PMP_WRITE - returns error during PMP Write, DRV_OTM2201A_ERROR_NO_ERROR - returns success without any error.

5.1.5.10.1.7.1.2 GFX_DRV_OTM2201A_BrightnessSet Function

C

```
void GFX_DRV_OTM2201A_BrightnessSet(  
    uint8_t instance,  
    uint16_t level  
);
```

Description

Sets the brightness of the display backlight.

Parameters

Parameters	Description
instance	instance of the driver
level	Brightness level. Valid values are 0 to 100. 0 = brightness level is zero or display is turned off. 100 = brightness level is maximum.

Returns

none

5.1.5.10.1.7.1.3 GFX_DRV_OTM2201A_Busy Function

C

```
uint16_t GFX_DRV_OTM2201A_Busy(  
    uint8_t instance  
);
```

Description

Returns non-zero value if LCD controller is busy (previous drawing operation is not completed).

Parameters

Parameters	Description
instance	driver instance

Returns

DRV_OTM2201A_ERROR_DEVICE_BUSY - Device is busy, DRV_OTM2201A_ERROR_NO_ERROR - Success, driver is not

busy.

5.1.5.10.1.7.1.4 GFX_DRV_OTM2201A_Close Function

C

```
uint16_t GFX_DRV_OTM2201A_Close(  
    uint8_t instance  
);
```

Description

Closes an instance if instance exists and not already closed

Parameters

Parameters	Description
instance	instance of the driver

Returns

DRV_OTM2201A_INSTANCE_CLOSED - instance closed, DRV_OTM2201A_INSTANCE_DOESNT_EXIST - instance doesn't exist, DRV_OTM2201A_INSTANCE_ALREADY_CLOSED - instance already closed.

5.1.5.10.1.7.1.5 GFX_DRV_OTM2201A_ColorSet Function

C

```
void GFX_DRV_OTM2201A_ColorSet(  
    GFX_COLOR color  
);
```

Description

Sets the color for the driver instance

Parameters

Parameters	Description
color	16 bit 565 format color value

Returns

none

5.1.5.10.1.7.1.6 GFX_DRV_OTM2201A_Initialize Function

C

```
uint16_t GFX_DRV_OTM2201A_Initialize(  
    uint8_t instance  
);
```

Description

Initialize OTM2201A device by initializing PMP interface, initializing command buffer for current instance initialize device registers

Parameters

Parameters	Description
instance	driver instance

Returns

DRV_OTM2201A_ERROR_QUEUE_FULL - OTM2201A command queue is full, DRV_OTM2201A_ERROR_REG_GET - OTM2201A Error while reading register, DRV_OTM2201A_ERROR_REG_SET - OTM2201A Error while writing register,

DRV_OTM2201A_ERROR_DEVICE_BUSY - OTM2201A device is busy, DRV_OTM2201A_ERROR_NO_ERROR - Success without any error.

5.1.5.10.1.7.1.7 GFX_DRV_OTM2201A_InstanceSet Function

C

```
void GFX_DRV_OTM2201A_InstanceSet(  
    uint8_t instance  
);
```

Description

Sets the instance for the driver

Parameters

Parameters	Description
instance	driver instance

Returns

none

5.1.5.10.1.7.1.8 GFX_DRV_OTM2201A_Open Function

C

```
uint16_t GFX_DRV_OTM2201A_Open(  
    uint8_t instance  
);
```

Description

Opens the instance of driver if instance is valid and not already opened

Parameters

Parameters	Description
instance	instance of the driver

Returns

DRV_OTM2201A_NEW_INSTANCE_OPEN - driver not initialised, DRV_OTM2201A_INSTANCE_DOESNT_EXIST - instance doesn't exist, DRV_OTM2201A_INSTANCE_ALREADY_OPEN - instance already open.

5.1.5.10.1.7.1.9 GFX_DRV_OTM2201A_PixelArrayGet Function

C

```
uint16_t* GFX_DRV_OTM2201A_PixelArrayGet(  
    uint16_t * color,  
    short x,  
    short y,  
    uint16_t count  
);
```

Description

Gets an array of pixels of length count into an array starting at *color

Parameters

Parameters	Description
color	Pointer to array where color data is to be loaded

x	pixel coordinate on x axis
y	pixel coordinate on y axis
count	count number of pixels

Returns

DRV_OTM2201A_ERROR_QUEUE_FULL - OTM2201A command queue is full, DRV_OTM2201A_ERROR_NO_ERROR - Success without any error.

5.1.5.10.1.7.1.10 GFX_DRV_OTM2201A_PixelArrayPut Function**C**

```
uint16_t* GFX_DRV_OTM2201A_PixelArrayPut(
    uint16_t * color,
    short x,
    short y,
    uint16_t count,
    uint16_t lineCount
);
```

Description

Outputs an array of pixels of length count starting at *color

Parameters

Parameters	Description
color	pointer to array of color of pixels
x	pixel coordinate on x axis.
y	pixel coordinate on y axis.
count	count number of pixels
lineCount	lineCount number of display lines

Returns

handle - handle to the number of pixels remaining, DRV_OTM2201A_ERROR_QUEUE_FULL - OTM2201A command queue is full.

5.1.5.10.1.7.1.11 GFX_DRV_OTM2201A_PixelPut Function**C**

```
uint16_t GFX_DRV_OTM2201A_PixelPut(
    short x,
    short y
);
```

Description

Outputs one pixel into the frame buffer at the x,y coordinate given

Parameters

Parameters	Description
x	pixel coordinate on x axis
y	pixel coordinate on y axis

Returns

DRV_OTM2201A_ERROR_QUEUE_FULL - OTM2201A command queue is full, DRV_OTM2201A_ERROR_NO_ERROR - Success without any error.

5.1.5.10.1.7.1.12 GFX_DRV_OTM2201A_PixelsPut Function

C

```
uint16_t GFX_DRV_OTM2201A_PixelsPut(  
    short x,  
    short y,  
    uint16_t count,  
    uint16_t lineCount  
);
```

Description

Outputs count number of pixels into the frame buffer from the given x,y coordinate.

Parameters

Parameters	Description
x	pixel coordinate on x axis
y	pixel coordinate on y axis
count	count number of pixels
lineCount	lineCount number of display lines

Returns

DRV_OTM2201A_ERROR_QUEUE_FULL - OTM2201A command queue is full, DRV_OTM2201A_ERROR_NO_ERROR - Success without any error.

5.1.5.10.1.7.1.13 GFX_DRV_OTM2201A_RegGet Function

C

```
uint8_t GFX_DRV_OTM2201A_RegGet(  
    uint16_t index,  
    uint16_t * data  
);
```

Description

Returns graphics controller register value (byte access)

Parameters

Parameters	Description
index	register number
*data	array to store register data

Returns

DRV_OTM2201A_ERROR_PMP_WRITE - returns error during PMP Write, DRV_OTM2201A_ERROR_PMP_READ - returns error during PMP Read, DRV_OTM2201A_ERROR_NO_ERROR - returns success without any error.

5.1.5.10.1.7.1.14 GFX_DRV_OTM2201A_RegSet Function

C

```
uint16_t GFX_DRV_OTM2201A_RegSet(  
    uint16_t index,  
    uint16_t value,  
    uint32_t repeatCount  
);
```

Description

This call can set "value" of the register accessed by its "index" and can repeat the same by number of times mentioned in "repeatCount"

Parameters

Parameters	Description
index	register number
value	value to write to register
repeatCount	repeatCount number of times value is to be written to the register.

Returns

DRV_OTM2201A_ERROR_PMP_WRITE - returns error during PMP Write, DRV_OTM2201A_ERROR_NO_ERROR - returns success without any error.

5.1.5.10.1.7.1.15 GFX_DRV_OTM2201A_Tasks Function**C**

```
void GFX_DRV_OTM2201A_Tasks();
```

Description

Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer.

5.1.5.10.1.7.2 Data Types and Constants**Enumerations**

	Name	Description
	OTM2201A_TASK	Enumeration for command type.

Structures

	Name	Description
	GFX_DRV_OTM2201A_COMMAND	Structure for the commands in the driver queue.

Description**5.1.5.10.1.7.2.1 GFX_DRV_OTM2201A_COMMAND Structure****C**

```
typedef struct {
    uint8_t instance;
    uint32_t address;
    uint16_t * array;
    uint16_t data;
    uint16_t count;
    uint16_t lineCount;
    OTM2201A_TASK task;
} GFX_DRV_OTM2201A_COMMAND;
```

Description

Structure: GFX_DRV_OTM2201A_COMMAND

Structure for the commands in the driver queue.

Parameters

Parameters	Description
instance	instance of the driver
address	pixel address
array	pointer to array of pixel data
data	pixel color
count	count number of pixels in one line
lineCount	lineCount number of lines of display
task	Type of task (OTM2201A_TASK enum)

5.1.5.10.1.7.2.2 OTM2201A_TASK Enumeration**C**

```
typedef enum {  
    INITIALIZE = 0,  
    BUSY,  
    PUT_ARRAY,  
    PUT_PIXELS  
} OTM2201A_TASK;
```

Description

Enum: OTM2201A_TASK

Enumeration for command type.

Parameters

Parameters	Description
INITIALIZE	driver initialization task
BUSY	driver busy task
PUT_ARRAY	driver put array task
PUT_PIXELS	driver put pixels task

5.1.5.10.1.7.3 Files**Files**

Name	Description
drv_gfx_otm2201a.h	Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal

Description**5.1.5.10.1.7.3.1 drv_gfx_otm2201a.h**

None

Enumerations

	Name	Description
	OTM2201A_TASK	Enumeration for command type.

Functions

	Name	Description
	GFX_DRV_OTM2201A_AddressSet	Sets the start GRAM address where pixel data to be written
	GFX_DRV_OTM2201A_BrightnessSet	Sets the brightness of the display backlight.
	GFX_DRV_OTM2201A_Busy	Returns non-zero value if LCD controller is busy (previous drawing operation is not completed).
	GFX_DRV_OTM2201A_Close	Closes an instance of the graphics controller
	GFX_DRV_OTM2201A_ColorSet	Sets the color for the driver instance
	GFX_DRV_OTM2201A_Initialize	Intialize OTM2201A device
	GFX_DRV_OTM2201A_InstanceSet	Sets the instance for the driver
	GFX_DRV_OTM2201A_Open	Opens an instance of the graphics controller
	GFX_DRV_OTM2201A_PixelArrayGet	Gets an array of pixels of length count into an array starting at *color
	GFX_DRV_OTM2201A_PixelArrayPut	Outputs an array of pixels of length count starting at *color
	GFX_DRV_OTM2201A_PixelPut	Outputs one pixel into the frame buffer at the x,y coordinate given
	GFX_DRV_OTM2201A_PixelsPut	Outputs count number of pixels into the frame buffer from the given x,y coordinate.
	GFX_DRV_OTM2201A_RegGet	Returns graphics controller register value (byte access)
	GFX_DRV_OTM2201A_RegSet	Updates graphics controller register value (byte access)
	GFX_DRV_OTM2201A_Tasks	Task machine that renders the driver calls for the graphics library it must be called peridically to output the contents of its circular buffer

Structures

	Name	Description
	GFX_DRV_OTM2201A_COMMAND	Structure for the commands in the driver queue.

File Name

drv_gfx_otm2201a.h

Company

Microchip Technology Incorporated

5.1.5.10.2 S1D13517 Graphics Controller Driver Library

5.1.5.10.2.1 Introduction

S1D13517 Graphics Controller Driver Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the S1D13517 Graphics Controller Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.1.5.10.2.2 Release Notes

MPLAB Harmony Version: v0.70b **S1D13517 Graphics Driver Library Version :** 4.00b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.1.5.10.2.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.5.10.2.4 Using the Library

This topic describes the basic architecture of the S1D13517 Graphics Controller Driver Library and provides information and examples on how to use it.

Interface Header File: gfx_drv_s1d13517.h

The interface to the S1D13517 Graphics Controller Driver Library is defined in the "gfx_drv_s1d13517.h" header file. This file is included by the "gfx_drv_s1d13517.h" file. Any C language source (.c) file that uses the S1D13517 Graphics Controller Driver Library should include "gfx_drv_s1d13517.h".

Library File:

The S1D13517 Graphics Controller Driver Library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the GFX S1D13517 Controller Driver interacts with the framework.

5.1.5.10.2.4.1 Abstraction Model

This library provides the low-level abstraction of the S1D13517 Graphics Controller Driver module of the Microchip family of graphic display controllers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

S1D13517 Graphics Controller Driver Abstraction Block Diagram

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.1.5.10.2.4.2 Library Overview

Refer to the section Driver Overview for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the S1D13517 Graphics Controller Driver module.

Library Interface Section	Description
Functions	This section lists the API functions available in the library.

5.1.5.10.2.5 Configuring the Library

Macros

	Name	Description
	USE_ANALOGCLOCK	Enable Analog Clock Object.

5.1.5.10.2.5.1 USE_ANALOGCLOCK Macro

C

```
#define USE_ANALOGCLOCK
```

Description

Enable Analog Clock Object.

5.1.5.10.2.5.2 USE_BITMAP_FLASH Macro

C

```
#define USE_BITMAP_FLASH
```

Description

Similar to Font data bitmaps can also be placed in two locations. One is in FLASH memory and the other is from external memory. Defining one or both enables the support for bitmaps located in internal flash and external memory. Define this in GraphicsConfig.h

- USE_BITMAP_FLASH - Images located in internal flash memory.
- USE_BITMAP_EXTERNAL - Images located in external memory (including external memory mapped to EDS)..

5.1.5.10.2.5.3 USE_BUTTON Macro

C

```
#define USE_BUTTON
```

Description

Enable Button Object.

5.1.5.10.2.5.4 USE_CHART Macro

C

```
#define USE_CHART
```

Description

This is macro USE_CHART.

5.1.5.10.2.5.5 USE_CHECKBOX Macro

C

```
#define USE_CHECKBOX
```

Description

Enable Checkbox Object.

5.1.5.10.2.5.6 USE_COMP_RLE Macro

C

```
#define USE_COMP_RLE
```

Description

- Overview: To enable support for DEFLATE compressed images for PutImage().
 - When this macro is enabled, the PutImage() function will
 - be able to process images generated by the Graphics Resource
 - Converter (GRC) that are compressed using the DEFLATE algorithm.
 - PutImage() will need the IPU module of the Microchip Graphics
 - Module to decompress. Enable this feature only when the driver
 - features the IPU module (example: PIC24FJ2456DA210).
 - Define this in GraphicsConfig.h
- *

```
*****
```

```
#define USE_COMP_IPU
```

5.1.5.10.2.5.7 USE_CUSTOM Macro

C

```
#define USE_CUSTOM
```

Description

Enable Custom Control Object (an example to create customized Object).

5.1.5.10.2.5.8 USE_DIGITALMETER Macro

C

```
#define USE_DIGITALMETER
```

Description

Enable DigitalMeter Object.

5.1.5.10.2.5.9 USE_EDITBOX Macro**C****#define USE_EDITBOX****Description**

Enable Edit Box Object.

5.1.5.10.2.5.10 USE_FONT_FLASH Macro**C****#define USE_FONT_FLASH****5.1.5.10.2.5.11 USE_GOL Macro****C****#define USE_GOL****Description**

Enable Graphics Object Layer.

5.1.5.10.2.5.12 USE_GRADIENT Macro**C****#define USE_GRADIENT****Description**

To enable support for Gradient bars and bevel primitives. Define this in GraphicsConfig.h.

5.1.5.10.2.5.13 USE_GROUPBOX Macro**C****#define USE_GROUPBOX****Description**

Enable Group Box Object.

5.1.5.10.2.5.14 USE_LISTBOX Macro**C****#define USE_LISTBOX****Description**

Enable List Box Object.

5.1.5.10.2.5.15 USE_METER Macro**C****#define USE_METER****Description**

Enable Meter Object.

5.1.5.10.2.5.16 USE_MULTIBYTECHAR Macro

C

```
#define USE_MULTIBYTECHAR
```

Description

To enable support for unicode fonts, USE_MULTIBYTECHAR must be defined. This sets the XCHAR definition (0-2¹⁶ range). See XCHAR for details. Define this in GraphicsConfig.h

5.1.5.10.2.5.17 USE_PICTURE Macro

C

```
#define USE_PICTURE
```

Description

Enable Picture Object.

5.1.5.10.2.5.18 USE_PROGRESSBAR Macro

C

```
#define USE_PROGRESSBAR
```

Description

Enable Progress Bar Object.

5.1.5.10.2.5.19 USE_RADIOBUTTON Macro

C

```
#define USE_RADIOBUTTON
```

Description

Enable Radio Button Object.

5.1.5.10.2.5.20 USE_ROUNDIAL Macro

C

```
#define USE_ROUNDIAL
```

Description

Enable Dial Object.

5.1.5.10.2.5.21 USE_SLIDER Macro

C

```
#define USE_SLIDER
```

Description

Enable Slider or Scroll Bar Object.

5.1.5.10.2.5.22 USE_STATICTEXT Macro

C

```
#define USE_STATICTEXT
```

Description

Enable Static Text Object.

5.1.5.10.2.5.23 USE_TEXTENTRY Macro**C**

```
#define USE_TEXTENTRY
```

Description

Enable TextEntry Object.

5.1.5.10.2.5.24 USE_TOUCHSCREEN Macro**C**

```
#define USE_TOUCHSCREEN
```

5.1.5.10.2.5.25 USE_WINDOW Macro**C**

```
#define USE_WINDOW
```

Description

Enable Window Object.

5.1.5.10.2.5.26 COLOR_DEPTH Macro**C**

```
#define COLOR_DEPTH 16
```

Description

Specifies the color depth used in the application defined in GraphicsConfig.h.

5.1.5.10.2.5.27 DRIVER_COUNT Macro**C**

```
#define DRIVER_COUNT 1
```

Description

- Overview: For XC16 or C30 builds only: When placing fonts in internal
- data memory, there is a 32K limit for data space. The total
- data should not exceed 32K. When this is unavoidable, the macro
- USE_GFX_FONT_IN_PROGRAM_SECTION will relocate the font in
- program space. This will remove the 32K restriction but at
- the expense of slower access.
- Define this in GraphicsConfig.h to enable the font to be
- placed in program space.

*

```
*****
```

```
#define USE_GFX_FONT_IN_PROGRAM_SECTION
```

5.1.5.10.2.5.28 GFX_free Macro**C**

```
#define GFX_free(pobj) free(pobj)           // <COPY GFX_malloc>
```

Description

When using Operating Systems (OS), define the OS specific malloc() and free() functions for compatibility with the OS based systems. Define these in GraphicsConfig.h

5.1.5.10.2.5.29 GFX_malloc Macro**C**

```
#define GFX_malloc(size) malloc(size)
```

Description

When using Operating Systems (OS), define the OS specific malloc() and free() functions for compatibility with the OS based systems. Define these in GraphicsConfig.h

5.1.5.10.2.6 Building the Library

This section list the files that are available in the \src of the S1D13517 Graphics Controller Driver Library. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- drv_gfx_s1d13517.c

5.1.5.10.2.7 Library Interface

This section describes the Application Programming Interface (API) functions of the S1D13517 Graphics Controller Driver Library.

Refer to each section for a detailed description.

5.1.5.10.2.7.1 Functions**Functions**

	Name	Description
≡	GFX_DRV_S1D13517_AlphaBlendWindow	SEE primitive layer alphablendWindow definition
≡	GFX_DRV_S1D13517_BrightnessSet	Sets the brightness of the display backlight.
≡	GFX_DRV_S1D13517_Close	closes an instance of the graphics controller
≡	GFX_DRV_S1D13517_GetReg	returns graphics controller register value (byte access)
≡	GFX_DRV_S1D13517_Initialize	resets LCD, initializes PMP
≡	GFX_DRV_S1D13517_Layer	Updates a Layer depending on the layer parameters.
≡	GFX_DRV_S1D13517_Open	opens an instance of the graphics controller
≡	GFX_DRV_S1D13517_PixelArrayPut	outputs an array of pixels of length count starting at *color
≡	GFX_DRV_S1D13517_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
≡	GFX_DRV_S1D13517_PixelsPut	outputs one pixel into the frame buffer at the x,y coordinate given
≡	GFX_DRV_S1D13517_SetColor	Sets the color for the driver instance
≡	GFX_DRV_S1D13517_SetInstance	Sets the instance for the driver
≡	GFX_DRV_S1D13517_SetPage	Sets the page of a certain page type
≡	GFX_DRV_S1D13517_SetReg	updates graphics controller register value (byte access)

	GFX_DRV_S1D13517_Tasks	Task machine that renders the driver calls for the graphics library it must be called peridically to output the contents of its circular buffer

Description

5.1.5.10.2.7.1.1 GFX_DRV_S1D13517_AlphaBlendWindow Function

C

```
uint16_t* GFX_DRV_S1D13517_AlphaBlendWindow(  
    GFX_ALPHA_PARAMS* alphaParams,  
    uint16_t width,  
    uint16_t height,  
    uint8_t alpha  
);
```

5.1.5.10.2.7.1.2 GFX_DRV_S1D13517_BrightnessSet Function

C

```
void GFX_DRV_S1D13517_BrightnessSet(  
    uint8_t instance,  
    uint16_t level  
);
```

Description

none

Parameters

Parameters	Description

5.1.5.10.2.7.1.3 GFX_DRV_S1D13517_Close Function

C

```
uint16_t GFX_DRV_S1D13517_Close(  
    uint8_t instance  
);
```

Description

none

Returns

0 - instance closed 2 - instance doesn't exist 3 - instance already closed

5.1.5.10.2.7.1.4 GFX_DRV_S1D13517_GetReg Function

C

```
uint8_t GFX_DRV_S1D13517_GetReg(  
    uint8_t index  
);
```

Description

none

Parameters

Parameters	Description
index	register number

*data	array to store data

Returns

0 - when call was passed

5.1.5.10.2.7.1.5 GFX_DRV_S1D13517_Initialize Function**C**

```
uint16_t GFX_DRV_S1D13517_Initialize(  
    uint8_t instance  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

5.1.5.10.2.7.1.6 GFX_DRV_S1D13517_Layer Function**C**

```
uint16_t* GFX_DRV_S1D13517_Layer(  
    uint8_t type,  
    GFX_LAYER_PARAMS* layer  
);
```

Description

none

Parameters

Parameters	Description
type	layer type
layer	parameters for Layer function call

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

5.1.5.10.2.7.1.7 GFX_DRV_S1D13517_Open Function**C**

```
uint16_t GFX_DRV_S1D13517_Open(  
    uint8_t instance  
);
```

Description

none

Returns

1 - driver not initialised 2 - instance doesn't exist 3 - instance already open instance to driver when successful

5.1.5.10.2.7.1.8 GFX_DRV_S1D13517_PixelArrayPut Function

C

```
uint16_t* GFX_DRV_S1D13517_PixelArrayPut(  
    uint16_t * color,  
    short x,  
    short y,  
    uint16_t count,  
    uint16_t lineCount  
);
```

Description

none

Parameters

Parameters	Description
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels

Returns

NULL - call not successful (PMP driver busy) !NULL - address to the number of pixels yet to be serviced

5.1.5.10.2.7.1.9 GFX_DRV_S1D13517_PixelPut Function

C

```
uint16_t GFX_DRV_S1D13517_PixelPut(  
    short x,  
    short y  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance
color	color to output
x,y	pixel coordinates

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

5.1.5.10.2.7.1.10 GFX_DRV_S1D13517_PixelsPut Function

C

```
uint16_t GFX_DRV_S1D13517_PixelsPut(  
    short x,  
    short y,  
    uint16_t count,  
    uint16_t lineCount  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance
color	color to output
x,y	pixel coordinates

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

5.1.5.10.2.7.1.11 GFX_DRV_S1D13517_SetColor Function**C**

```
void GFX_DRV_S1D13517_SetColor(  
    GFX_COLOR color  
);
```

Returns

none

5.1.5.10.2.7.1.12 GFX_DRV_S1D13517_SetInstance Function**C**

```
void GFX_DRV_S1D13517_SetInstance(  
    uint8_t instance  
);
```

Returns

none

5.1.5.10.2.7.1.13 GFX_DRV_S1D13517_SetPage Function**C**

```
uint16_t GFX_DRV_S1D13517_SetPage(  
    uint8_t pageType,  
    uint8_t page  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

5.1.5.10.2.7.1.14 GFX_DRV_S1D13517_SetReg Function**C**

```
uint16_t GFX_DRV_S1D13517_SetReg(
    uint8_t index,
    uint8_t value
);
```

Description

none

Parameters

Parameters	Description
index	register number
value	value to write to register

Returns

1 - call was not passed 0 - call was passed

5.1.5.10.2.7.1.15 GFX_DRV_S1D13517_Tasks Function**C**

```
void GFX_DRV_S1D13517_Tasks();
```

5.1.5.10.2.7.2 S1D13517 Data Types and Constants**Structures**

	Name	Description
	GFX_DRV_S1D13517_COMMAND	Structure for the commands in the driver queue.
	LAYER_REGISTERS	This structure is used to describe layer registers.

Description**5.1.5.10.2.7.2.1 GFX_DRV_S1D13517_COMMAND Structure****C**

```
typedef struct {
    uint8_t instance;
    short x;
    short y;
    uint16_t * array;
    uint16_t data;
    uint16_t count;
    uint16_t lineCount;
    uint8_t task;
} GFX_DRV_S1D13517_COMMAND;
```

Description

Structure: GFX_DRV_S1D13517_COMMAND

None

5.1.5.10.2.7.2.2 LAYER_REGISTERS Structure

C

```
typedef struct {
    uint8_t XStart;
    uint8_t XEnd;
    uint8_t YStart0;
    uint8_t YStart1;
    uint8_t YEnd0;
    uint8_t YEnd1;
    uint8_t StartAddress0;
    uint8_t StartAddress1;
    uint8_t StartAddress2;
} LAYER_REGISTERS;
```

Description

This structure is used to describe layer registers.

5.1.5.10.2.8 Files

Files

Name	Description

5.1.5.10.2.8.1 drv_gfx_s1d13517_config_template.h

- Module for Microchip Graphics Library
- This file contains compile time options for the Graphics Library.

- FileName: GraphicsConfig.h
- Dependencies: none
- Processor: PIC24F, PIC24H, dsPIC, PIC32
- Compiler: C30 V3.00/C32
- Company: Microchip Technology, Inc.
- *
- Software License Agreement
- *
- Copyright © 2008 Microchip Technology Inc. All rights reserved.
- Microchip licenses to you the right to use, modify, copy and distribute
- Software only when embedded on a Microchip microcontroller or digital
- signal controller, which is integrated into your product or third party
- product (pursuant to the sublicense terms in the accompanying license
- agreement).
- *
- You should refer to the license agreement accompanying this Software
- for additional information regarding your rights and obligations.
- *

- SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
- KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
- OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
- PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
- OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
- BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
- DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
- INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
- COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
- CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
- OR OTHER SIMILAR COSTS.

*

- Author Date Comment

*~-----

- Anton Alkhimenok 10/28/2007 Initial Version
- Pradeep Budagutta 10/28/2007 Display related defines
- moved to DisplayConfig.h

5.1.5.10.3 SSD1926 Graphics Controller Driver Library

5.1.5.10.3.1 Introduction

SSD1926 Graphics Controller Driver Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the SSD1926 Graphics Controller Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.1.5.10.3.2 Release Notes

MPLAB Harmony Version: v0.70b **SSD1926 Graphics Controller Driver Library Version :** 4.00b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.1.5.10.3.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.5.10.3.4 Using the Library

This topic describes the basic architecture of the SSD1926 Graphics Controller Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_gfx_ssd1926.h](#)

The interface to the SSD1926 Graphics Controller Driver Library is defined in the "[drv_gfx_ssd1926.h](#)" header file. This file is included by the "[gfx_primitive.h](#)" file. Any C language source (.c) file that uses the SSD1926 Graphics Controller Driver Library should include "[gfx_primitive.h](#)".

Library File:

The SSD1926 Graphics Controller Driver Library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the SSD1926 Graphics Controller Driver Library interacts with the framework.

5.1.5.10.3.4.1 Abstraction Model

This library provides the low-level abstraction of the SSD1926 Graphics Controller Driver module of the Microchip family of graphic display controllers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

SSD1926 Graphics Controller Driver Abstraction Block Diagram

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.1.5.10.3.4.2 Library Overview

Refer to the section Driver Overview for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the SSD1926 Graphics Controller Driver module.

Library Interface Section	Description
Functions	This section lists the API functions available in the library.

5.1.5.10.3.5 Configuring the Library

The configuration of the SSD1926 Graphics Controller Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the SSD1926 Graphics Controller Driver. Based on the selections made, the SSD1926 Graphics Controller Driver will support or not support selected features. These configuration settings will apply to all instances of the SSD1926 Graphics Controller Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.5.10.3.6 Building the Library

This section list the files that are available in the `\src` of the SSD1926 Graphics Controller Driver Library. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- `drv_gfx_ssd1926.c`

5.1.5.10.3.7 Library Interface

This section describes the Application Programming Interface (API) functions of the SSD1926 Graphics Controller Driver Library.

Refer to each section for a detailed description.

5.1.5.10.3.7.1 Functions

Functions

	Name	Description
≡	GFX_DRV_SSD1926_BarFill	Hardware accelerated barfill function
≡	GFX_DRV_SSD1926_BrightnessSet	Sets the brightness of the display backlight.
≡	GFX_DRV_SSD1926_Busy	Returns non-zero if LCD controller is busy (previous drawing operation is not completed).
≡	GFX_DRV_SSD1926_Close	closes an instance of the graphics controller
≡	GFX_DRV_SSD1926_GetReg	returns graphics controller register value (byte access)
≡	GFX_DRV_SSD1926_Initialize	resets LCD, initializes PMP
≡	GFX_DRV_SSD1926_Open	opens an instance of the graphics controller
≡	GFX_DRV_SSD1926_PixelArrayGet	gets an array of pixels of length count starting at *color
≡	GFX_DRV_SSD1926_PixelArrayPut	outputs an array of pixels of length count starting at *color
≡	GFX_DRV_SSD1926_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
≡	GFX_DRV_SSD1926_PixelsPut	outputs one pixel into the frame buffer at the x,y coordinate given
≡	GFX_DRV_SSD1926_SetColor	Sets the color for the driver instance
≡	GFX_DRV_SSD1926_SetInstance	Sets the instance for the driver
≡	GFX_DRV_SSD1926_SetReg	updates graphics controller register value (byte access)
≡	GFX_DRV_SSD1926_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

Description

5.1.5.10.3.7.1.1 GFX_DRV_SSD1926_BarFill Function

```
C
uint16_t GFX_DRV_SSD1926_BarFill(
    short left,
    short top,
    short right,
    short bottom
);
```

Description

see primitive BarFill

Returns

1 - call not successful (PMP driver busy) 0 - call successful

5.1.5.10.3.7.1.2 GFX_DRV_SSD1926_BrightnessSet Function

```
C
void GFX_DRV_SSD1926_BrightnessSet(
    uint8_t instance,
    uint16_t level
);
```

Description

none

Parameters

Parameters	Description

5.1.5.10.3.7.1.3 GFX_DRV_SSD1926_Busy Function

```
C
uint16_t GFX_DRV_SSD1926_Busy(
    uint8_t instance
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance

Returns

1 - busy 0 - not busy

5.1.5.10.3.7.1.4 GFX_DRV_SSD1926_Close Function

```
C
uint16_t GFX_DRV_SSD1926_Close(
    uint8_t instance
);
```



```
);
```

Description

none

Returns

0 - instance closed 2 - instance doesn't exist 3 - instance already closed

5.1.5.10.3.7.1.5 GFX_DRV_SSD1926_GetReg Function**C**

```
uint8_t GFX_DRV_SSD1926_GetReg(  
    uint16_t index,  
    uint8_t * data  
);
```

Description

none

Parameters

Parameters	Description
index	register number
*data	array to store data

Returns

0 - when call was passed

5.1.5.10.3.7.1.6 GFX_DRV_SSD1926_Initialize Function**C**

```
uint16_t GFX_DRV_SSD1926_Initialize(  
    uint8_t instance  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance

Returns

1 - call not successful (PMP driver busy) 0 - call successful

5.1.5.10.3.7.1.7 GFX_DRV_SSD1926_Open Function**C**

```
uint16_t GFX_DRV_SSD1926_Open(  
    uint8_t instance  
);
```

Description

none

Returns

1 - driver not initialised 2 - instance doesn't exist 3 - instance already open instance to driver when successful

5.1.5.10.3.7.1.8 GFX_DRV_SSD1926_PixelArrayGet Function**C**

```
uint16_t* GFX_DRV_SSD1926_PixelArrayGet(  
    uint16_t * color,  
    short x,  
    short y,  
    uint16_t count  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels

Returns

NULL - call not successful !NULL - address of the display driver queue command

5.1.5.10.3.7.1.9 GFX_DRV_SSD1926_PixelArrayPut Function**C**

```
uint16_t* GFX_DRV_SSD1926_PixelArrayPut(  
    uint16_t * color,  
    short x,  
    short y,  
    uint16_t count,  
    uint16_t lineCount  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels

Returns

NULL - call not successful !NULL - handle to the number of pixels remaining

5.1.5.10.3.7.1.10 GFX_DRV_SSD1926_PixelPut Function

C

```
uint16_t GFX_DRV_SSD1926_PixelPut(  
    short x,  
    short y  
);
```

Description

none

Parameters

Parameters	Description
instance	driver instance
color	color to output
x,y	pixel coordinates

Returns

NULL - call not successful !NULL - address of the display driver queue command

5.1.5.10.3.7.1.11 GFX_DRV_SSD1926_PixelsPut Function

C

```
uint16_t GFX_DRV_SSD1926_PixelsPut(  
    short x,  
    short y,  
    uint16_t count,  
    uint16_t lineCount  
);
```

Description

none

Parameters

Parameters	Description
x,y	pixel coordinates

5.1.5.10.3.7.1.12 GFX_DRV_SSD1926_SetColor Function

C

```
void GFX_DRV_SSD1926_SetColor(  
    GFX_COLOR color  
);
```

Returns

none

5.1.5.10.3.7.1.13 GFX_DRV_SSD1926_SetInstance Function

C

```
void GFX_DRV_SSD1926_SetInstance(  
    uint8_t instance  
);
```

Returns

none

5.1.5.10.3.7.1.14 GFX_DRV_SSD1926_SetReg Function**C**

```
uint16_t GFX_DRV_SSD1926_SetReg(
    uint16_t index,
    uint8_t value
);
```

Description

none

Parameters

Parameters	Description
index	register number
value	value to write to register

Returns

1 - call was not passed 0 - call was passed

5.1.5.10.3.7.1.15 GFX_DRV_SSD1926_Tasks Function**C**

```
void GFX_DRV_SSD1926_Tasks();
```

5.1.5.10.3.7.2 Data Types and Constants**Structures**

	Name	Description
	GFX_DRV_SSD1926_COMMAND	Structure for the commands in the driver queue.
	GFX_DRV_SSD1926_TASK	Structure for the task machine

Description**5.1.5.10.3.7.2.1 GFX_DRV_SSD1926_COMMAND Structure****C**

```
typedef struct {
    uint8_t instance;
    uint32_t address;
    uint16_t * array;
    uint16_t data;
    uint16_t count;
    uint16_t lineCount;
    SSD1926_TASK task;
} GFX_DRV_SSD1926_COMMAND;
```

Description

Structure: GFX_DRV_SSD1926_COMMAND

None

Members

Members	Description
uint32_t address;	wether the task is complete or not

5.1.5.10.3.7.2.2 GFX_DRV_SSD1926_TASK Structure**C**

```
typedef struct {
    uint8_t instance;
    uint32_t address;
    uint16_t color;
    uint8_t state;
} GFX_DRV_SSD1926_TASK;
```

Description

Structure: GFX_DRV_SSD1926_TASK

None

5.1.5.10.3.8 Files**Files**

Name	Description

5.1.5.10.3.8.1 drv_gfx_ssd1926_config_template.h

- Module for Microchip Graphics Library
- This file contains compile time options for the Graphics Library.

- FileName: GraphicsConfig.h
- Dependencies: none
- Processor: PIC24F, PIC24H, dsPIC, PIC32
- Compiler: C30 V3.00/C32
- Company: Microchip Technology, Inc.
- *
- Software License Agreement
- *
- Copyright © 2008 Microchip Technology Inc. All rights reserved.
- Microchip licenses to you the right to use, modify, copy and distribute
- Software only when embedded on a Microchip microcontroller or digital
- signal controller, which is integrated into your product or third party
- product (pursuant to the sublicense terms in the accompanying license
- agreement).
- *
- You should refer to the license agreement accompanying this Software
- for additional information regarding your rights and obligations.

*

- SOFTWARE AND DOCUMENTATION ARE PROVIDED ?AS IS? WITHOUT WARRANTY OF ANY
- KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
- OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
- PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
- OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
- BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
- DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
- INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
- COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
- CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
- OR OTHER SIMILAR COSTS.

*

- Author Date Comment

*~-----

- Anton Alkhimenok 10/28/2007 Initial Version
- Pradeep Budagutta 10/28/2007 Display related defines
- moved to DisplayConfig.h

5.1.6 Non-Volatile Memory (NVM) Driver Library

5.1.6.1 Introduction

Non-Volatile Memory (NVM) Driver Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the NVM Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The functions provided in the NVM Driver Library also perform certain mandatory assembly language instructions specific for access and modification of memory, which are thus abstracted.

Memory Devices for PIC Microcontrollers

There are two primary forms of on-chip memory available for PIC microcontrollers: programmable Flash memory and data EEPROM memory. The access mechanism for both of these types are varied and different techniques are specified for the same.:

Flash Program Memory

The Flash program memory is readable, writeable, and erasable during normal operation over the entire operating voltage range.

A read from program memory is executed at one byte/word at a time depending on the width of the data bus.

A write to the program memory is executed in either blocks of specific sizes or a single word depending on the type of processor used.

An erase is performed in blocks. A bulk erase may be performed from user code depending on the type of processor supporting the operation.

Writing or erasing program memory will cease instruction fetches until the operation is complete, restricting memory access, and therefore preventing code execution. This is controlled by an internal programming timer.

There are three processor dependant methods for program memory modification.

- Run-Time Self Programming (RTSP)
- In-Circuit Serial Programming (ICSP)
- EJTAG programming

This section describes the RTSP techniques.

5.1.6.2 Release Notes

MPLAB Harmony Version: v0.70b **NVM Driver Library Version :** 0.01 **Release Date:** 18Nov2013

New This Release:

- Internal handling of erase feature for sector write function

Known Issues:

- Not tested by mapping the image to different parts of the memory

5.1.6.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED AS IS MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.6.4 Using the Library

This topic describes the basic architecture of the NVM Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_nvm.h](#)

The interface to the NVM Driver Library is defined in the "[drv_nvm.h](#)" header file. Any C language source (.c) file that uses the NVM Driver library should include "drv.h".

Library File: The NVM Driver Library archive (.a) file installed with MPLAB Harmony.

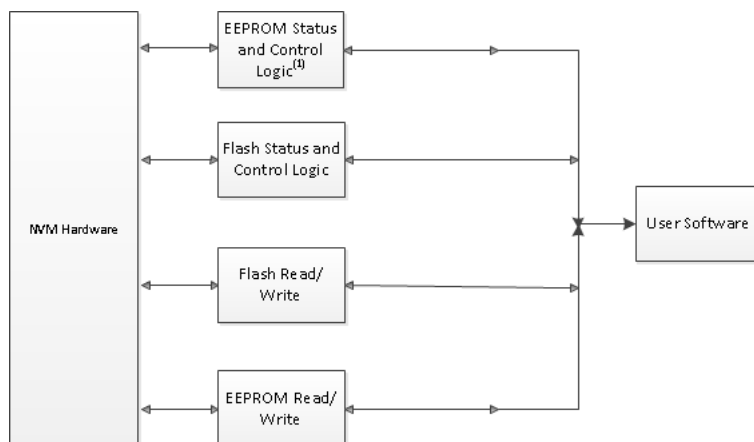
Please refer to the [MPLAB Harmony Overview](#) for how the driver interacts with the framework.

5.1.6.4.1 Abstraction Model

This library provides a low-level abstraction of the NVM module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

NVM Software Abstraction Block Diagram



Note 1: This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Abstraction Model

The NVM peripheral library provides interface routines to interact with the blocks shown in the diagram.

The **Flash Status and Control Logic** ensures that the Flash memory is configured appropriately for modification. It also provides the status of the different operations in progress as well as errors, if any. It also decides if the operation is for Flash program memory or the special device configuration registers.

The **Flash Read/Write** block ensures that the user data is written to/read from the program memory holding latches. It provides a layer of abstraction over the sequence of processor-specific instructions that are required to access this data. Depending on the processor type either block or Word programming options area available.

5.1.6.4.2 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

The library interface routines are divided into various subsections, each of the sub-sections addresses one of the blocks or the

overall operation of the NVM module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Core Functions	Provides open, close, status and other setup functions.
Client Block Transfer Functions	Provides buffered data operation functions available in the core configuration.
Miscellaneous Functions	Provides driver miscellaneous functions related to versions and others.

5.1.6.4.3 How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the modes supported for your device.

5.1.6.4.3.1 System Initialization

This section provides information for system initialization and reinitialization.

Description

System Initialization and Reinitialization

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the NVM module would be initialized with the following configuration settings (either passed dynamically at run time using [DRV_NVM_INIT](#) or by using initialization overrides) that are supported by the specific NVM device hardware:

- Device requested power state: One of the system module power states. For specific details please refer to the Data Types and Constants section.
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., NVM_ID_0)
- Defining the respective interrupt sources for transmit (TX), receive (RX), and error interrupts

The [DRV_NVM_Initialize](#) function returns an object handle of the type [SYS_MODULE_OBJ](#). After this, the object handle returned by the initialize interface would be used by the other system interfaces, such as [DRV_NVM_Reinitialize](#), [DRV_NVM_Deinitialize](#), [DRV_NVM_Status](#), and [DRV_NVM_Tasks](#).

Note: The system initialization and the reinitialization settings, only effect the instance of the peripheral that is being initialized or reinitialized.

The [SYS_MODULE_INDEX](#) is passed to the [DRV_NVM_Initialize](#) function to determine which type of memory is selected using:

- [DRV_NVM_INDEX_0](#) - FLASH

Example:

```
// This code snippet shows an example  
// of initializing the NVM Driver.
```

```
DRV_NVM_INIT    NVMInitData;  
SYS_MODULE_OBJ  objectHandle;
```

```

NVMInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
NVMInitData.moduleId             = NVM_ID_0;
NVMInitData.flashInterruptSource = INT_SOURCE_FLASH_CONTROL;

//usage of DRV_NVM_INDEX_0 indicates usage of flash realted APIs
objectHandle = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT*)NVMInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}

```

Tasks Routine

The system will call [DRV_NVM_Tasks](#), from system task service (in a polled environment) or DRV_NVM_Task will be called from the Interrupt Service Routine (ISR) of the NVM.

5.1.6.4.3.2 Client Access Operation

This section provides information for general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the [DRV_NVM_Open](#) function. This provides the configuration required to open the NVM instance for operation. If the driver is deinitialized using the function [DRV_NVM_Deinitialize](#), the application must call the [DRV_NVM_Open](#) function again to set up the instance of the NVM. The function [DRV_NVM_Open](#) need not be called again if the system is reinitialized using the DRV_NVM_Reinitialize function.

For the various options available for I/O INTENT please refer to the Data Types and Constants section.

Example:

```

DRV_HANDLE handle;

handle = DRV_NVM_Open(DRV_NVM_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}

```

5.1.6.4.3.3 Client Basic Operation

This section provides information for client basic functionality.

Description

An extremely basic interface for the driver operation is provided through client basic functionality.

The NVM driver provides the word and row programming operations in the basic mode of operation. The type of memory on which these operations will be performed is provided as an index while initialization.

The [SYS_MODULE_INDEX](#) is passed to the [DRV_NVM_Initialize](#) function to determine which type of memory is selected using:

- [DRV_NVM_INDEX_0](#) - FLASH

Note: Not all devices support the word/row programming operations. It is the responsibility of the user to determine which devices support which mode of operation. The memory access method described in this document only refers to Real-Time Self-Programming (RTSP).

Word Programming Operations

Applications using the NVM word write functionality need to perform the following:

1. The system should have completed necessary initialization and [DRV_NVM_Tasks](#) should either be running in a polled environment, or in an interrupt environment.
2. Open the driver using [DRV_NVM_Open](#) with the necessary intent.
3. Provide the desired mode of operation using the operation mode.
4. Write a word using [DRV_NVM_WriteWord](#).
5. The client will be able to close itself using the [DRV_NVM_Close](#) when required.

```

DRV_HANDLE      handle;    // Returned from DRV_NVM_Open
char            myBuffer[MY_BUFFER_SIZE];
char*           nvmAddr = BASE_ADDRESS_OF_NVM_TO_WRITE_TO
DRV_NVM_OPERATION_STATUS myStatus = 0;

myStatus = DRV_NVM_WriteWord( handle, myBuffer, nvmAddr);

if(myStatus == DRV_NVM_STATUS_OPERATION_COMPLETE)
{
    //Write successful
}

```

Applications using the NVM word erase functionality need to perform the following:

1. The system should have completed necessary initialization and [DRV_NVM_Tasks](#) should either be running in a polled environment, or in an interrupt environment.
2. Open the driver using [DRV_NVM_Open](#) with the necessary intent.
3. Provide the desired mode of operation using the operation mode.
4. Erase a word using [DRV_NVM_EraseWord](#).
5. The client will be able to close itself using the [DRV_NVM_Close](#) when required.

```

DRV_HANDLE      handle;    // Returned from DRV_NVM_Open
char*           nvmAddr = BASE_ADDRESS_OF_NVM_TO_ERASE_FROM
DRV_NVM_OPERATION_STATUS myStatus = 0;

myStatus = DRV_NVM_EraseWord( handle, myBuffer, nvmAddr);

if(myStatus == DRV_NVM_STATUS_OPERATION_COMPLETE)
{
    //Erase successful
}

```

Row Programming Operations

Applications using the NVM row erase functionality needs to perform the following:

1. The system should have completed necessary initialization and [DRV_NVM_Tasks](#) should either be running in a polled environment, or in an interrupt environment.
2. Open the driver using [DRV_NVM_Open](#) with the necessary intent.
3. Provide the desired mode of operation using the operation mode.
4. Erase a word using [DRV_NVM_EraseROW](#).
5. The client will be able to close itself using the [DRV_NVM_Close](#) when required.

```

DRV_HANDLE      handle;    // Returned from DRV_NVM_Open
char*           nvmAddr = BASE_ADDRESS_OF_NVM_TO_ERASE_FROM
DRV_NVM_OPERATION_STATUS myStatus = 0;

myStatus = DRV_NVM_EraseROW( handle, nvmAddr);

if(myStatus == DRV_NVM_STATUS_OPERATION_COMPLETE)
{
    //Erase successful
}

```

```
}
```

5.1.6.4.3.4 Client Block Data Operation

Client Block Data functionality provides advanced interface for the driver operation.

These block data functions depend heavily on the block sizes and boundaries of individual devices and the APIs are responsible for keeping that information transparent from the application. These operations return the number of bytes of a successfully executed operation.

Block Operations

Applications using the NVM block write functionality need to perform the following:

1. The system should have completed necessary initialization and [DRV_NVM_Tasks](#) should either be running in a polled environment, or in an interrupt environment.
2. Open the driver using [DRV_NVM_Open](#) with the necessary intent.
3. Provide the desired mode of operation using the operation mode.
4. Provide the source and destination addresses and begin write using [DRV_NVM_Write](#).
5. The client will be able to close itself using the [DRV_NVM_Close](#) when required.

```
DRV_HANDLE      handle;    // Returned from DRV_NVM_Open
char            myBuffer[2 * DRV_NVM_ROW_SIZE];

// Destination address should be row aligned.
char            *destAddress = (char *)NVM_BASE_ADDRESS_TO_WRITE;

unsigned int     count = 2 * MY_BUFFER_SIZE;
DRV_NVM_BUFFER_HANDLE bufferHandle;

bufferHandle = DRV_NVM_Write(myNVMHandle, destAddress, &myBuffer[total], count);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait till the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

Applications using the NVM block Read functionality need to perform the following:

1. The system should have completed necessary initialization and [DRV_NVM_Tasks](#) should either be running in a polled environment, or in an interrupt environment.
2. Open the driver using [DRV_NVM_Open](#) with the necessary intent.
3. Provide the desired mode of operation using the operation mode.
4. Provide the source and destination addresses and begin write using [DRV_NVM_ReadBlock](#).
5. The client will be able to close itself using the [DRV_NVM_Close](#) when required.

```
DRV_HANDLE      handle;    // Returned from DRV_NVM_Open
char            myBuffer[MY_BUFFER_SIZE];
char            *srcAddress = NVM_BASE_ADDRESS_TO_READ_FROM;
unsigned int     count = MY_BUFFER_SIZE;
DRV_NVM_BUFFER_HANDLE bufferHandle;

bufferHandle = DRV_NVM_Read(myNVMHandle, &myBuffer[total], srcAddress, count);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}
```

```

}

// Wait till the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);

```

Internally, a buffer is maintained whose status can be obtained by calling API [DRV_NVM_BufferStatus](#).

5.1.6.5 Configuring the Library

Macros

	Name	Description
	DRV_NVM_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_NVM_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_NVM_ERASE_SIZE	Specifies the NVM Driver Erase Page Size in bytes.
	DRV_NVM_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_NVM_INTERRUPT_MODE	Macro specifies operation of the driver to be in the interrupt mode or polled mode
	DRV_NVM_ROW_SIZE	Specifies the NVM Driver Program Row Size in bytes.
	DRV_NVM_UNLOCK_KEY1	Specifies the NVM Driver Program Unlock Key 1
	DRV_NVM_UNLOCK_KEY2	Specifies the NVM Driver Program Unlock Key 2

Description

The configuration of the NVM Driver is based on the file sys_config.h

This header file contains the configuration selection for the NVM Driver. Based on the selections made, the NVM Driver will support or not support selected features. These configuration settings will apply to all instances of the NVM Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.6.5.1 DRV_NVM_BUFFER_OBJECT_NUMBER Macro

C

```
#define DRV_NVM_BUFFER_OBJECT_NUMBER 5
```

Description

NVM Driver maximum number of buffer objects

This definition selects the maximum number of buffer objects. This indirectly also specifies the queue depth. The NVM Driver can queue up [DRV_NVM_BUFFER_OBJECT_NUMBER](#) of read/write/erase requests before return a [DRV_NVM_BUFFER_HANDLE_INVALID](#) due to the queue being full. Increasing this number increases the RAM requirement of the driver.

Remarks

None.

5.1.6.5.2 DRV_NVM_CLIENTS_NUMBER Macro

C

```
#define DRV_NVM_CLIENTS_NUMBER 1
```

Description

NVM maximum number of clients

This definition selects the maximum number of clients that the NVM driver can supported at run time.

Remarks

None.

5.1.6.5.3 DRV_NVM_ERASE_SIZE Macro

C

```
#define DRV_NVM_ERASE_SIZE 4096
```

Description

NVM Driver Erase Page Size.

This definition specifies the NVM Driver Erase Page Size in bytes. This parameter is device specific and should be obtained from the device specific data sheet. The Erase Page Size is the minimum block size that can be erased in one erase operation.

Remarks

None.

5.1.6.5.4 DRV_NVM_INSTANCES_NUMBER Macro

C

```
#define DRV_NVM_INSTANCES_NUMBER 1
```

Description

NVM Driver instance configuration

This definition selects the maximum number of Driver instances that can be supported by the dynamic driver. In case of this driver, multiple instances of the driver could use the same hardware instance.

Remarks

None

5.1.6.5.5 DRV_NVM_INTERRUPT_MODE Macro

C

```
#define DRV_NVM_INTERRUPT_MODE false
```

Description

NVM interrupt and polled mode operation control

This macro specifies operation of the driver to be in the interrupt mode or polled mode

- true - Select if interrupt mode of NVM operation is desired
- false - Select if polling mode of NVM operation is desired

Not defining this option to true or false will result in build error.

Remarks

None.

5.1.6.5.6 DRV_NVM_ROW_SIZE Macro

C

```
#define DRV_NVM_ROW_SIZE 512
```

Description

NVM Driver Program Row Size.

This definition specifies the NVM Driver Program Row Size in bytes. This parameter is device specific and should be obtained from the device specific data sheet. The Program Row Size is the minimum block size that can be programmed in one program operation.

Remarks

None.

5.1.6.5.7 DRV_NVM_UNLOCK_KEY1 Macro

C

```
#define DRV_NVM_UNLOCK_KEY1 0xAA996655
```

Description

NVM Driver Program Unlock Key 1

This definition specifies the NVM Driver Program Unlock Key 1 parameter is device specific and should be obtained from the device specific data sheet.

Remarks

None.

5.1.6.5.8 DRV_NVM_UNLOCK_KEY2 Macro

C

```
#define DRV_NVM_UNLOCK_KEY2 0x556699AA
```

Description

NVM Driver Program Unlock Key 2

This definition specifies the NVM Driver Program Unlock Key 2 parameter is device specific and should be obtained from the device specific data sheet.

Remarks

None.

5.1.6.6 Building the Library

This section lists the files that are available in the \src of the NVM driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.1.6.7 Library Interface

Data Types and Constants

	Name	Description
	<u>DRV_NVM_BUFFER_HANDLE_INVALID</u>	This value defines the NVM Driver Buffer Invalid handle.
	<u>DRV_NVM_INDEX_0</u>	NVM driver index definitions
	<u>DRV_NVM_INDEX_COUNT</u>	Number of valid NVM driver indices
	<u>DRV_NVM_BUFFER_HANDLE</u>	This type defines the NVM Driver Buffer handle.
	<u>DRV_NVM_BUFFER_STATUS</u>	Specifies the status of the buffer for the read, write and erase operations.
	<u>DRV_NVM_CLIENT_STATUS</u>	Defines the client status
	<u>DRV_NVM_INIT</u>	Defines the data required to initialize or reinitialize the NVM driver
	<u>DRV_NVM_INDEX_1</u>	This is macro DRV_NVM_INDEX_1.

Client Block Data Functions

	Name	Description
	<u>DRV_NVM_Erase</u>	Erase the specified number of pages in flash memory.
	<u>DRV_NVM_Read</u>	Reads a block of data from the specified address in memory.
	<u>DRV_NVM_Tasks</u>	Maintains the driver's erase and write state machine and implements its ISR.
	<u>DRV_NVM_Write</u>	Write a block of data to a specified address in memory.

Client Core Functions

	Name	Description
	<u>DRV_NVM_ClientStatus</u>	Gets current client-specific status the NVM driver
	<u>DRV_NVM_Close</u>	Closes an opened-instance of the NVM driver
	<u>DRV_NVM_Open</u>	Opens the specified timer driver instance and returns a handle to it

Miscellaneous Functions

	Name	Description
	<u>DRV_NVM_BufferStatus</u>	Gets the current status of the buffer.

System Functions

	Name	Description
	<u>DRV_NVM_Deinitialize</u>	Deinitializes the specified instance of the NVM driver module
	<u>DRV_NVM_Initialize</u>	Initializes the NVM instance for the specified driver index
	<u>DRV_NVM_Status</u>	Gets the current status of the NVM driver module.

Description

This section describes the Application Programming Interface (API) functions of the NVM Driver Library.

Refer to each section for a detailed description.

5.1.6.7.1 System Functions

5.1.6.7.1.1 DRV_NVM_Deinitialize Function

C

```
void DRV_NVM_Deinitialize(  
    SYS_MODULE_OBJ object  
);
```

Description

Deinitializes the specified instance of the NVM driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Preconditions

Function [DRV_NVM_Initialize](#) should have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_NVM_Initialize routine

Returns

None.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This routine will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the [DRV_NVM_Status](#) operation. The system has to use [DRV_NVM_Status](#) to find out when the module is in the ready state.

Example

```
// This code snippet shows an example  
// of deinitializing the driver.  
  
SYS_MODULE_OBJ    object;    // Returned from DRV_NVM_Initialize  
SYS_STATUS        status;  
  
DRV_NVM_Deinitialize(object);  
  
status = DRV_NVM_Status(object);  
if (SYS_MODULE_DEINITIALIZED != status)  
{  
    // Check again later if you need to know  
    // when the driver is deinitialized.  
}
```

5.1.6.7.1.2 DRV_NVM_Initialize Function

C

```
SYS_MODULE_OBJ DRV_NVM_Initialize(  
    const SYS_MODULE_INDEX index,  
    const SYS_MODULE_INIT * const init  
);
```

Description

This routine initializes the NVM driver instance for the specified driver index, making it ready for clients to open and use it.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the instance to be initialized also the type of memory used
init	Pointer to a data structure containing any data necessary to initialize the driver.

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns `SYS_MODULE_OBJ_INVALID`.

Remarks

This routine must be called before any other NVM routine is called.

This routine should only be called once during system initialization unless `DRV_NVM_Deinitialize` is called to de-initialize the driver instance.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to re-initialize, it will be reported by the `DRV_NVM_Status` operation. The system must use `DRV_NVM_Status` to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this routine.

Example

```
// This code snippet shows an example
// of initializing the NVM Driver.

DRV_NVM_INIT    NVMInitData;
SYS_MODULE_OBJ  objectHandle;

NVMInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
NVMInitData.moduleId              = NVM_ID_0;
NVMInitData.flashInterruptSource = INT_SOURCE_FLASH_CONTROL;

//usage of DRV_NVM_INDEX_0 indicates usage of flash realted APIs
objectHandle = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT*)NVMInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

5.1.6.7.1.3 DRV_NVM_Status Function**C**

```
SYS_STATUS DRV_NVM_Status(
    SYS_MODULE_OBJ object
);
```

Description

This routine provides the current status of the NVM driver module.

Preconditions

Function `DRV_NVM_Initialize` should have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_NVM_Initialize routine

Returns

SYS_STATUS_READY - Indicates that the driver is ready and [accept](#) requests for new operations.

Note Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to [accept](#) new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Remarks

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been de-initialized

This value is less than SYS_STATUS_ERROR

The this operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This routine will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Example

```

SYS_MODULE_OBJ    object;    // Returned from DRV_NVM_Initialize
SYS_STATUS        NVMStatus;

NVMStatus = DRV_NVM_Status(object);
else if (SYS_STATUS_ERROR >= NVMStatus)
{
    // Handle error
}

```

5.1.6.7.2 Client Core Functions

5.1.6.7.2.1 DRV_NVM_ClientStatus Function

C

```

DRV_NVM_CLIENT_STATUS DRV_NVM_ClientStatus(
    const DRV_HANDLE handle
);

```

Description

This routine gets the client-specific status of the NVM driver associated with the given handle.

Preconditions

The [DRV_NVM_Initialize](#) routine must have been called.

[DRV_NVM_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

A [DRV_NVM_CLIENT_STATUS](#) value describing the current status of the driver

Remarks

This routine will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE          handle;           // Returned from DRV_NVM_Open
DRV_NVM_CLIENT_STATUS clientStatus;

clientStatus = DRV_NVM_ClientStatus(handle);
if(DRV_NVM_CLIENT_STATUS_ERROR >= clientStatus)
{
    // Handle the error
}
```

5.1.6.7.2.2 DRV_NVM_Close Function**C**

```
void DRV_NVM_Close(
    const DRV_HANDLE handle
);
```

Description

This routine closes an opened-instance of the NVM driver, invalidating the handle.

Preconditions

The [DRV_NVM_Initialize](#) routine must have been called for the specified NVM driver instance.

[DRV_NVM_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

None

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling [DRV_NVM_Open](#) before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the [DRV_NVM_Status](#) operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Example

```
DRV_HANDLE handle; // Returned from DRV_NVM_Open
```

```
DRV_NVM_Close(handle);
```

5.1.6.7.2.3 DRV_NVM_Open Function

C

```
DRV_HANDLE DRV_NVM_Open(
    const SYS_MODULE_INDEX index,
    const DRV_IO_INTENT ioIntent
);
```

Description

This routine opens the specified NVM driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Preconditions

Function [DRV_NVM_Initialize](#) must have been called before calling this function.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "or'd" together to indicate the intended use of the driver

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is [DRV_HANDLE_INVALID](#).

Remarks

The handle returned is valid until the [DRV_NVM_Close](#) routine is called.

This routine will NEVER block waiting for hardware.

If the DRV_IO_INTENT_BLOCKING is requested and the driver was built appropriately to support blocking behavior, then other client-level operations may block waiting on hardware until they are complete.

If DRV_IO_INTENT_NON_BLOCKING is requested the driver client can call the [DRV_NVM_ClientStatus](#) operation to find out when the module is in the ready state.

If the requested intent flags are not supported, the routine will return [DRV_HANDLE_INVALID](#).

Example

```
DRV_HANDLE handle;

handle = DRV_NVM_Open(DRV_NVM_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

5.1.6.7.3 Client Block Data Functions

5.1.6.7.3.1 DRV_NVM_Erase Function

C

```
DRV_NVM_BUFFER_HANDLE DRV_NVM_Erase(
    const DRV_HANDLE handle,
```

```
uint8_t * targetbuffer,
const unsigned int numbytes
);
```

Description

This routine erases the specified number of pages in flash memory. It returns a buffer handle that can be queried by the [DRV_NVM_BufferStatus\(\)](#) function to check for completion of the operation. The target address should be aligned on a `DRV_NVM_PAGE_SIZE` byte boundary. The number of bytes to write should be equal to or should be multiples of `DRV_NVM_PAGE_SIZE`.

Preconditions

The [DRV_NVM_Initialize](#) routine must have been called for the specified NVM driver instance.

[DRV_NVM_Open](#) must have been called to obtain a valid opened device handle.

`DRV_IO_INTENT_READ` or `DRV_IO_INTENT_READWRITE` must have been specified in the [DRV_NVM_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
targetBuffer	Start address in NVM memory from where the erase should begin. Should be aligned on a <code>DRV_NVM_PAGE_SIZE</code> byte boundary.
numbytes	Total number of pages to be erased expressed in bytes. This should be equal to or a multiple of <code>DRV_NVM_PAGE_SIZE</code> .

Returns

[DRV_NVM_BUFFER_HANDLE_INVALID](#) in case the operation was not successful. A valid handle otherwise.

Remarks

If the `DRV_IO_INTENT_BLOCKING` flag was given in the call to the [DRV_NVM_Open](#) routine and the driver was built to support blocking behavior the call will block until the operation is complete.

If the `DRV_IO_INTENT_NONBLOCKING` flag was given in the call to the [DRV_NVM_Open](#) routine or the driver was built to only support non-blocking behavior the call will return immediately.

Example

```
// Returned from DRV_NVM_Open
DRV_HANDLE    handle;

// Destination address should be row aligned.
char          *destAddress = (char *)NVM_BASE_ADDRESS_TO_ERASE;

unsigned int   count = 2 * DRV_NVM_PAGE_SIZE;
DRV_NVM_BUFFER_HANDLE bufferHandle;

bufferHandle = DRV_NVM_Write(myNVMHandle, destAddress, count);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait till the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

5.1.6.7.3.2 DRV_NVM_Read Function

C

```
DRV_NVM_BUFFER_HANDLE DRV_NVM_Read(
    const DRV_HANDLE handle,
    uint8_t * targetBuffer,
    uint8_t * srcAddress,
    const unsigned int numbytes
);
```

Description

This routine reads a block of data from the specified address in memory. It returns a buffer handle that can be queried by the [DRV_NVM_BufferStatus\(\)](#) function to check for completion of the operation.

Preconditions

The [DRV_NVM_Initialize](#) routine must have been called for the specified NVM driver instance.

[DRV_NVM_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_NVM_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
targetBuffer	Buffer into which the data read from the NVM instance will be placed.
srcAddress	The base address in NVM memory from data read should begin
numbytes	Total number of bytes that need to be read from the module instance (must be equal to or less than the size of the buffer)

Returns

[DRV_NVM_BUFFER_HANDLE_INVALID](#) in case the operation was not successful. A valid handle otherwise.

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_NVM_Open](#) routine and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_NVM_Open](#) routine or the driver was built to only support non-blocking behavior the call will return immediately, identifying how many bytes of data were actually copied into the client's buffer and the client must call DRV_NVM_Read again with adjusted parameters as shown in the example.

Example

```
DRV_HANDLE    handle;    // Returned from DRV_NVM_Open
char          myBuffer[MY_BUFFER_SIZE];
char          *srcAddress = NVM_BASE_ADDRESS_TO_READ_FROM;
unsigned int   count = MY_BUFFER_SIZE;
DRV_NVM_BUFFER_HANDLE bufferHandle;

bufferHandle = DRV_NVM_Read(myNVMHandle, &myBuffer[total], srcAddress, count);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait till the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

5.1.6.7.3.3 DRV_NVM_Tasks Function

C

```
void DRV_NVM_Tasks(
    SYS_MODULE_OBJ object
);
```

Description

This routine is used to maintain the driver's internal write and erase state machine and implement its ISR for interrupt-driven implementations.

Preconditions

The [DRV_NVM_Initialize](#) routine must have been called for the specified NVM driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_NVM_Initialize)

Returns

None.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_NVM_Initialize

while (true)
{
    DRV_NVM_Tasks (object);

    // Do other tasks
}
```

5.1.6.7.3.4 DRV_NVM_Write Function

C

```
DRV_NVM_BUFFER_HANDLE DRV_NVM_Write(
    DRV_HANDLE handle,
    uint8_t * destination,
    uint8_t * source,
    const unsigned int nBytes
);
```

Description

This routine writes a block of data to a specified address in memory. It returns a buffer handle that can be queried by the [DRV_NVM_BufferStatus\(\)](#) function to check for completion of the operation. The contents of the source buffer should not be changed while the operation is in progress. The target address should be aligned on a [DRV_NVM_ROW_SIZE](#) byte boundary. The number of bytes to write should be equal to or should be multiples of [DRV_NVM_ROW_SIZE](#).

Preconditions

The [DRV_NVM_Initialize](#) routine must have been called for the specified NVM driver instance.

[DRV_NVM_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_NVM_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
targetBuffer	Buffer in NVM memory where the data from srcAddress will be placed. Should be aligned on a DRV_NVM_ROW_SIZE byte boundary.
srcAddress	The source buffer containing data to programmed into NVM
numbytes	Total number of bytes that need to be written to the NVM. This should be equal to or a multiple of DRV_NVM_ROW_SIZE .

Returns

[DRV_NVM_BUFFER_HANDLE_INVALID](#) in case the operation was not successful. A valid handle otherwise.

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_NVM_Open](#) routine and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_NVM_Open](#) routine or the driver was built to only support non-blocking behavior the call will return immediately.

Example

```
DRV_HANDLE    handle;    // Returned from DRV_NVM_Open
char          myBuffer[2 * DRV_NVM_ROW_SIZE];

// Destination address should be row aligned.
char          *destAddress = (char *)NVM_BASE_ADDRESS_TO_WRITE;

unsigned int   count = 2 * MY_BUFFER_SIZE;
DRV_NVM_BUFFER_HANDLE bufferHandle;

bufferHandle = DRV_NVM_Write(myNVMHandle, destAddress, &myBuffer[total], count);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait till the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

5.1.6.7.4 Client Basic Functions

5.1.6.7.5 Miscellaneous Functions

5.1.6.7.5.1 DRV_NVM_BufferStatus Function

C

```
DRV_NVM_BUFFER_STATUS DRV_NVM_BufferStatus(
    DRV_HANDLE handle,
    const DRV_HANDLE bufferHandle
);
```

Description

This routine gets the current status of the buffer. The application must use this routine in case a polling based implementation is desired.

Preconditions

The [DRV_NVM_Initialize](#) routine must have been called.

[DRV_NVM_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

A [DRV_NVM_BUFFER_STATUS](#) value describing the current status of the buffer

Remarks

This routine will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE      handle;           // Returned from DRV_NVM_Open
DRV_HANDLE      bufferHandle;
DRV_NVM_BUFFER_STATUS status;

status = DRV_NVM_BufferStatus(handle, bufferHandle);
if(status == DRV_NVM_BUFFER_COMPLETED)
{
    // Operation Done
}
```

5.1.6.7.6 Data Types and Constants

5.1.6.7.6.1 DRV_NVM_BUFFER_HANDLE_INVALID Macro

C

```
#define DRV_NVM_BUFFER_HANDLE_INVALID
```

Description

NVM Driver Buffer Invalid Handle

This value defines the NVM Driver Buffer Invalid handle. This value is returned by read/write/erase routines when the desired operation could not be completed.

Remarks

None.

5.1.6.7.6.2 DRV_NVM_INDEX_0 Macro

C

```
#define DRV_NVM_INDEX_0 0
```

Description

Driver NVM Module Index reference

These constants provide NVM driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the [DRV_NVM_Initialize](#) and [DRV_NVM_Open](#) routines to identify the driver instance in use.

5.1.6.7.6.3 DRV_NVM_INDEX_COUNT Macro**C**

```
#define DRV_NVM_INDEX_COUNT 1
```

Description

NVM Driver Module Index Count

This constant identifies NVM driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is part-specific.

5.1.6.7.6.4 DRV_NVM_BUFFER_HANDLE Type**C**

```
typedef uintptr_t DRV_NVM_BUFFER_HANDLE;
```

Description

NVM Driver Buffer Handle

This type defines the NVM Driver Buffer handle.

Remarks

None.

5.1.6.7.6.5 DRV_NVM_BUFFER_STATUS Enumeration**C**

```
typedef enum {
    DRV_NVM_BUFFER_COMPLETED = 0,
    DRV_NVM_BUFFER_QUEUED = 1,
    DRV_NVM_BUFFER_IN_PROGRESS = 2,
    DRV_NVM_BUFFER_ERROR_UNKNOWN = -1
} DRV_NVM_BUFFER_STATUS;
```

Description

NVM Driver Buffer Status

This type specifies the status of the buffer for the read, write and erase operations.

Members

Members	Description
DRV_NVM_BUFFER_COMPLETED = 0	Done OK and ready
DRV_NVM_BUFFER_QUEUED = 1	Scheduled but not started
DRV_NVM_BUFFER_IN_PROGRESS = 2	Currently being in transfer
DRV_NVM_BUFFER_ERROR_UNKNOWN = -1	Unknown buffer

Remarks

None.

5.1.6.7.6.6 DRV_NVM_CLIENT_STATUS Enumeration

C

```
typedef enum {
    DRV_NVM_CLIENT_STATUS_READY,
    DRV_NVM_CLIENT_STATUS_BUSY,
    DRV_NVM_WRITE_TERMINATED,
    DRV_NVM_ERASE_TERMINATED,
    DRV_NVM_LOW_VOLTAGE_ERROR,
    DRV_NVM_LOW_VOLTAGE_DETECT_ERROR,
    DRV_NVM_STATUS_INVALID
} DRV_NVM_CLIENT_STATUS;
```

Description

NVM Client Status

Defines the various client status codes.

Members

Members	Description
DRV_NVM_CLIENT_STATUS_READY	Up and running, ready to start new operations
DRV_NVM_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_NVM_WRITE_TERMINATED	Write operation terminated Occured
DRV_NVM_ERASE_TERMINATED	Erase operation terminated Occured
DRV_NVM_LOW_VOLTAGE_ERROR	Low voltage Error
DRV_NVM_LOW_VOLTAGE_DETECT_ERROR	Low voltage Error
DRV_NVM_STATUS_INVALID	Client Invalid

Remarks

None

5.1.6.7.6.7 DRV_NVM_INIT Structure

C

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    NVM_MODULE_ID nvmID;
    INT_SOURCE interruptSource;
} DRV_NVM_INIT;
```

Description

NVM Driver Initialization Data

This data type defines the data required to initialize or reinitialize the NVM driver.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
NVM_MODULE_ID nvmID;	Identifies NVM hardware module (PLIB-level) ID
INT_SOURCE interruptSource;	Interrupt Source for Write Interrupt

Remarks

Not all the init features are available for all the microcontrollers

5.1.6.7.6.8 DRV_NVM_INDEX_1 Macro**C**

```
#define DRV_NVM_INDEX_1 1
```

Description

This is macro DRV_NVM_INDEX_1.

5.1.6.8 Files**Files**

Name	Description
drv_nvm.h	NVM Driver Interface Definition
drv_nvm_config_template.h	NVM driver configuration definitions template

Description**5.1.6.8.1 drv_nvm.h**

NVM Driver Interface Definition

The NVM device driver provides a simple interface to manage the NVM modules on Microchip microcontrollers. This file defines the interface definition for the NVM driver.

Enumerations

	Name	Description
	DRV_NVM_BUFFER_STATUS	Specifies the status of the buffer for the read, write and erase operations.
	DRV_NVM_CLIENT_STATUS	Defines the client status

Functions

	Name	Description
	DRV_NVM_BufferStatus	Gets the current status of the buffer.
	DRV_NVM_ClientStatus	Gets current client-specific status the NVM driver
	DRV_NVM_Close	Closes an opened-instance of the NVM driver
	DRV_NVM_Deinitialize	Deinitializes the specified instance of the NVM driver module
	DRV_NVM_Erase	Erase the specified number of pages in flash memory.
	DRV_NVM_Initialize	Initializes the NVM instance for the specified driver index
	DRV_NVM_Open	Opens the specified timer driver instance and returns a handle to it
	DRV_NVM_Read	Reads a block of data from the specified address in memory.
	DRV_NVM_Status	Gets the current status of the NVM driver module.
	DRV_NVM_Tasks	Maintains the driver's erase and write state machine and implements its ISR.
	DRV_NVM_Write	Write a block of data to a specified address in memory.

Macros

	Name	Description
	<u>DRV_NVM_BUFFER_HANDLE_INVALID</u>	This value defines the NVM Driver Buffer Invalid handle.
	<u>DRV_NVM_INDEX_0</u>	NVM driver index definitions
	<u>DRV_NVM_INDEX_1</u>	This is macro DRV_NVM_INDEX_1.
	<u>DRV_NVM_INDEX_COUNT</u>	Number of valid NVM driver indices

Structures

	Name	Description
	<u>DRV_NVM_INIT</u>	Defines the data required to initialize or reinitialize the NVM driver

Types

	Name	Description
	<u>DRV_NVM_BUFFER_HANDLE</u>	This type defines the NVM Driver Buffer handle.

File Name

drv_nvm.h

Company

Microchip Technology Incorporated

5.1.6.8.2 drv_nvm_config_template.h

NVM Driver Configuration Definitions for the template version

These definitions statically define the driver's mode of operation.

Macros

	Name	Description
	<u>DRV_NVM_BUFFER_OBJECT_NUMBER</u>	Selects the maximum number of buffer objects
	<u>DRV_NVM_CLIENTS_NUMBER</u>	Selects the maximum number of clients
	<u>DRV_NVM_ERASE_SIZE</u>	Specifies the NVM Driver Erase Page Size in bytes.
	<u>DRV_NVM_INSTANCES_NUMBER</u>	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	<u>DRV_NVM_INTERRUPT_MODE</u>	Macro specifies operation of the driver to be in the interrupt mode or polled mode
	<u>DRV_NVM_ROW_SIZE</u>	Specifies the NVM Driver Program Row Size in bytes.
	<u>DRV_NVM_UNLOCK_KEY1</u>	Specifies the NVM Driver Program Unlock Key 1
	<u>DRV_NVM_UNLOCK_KEY2</u>	Specifies the NVM Driver Program Unlock Key 2

File Name

drv_NVM_config_template.h

Company

Microchip Technology Incorporated

5.1.7 Parallel Master Port (PMP) Driver Library

5.1.7.1 Introduction

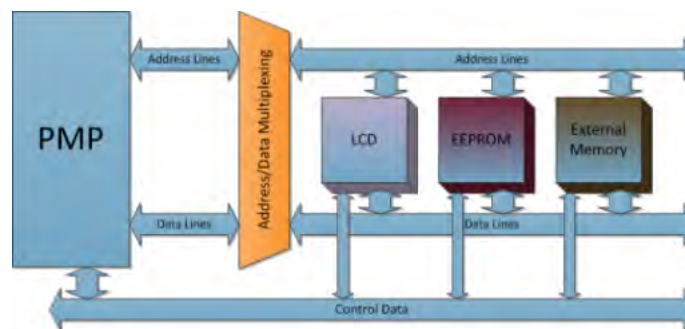
PMP Driver Library
for
Microchip Microcontrollers

This library provides an interface to manage the Parallel Master Port (PMP) module on Microchip family of microcontrollers in different modes of operation.

Description

The Parallel Master Port (PMP) is a parallel 8-bit/16-bit I/O module specifically designed to communicate with a wide variety of parallel devices such as communications peripherals, LCDs, external memory devices and microcontrollers. Because the interfaces to parallel peripherals vary significantly, the PMP module is highly configurable.

The following figure shows a generic block diagram of the ways PMP module can be used:



PMP module can be used in different modes. Master and Slave are the two modes which can further have sub-modes depending on the different microcontroller families.

Master Mode: In master mode, the PMP module can provide a 8-bit or 16-bit data bus, up to 16 bits of address, and all the necessary control signals to operate a variety of external parallel devices such as memory devices, peripherals and slave microcontrollers. The PMP master modes provide a simple interface for reading and writing data, but not executing program instructions from external devices, such as SRAM or Flash memories.

Slave Mode: Slave modes support 8-bit data only and the module control pins are automatically dedicated when this mode is selected.

5.1.7.2 Release Notes

MPLAB Harmony Version: v0.70b **PMP Driver Library Version :** 0.01 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.1.7.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.7.4 Using the Library

This topic describes the basic architecture of the PMP Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_pmp.h](#)

The interface to the PMP Driver library is defined in the "[drv_pmp.h](#)" header file. This file is included by the "drv.h" file. Any C language source (.c) file that uses the PMP Driver Library should include "drv.h".

Library File: The PMP Driver Library archive (.a) file installed with MPLAB Harmony.

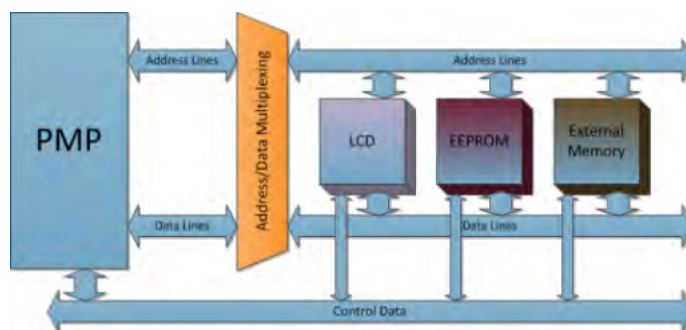
Please refer to the [MPLAB Harmony Overview](#) for how the driver interacts with the framework.

5.1.7.4.1 Abstraction Model

This library provides a low-level abstraction of the Parallel Master Port (PMP) module on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description**Hardware Abstraction Model Description**

The PMP module provides interface routines to interact with external peripherals such as LCD, EEPROM, Flash memory, etc., as shown in the following diagram. The diagram shows the PMP module acting as a master. The PMP module can be easily configured to act as a slave. The address and data lines can be multiplexed to suit the application. The address and data buffers are up to 2-byte (16-bit) buffers for data transmitted or received by the parallel interface to the PMP bus over the data and address lines synchronized with control logic including the read and write strobe.



PMP Hardware Abstraction Model Diagram

The desired timing wait states to suit different peripheral timings can also be programmed using the PMP module.

5.1.7.4.2 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each sub-section addresses one of the blocks or the overall operation of the PMP module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, re-initialization, tasks, and status functions.
Client Interaction Functions	Provides open, close, client status and client mode configuration functions.
Client Transfer Functions	Provides interface for data transfer in master and slave mode.
Miscellaneous	Provides driver miscellaneous functions, version identification functions etc.

5.1.7.4.3 How the Library Works

This section describes how the PMP Driver Library operates.

Description

Before the driver is ready for use, it should be configured (compile time configuration). Refer to the [Configuring the Library](#) section for more details on how to configure the driver.

There are few run-time configuration items that are done during initialization of the driver instance, and a few that are client-specific and are done using dedicated functions.

To use the PMP Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation.

The following is the sequence in which various routines should be called:

1. Call `DRV_PMP_Init` to initialize the PMP Driver. Note that this may be performed by the MPLAB Harmony system module. The `DRV_PMP_Status` function may be used to check the status of the initialization.
2. Once initialization for a particular driver instance is done, the client wanting to use the driver can open it using API `DRV_PMP_Open`.
3. The `DRV_PMP_ModeConfig` function should now be called, which will configure the driver for the exact mode of operation required by that client.

4. After configuring the mode, [DRV_PMP_Write](#) and/or [DRV_PMP_Read](#) can be called by the user application to Write/Read using the PMP module. Calling these functions does not start the PMP transfer immediately in non-interrupt mode. Instead, all of these transfer tasks are queued in an internal queue. Actual transfer starts only when the PMP Task function is called by the system/user. In interrupt mode, although transfer tasks are queued, the actual transfer starts immediately.
5. PMP Write and Read functions return an ID of that particular transfer, which should be saved by user to get the status of that transfer later.
6. The system will either call [DRV_PMP_Tasks](#) from the System Task Service (in a polled environment), or it will be called from interrupt service routine of the PMP.
7. At any time status of the transfer can be obtained by using [DRV_PMP_TransferStatus](#).

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

5.1.7.4.3.1 System Initialization

This section describes initialization and reinitialization features.

Description

Initialization and Reinitialization

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the PMP device will be initialized with the following configuration settings:

Initialization member	Description
moduleInit	System module initialization of the power state.
pmpId	PMP hardware module ID (peripheral library-level ID).
stopInIdle	Decide whether or not the module should be stopped in Idle mode.
muxMode	To select one of the different multiplexing modes possible for PMP module.
inputBuffer	Select the type of Input Buffer will be of TTL or Schmitt Trigger.
polarity	Select polarity of different PMP pins.
ports	Set the pins the user wants to use as port or PMP pins.

The [DRV_PMP_Initialize](#) function returns an object handle of the type [SYS_MODULE_OBJ](#). After this, the object handle returned by the initialize interface would be used by the other system interfaces, such as [DRV_PMP_Reinitialize](#), [DRV_PMP_Deinitialize](#), [DRV_PMP_Status](#), and [DRV_PMP_Tasks](#).

Example for PMP Initialization Through the [DRV_PMP_INIT](#) Structure

```

DRV_PMP_INIT    init;
SYS_MODULE_OBJ  object;
SYS_STATUS      pmpStatus;

// populate the PMP init configuration structure
init.inputBuffer = PMP_INPUT_BUFFER_TTL;
init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS |
PMP_PMA14_PORT;
init.ports.readWriteStrobe = PORT_ENABLE;
init.ports.writeEnableStrobe = PORT_ENABLE;

```

```

    init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
    init.pmpID            = PMP_ID_0;
    init.stopInIdle       = false;
    init.muxMode          = PMP_MUX_NONE;

    object = DRV_PMP_Initialize (DRV_PMP_INDEX_0, (SYS_MODULE_INIT *)&init);

    pmpStatus = DRV_PMP_Status(object);

    if ( SYS_STATUS_READY != pmpStatus)
    {
        // Handle error
    }

```

Deinitialization

Once the initialize operation has been called, the deinitialize operation must be called before the initialize operation can be called again. This routine may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the function will never block for hardware PMP access. If the operation requires time to allow the hardware to complete, which will be reported by [DRV_PMP_Status](#).

Status

PMP status is available to query the module state before, during and after initialization, deinitialization, and reinitialization.

Tasks Routine

The [DRV_PMP_Tasks](#) function will see the queue status and perform the task of transferring the data accordingly. In the Blocking mode when interrupts are disabled, it will finish one of the tasks completely (that means emptying one space in queue), and then return back. Whereas in Non-Blocking mode, it will return back just after starting one word (8-bit or 16-bit) of transfer (may not be emptying one space in the queue, as that task may not be completely finished).

The [DRV_PMP_Tasks](#) function can be called in two ways:

- By the system task service in a polled environment
- By the interrupt service routine of the PMP in an interrupt-based system

Example: Polling

```

int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_PMP_Initialize( DRV_PMP_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_PMP_Status( object ) )
        return 0;

    while (1)
    {
        DRV_PMP_Tasks (object);
    }
}

```

Example: Interrupt

```

int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_PMP_Initialize( DRV_PMP_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_PMP_Status( object ) )
        return 0;

    while (1);
}

```

```

}

/* Sample interrupt routine not specific to any device family */
void ISR PMPInterrupt(void)
{
    //Call the PMP Tasks routine
    DRV_PMP_Tasks(object);
}

```

Note: A PMP transfer in Blocking mode in an interrupt environment is not supported.

5.1.7.4.3.1.1 Transfer Operation

This section describes transfer operation.

Description

Once the PMP Driver is open and configured for a client, it is set to start Reading/Writing through [DRV_PMP_Read](#) and [DRV_PMP_Write](#). However, these functions will not directly start reading or writing. These will just put the relevant information in a queue in non-interrupt mode and return an ID that can be used later for checking the transfer status. In Interrupt mode, the Read/Write functions will trigger the transfer immediately after storing the transfer information in the queue.

The user must use a buffer pointing to character for data values.

The repeatCount parameter allows the user to repeatedly write the same nBytes of data into the slave devices.

Example:

```

unsigned char myReadBuffer[300], myWriteBuffer[100]; // has to be 'char' arrays
uint32_t deviceAddress, nBytes, repeatCount, i;
uint32_t writeID, readID;
DRV_HANDLE handle;

//initialize, open and configure the driver/client
/* ... */

deviceAddress = 0x0206;
nBytes = 100;
repeatCount = 0x01;
for (i=0; i<nBytes; i++)
{
    myWriteBuffer[i]=i*5+7;
}

/* it will write 100 bytes of data in the location starting from 0x0206 and then it will repeat
   writing the same set of data in next 100 location starting from 0x206+100 for 8 bit data mode
   and 50 location starting from 0x206+50 for 16 bit data mode. */
writeID = DRV_PMP_Write ( handle, deviceAddress, &myWriteBuffer[0], nBytes, repeatCount);

// it will read 300 locations starting from 0x0206 into myReadBuffer
readID = DRV_PMP_Read ( handle, deviceAddress, &myReadBuffer[0], nBytes);

```

Transfer Status

The status of the read/write transfers can be obtained using API [DRV_PMP_TransferStatus](#).

Example:

```

DRV_PMP_TRANSFER_STATUS writeStatus, readStatus;
uint32_t writeID, readID;

writeStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, writeID);
readStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, readID);

```

5.1.7.4.3.1.2 Client Operation

This section describes general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the [DRV_PMP_Open](#) function with a specific intent. This provides the configuration required to open the PMP instance for operation. If the driver is deinitialized using the function [DRV_PMP_Deinitialize](#), the application must call the [DRV_PMP_Open](#) function again to set up the instance of the PMP. The function [DRV_PMP_Open](#) need not be called again if the system is reinitialized using the [DRV_PMP_Reinitialize](#) function.

The PMP driver supports DRV_IO_INTENT_NONBLOCKING, DRV_IO_INTENT_BLOCKING, DRV_IO_INTENT_EXCLUSIVE, and DRV_IO_INTENT_SHARED IO.

Example:

```
DRV_HANDLE handle;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

if( handle == DRV_HANDLE_INVALID )
{
    // Client cannot open the instance.
}
```

The function [DRV_PMP_Close](#) closes an already opened instance of the PMP driver, invalidating the handle. [DRV_PMP_Open](#) must have been called to obtain a valid opened device handle.

Example:

```
DRV_HANDLE handle;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

/*...*/

DRV_PMP_Close( handle );
```

The client has the option to check the status through the function [DRV_PMP_ClientStatus](#).

Example:

```
DRV_HANDLE handle;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

if ( DRV_PMP_CLIENT_STATUS_OPEN != DRV_PMP_ClientStatus( handle ) )
    return 0;
```

Client Mode Setting

Any client-specific PMP configuration has to be done using a separate function, [DRV_PMP_ModeConfig](#). This function must be called after the client is open using [DRV_PMP_Open](#).

Following are the client-specific configuration parameters the user can set using this function:

Configuration Parameter	Description
pmpMode	Selects the PMP mode (master or slave) to use.

intMode	Selects the interrupt mode to use.
incrementMode	Sets up address for either auto-increment or decrement mode.
endianMode	Sets Little/Big endian mode.
portSize	Specifies the data width (8-bit or 16-bit).
waitStates	Selects the different wait states.
chipSelect	Selects the Chip Select line.

Example:

```

DRV_HANDLE handle;
DRV_PMP_MODE_CONFIG config;

config.chipSelect = PMCS1_AND_PMCS2_AS_CHIP_SELECT;
config.endianMode = LITTLE_ENDIAN;
config.incrementMode = PMP_ADDRESS_AUTO_INCREMENT;
config.intMode = PMP_INTERRUPT_NONE;
config.pmpMode = PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT; //Master Mode 2
config.portSize = PMP_DATA_SIZE_8_BITS;
config.waitStates.dataHoldWait = PMP_DATA_HOLD_2;
config.waitStates.dataWait = PMP_DATA_WAIT_THREE;
config.waitStates.strobeWait = PMP_STROBE_WAIT_5;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

// Configure the client
DRV_PMP_ModeConfig ( handle, config );

```

5.1.7.4.3.1.3 Example Code for Complete Operation

A code example of complete operation is provided in this section.

Description

This example code will write 100 bytes of data twice (i.e., repeat once) in the memory location starting from 0x0206, and then it will be read in the buffer, myReadBuffer. The modes selected for this transfer are:

- Non-Blocking
- No Interrupt
- PMP Master Mode 2
- Address Auto-increment
- No Address/Data Lines Multiplexing
- 8-bit data

Example:

```

void main(void)
{
    DRV_PMP_INIT      init;
    SYS_MODULE_OBJ    object;
    SYS_STATUS        pmpStatus;
    DRV_HANDLE        handle;
    DRV_PMP_MODE_CONFIG config;
    unsigned char myReadBuffer[300], myWriteBuffer[100];
    uint32_t deviceAddress, nBytes, repeatCount, i;
    uint32_t writeID, readID;
    DRV_PMP_TRANSFER_STATUS writeStatus=0, readStatus=0;

```

```

// populate the PMP init configuration structure
init.inputBuffer = PMP_INPUT_BUFFER_TTL;
init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS;
init.ports.readWriteStrobe = PORT_ENABLE;
init.ports.writeEnableStrobe = PORT_ENABLE;
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.pmpID = PMP_ID_0;
init.stopInIdle = false;
init.muxMode = PMP_MUX_NONE;

object = DRV_PMP_Initialize (DRV_PMP_INDEX_0, (SYS_MODULE_INIT *)&init);

pmpStatus = DRV_PMP_Status(object);

if ( SYS_STATUS_READY != pmpStatus)
{
    // Handle error
}

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

if( handle == DRV_HANDLE_INVALID )
{
    // Client cannot open the instance.
}

config.chipSelect = PMCS1_AND_PMCS2_AS_CHIP_SELECT;
config.endianMode = LITTLE_ENDIAN;
config.incrementMode = PMP_ADDRESS_AUTO_INCREMENT;
config.intMode = PMP_INTERRUPT_NONE;
config.pmpMode = PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT; //Master Mode 2
config.portSize = PMP_DATA_SIZE_8_BITS;
config.waitStates.dataHoldWait = PMP_DATA_HOLD_2;
config.waitStates.dataWait = PMP_DATA_WAIT_THREE;
config.waitStates.strobeWait = PMP_STROBE_WAIT_5;

// Configure the client
DRV_PMP_ModeConfig ( handle, config );

deviceAddress = 0x0206;
nBytes = 100;
repeatCount = 0x01;
for (i=0; i<nBytes; i++)
{
    myWriteBuffer[i]=i*5+7;
}

writeID = DRV_PMP_Write ( handle, deviceAddress, &myWriteBuffer[0], nBytes, repeatCount);
readID = DRV_PMP_Read ( handle, deviceAddress, &myReadBuffer[0], nBytes*2);

while(!((writeStatus == PMP_TRANSFER_FINISHED)&&(readStatus == PMP_TRANSFER_FINISHED)))
{
    DRV_PMP_Tasks (object);

    writeStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, writeID);
    readStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, readID);
}

while(1);

```

```
}

```

5.1.7.5 Configuring the Library

Macros

	Name	Description
	DRV_PMP_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_PMP_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_PMP_QUEUE_SIZE	PMP queue size for different instances.

Description

The configuration of the PMP driver is based on the file [drv_pmp_config.h](#)

This header file contains the configuration selection for the PMP Driver. Based on the selections made, the PMP Driver will support or not support selected features. These configuration settings will apply to all instances of the PMP Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.7.5.1 DRV_PMP_CLIENTS_NUMBER Macro

C

```
#define DRV_PMP_CLIENTS_NUMBER 2
```

Description

PMP maximum number of clients

This definition select the maximum number of clients that the PMP driver can support at run time.

Remarks

None.

5.1.7.5.2 DRV_PMP_INSTANCES_NUMBER Macro

C

```
#define DRV_PMP_INSTANCES_NUMBER 1
```

Description

PMP hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

None.

5.1.7.5.3 DRV_PMP_QUEUE_SIZE Macro

C

```
#define DRV_PMP_QUEUE_SIZE 8
```


Description

PMP queue size

The PMP queue size for a driver instances should be placed here. If more than one driver instance of PMP is present, then all takes the same queue size.

Remarks

All the transfers (Read/Write) first gets queued and gets completed sequentially when Task API is called in a loop. Therefore, the minimum value of this index should be 1.

5.1.7.6 Building the Library

This section list the files that are available in the \src of the PMP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.1.7.7 Library Interface

Data Types and Constants

	Name	Description
	<u>DRV_PMP_INDEX_COUNT</u>	Number of valid PMP driver indices.
	<u>DRV_PMP_CHIPX_STROBE_MODE</u>	PMP writeEnable/ReadWrite strobes.
	<u>DRV_PMP_CLIENT_STATUS</u>	PMP client status definitions.
	<u>DRV_PMP_ENDIAN_MODE</u>	PMP endian modes.
	<u>DRV_PMP_INDEX</u>	PMP driver index definitions.
	<u>DRV_PMP_INIT</u>	Defines the PMP driver initialization data.
	<u>DRV_PMP_MODE_CONFIG</u>	PMP modes configuration.
	<u>DRV_PMP_POLARITY_OBJECT</u>	PMP polarity object.
	<u>DRV_PMP_PORT_CONTROL</u>	PMP port enable/disable definitions.
	<u>DRV_PMP_PORTS</u>	PMP port configuration.
	<u>DRV_PMP_QUEUE_ELEMENT_OBJ</u>	Defines the object for PMP queue element.
	<u>DRV_PMP_TRANSFER_STATUS</u>	Defines the PMP transfer status.
	<u>DRV_PMP_WAIT_STATES</u>	PMP wait states object.
	<u>QUEUE_ELEMENT_OBJECT</u>	Defines the structure required for maintaining the queue element.
	<u>MAX_NONBUFFERED_BYTE_COUNT</u>	After this number the PMP transfer should be polled to gurantee data trasnfer
	<u>DRV_PMP_TRANSFER_TYPE</u>	This is type DRV_PMP_TRANSFER_TYPE.

Client Interaction Functions

	Name	Description
	<u>DRV_PMP_ClientStatus</u>	Gets the current client-specific status of the PMP driver.
	<u>DRV_PMP_Close</u>	Closes an opened instance of the PMP driver.
	<u>DRV_PMP_ModeConfig</u>	Configure the PMP modes.
	<u>DRV_PMP_Open</u>	Opens the specified PMP driver instance and returns a handle to it.
	<u>DRV_PMP_Read</u>	Read the data from external device.
	<u>DRV_PMP_Write</u>	Transfers the data from the MCU to the external device.

Client Transfer Functions

	Name	Description
	DRV_PMP_TransferStatus	Returns the transfer status.

Miscellaneous Functions

	Name	Description
	DRV_PMP_VersionGet	Gets the PMP driver version in numerical format.
	DRV_PMP_VersionStrGet	Gets the PMP driver version in string format.

System Functions

	Name	Description
	DRV_PMP_Deinitialize	Deinitializes the specified instance of the PMP driver module
	DRV_PMP_Initialize	Initializes the PMP driver.
	DRV_PMP_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings.
	DRV_PMP_Status	Provides the current status of the PMP driver module.
	DRV_PMP_Tasks	Maintains the driver's state machine and implements its ISR.

Description

This section describes the Application Programming Interface (API) functions of the PMP Driver.

Refer to each section for a detailed description.

5.1.7.7.1 System Functions**5.1.7.7.1.1 DRV_PMP_Deinitialize Function****C**

```
void DRV_PMP_Deinitialize(
    const SYS_MODULE_OBJ pmpDriverObject
);
```

Description

This function deinitializes the specified instance of the PMP driver module, disabling its operation (and any hardware). All internal data is invalidated.

Preconditions

The [DRV_PMP_Initialize](#) function must have been called before calling this routine and a valid [SYS_MODULE_OBJ](#) must have been returned.

Parameters

Parameters	Description
pmpDriverObject	Driver object handle, returned from the DRV_PMP_Initialize

Returns

None.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be

reported by the [DRV_PMP_Status](#) operation. The system has to use [DRV_PMP_Status](#) to find out when the module is in the ready state.

Example

```
SYS_MODULE_OBJ    pmpDriverObject;    // Returned from DRV_PMP_Initialize
SYS_STATUS        status;

DRV_PMP_Deinitialize(pmpDriverObject);

status = DRV_PMP_Status(pmpDriverObject);
if (SYS_MODULE_DEINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

5.1.7.7.1.2 DRV_PMP_Initialize Function**C**

```
SYS_MODULE_OBJ DRV_PMP_Initialize(
    const SYS_MODULE_INDEX drvIndex,
    const SYS_MODULE_INIT * const init
);
```

Description

This function initializes the PMP driver, making it ready for clients to open and use it.

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver

Returns

If successful, it returns a valid handle to a driver object. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#). The returned object must be passed as argument to [DRV_PMP_Reinitialize](#), [DRV_PMP_Deinitialize](#), [DRV_PMP_Tasks](#) and [DRV_PMP_Status](#) routines.

Remarks

This function must be called before any other PMP function is called.

This function should only be called once during system initialization unless [DRV_PMP_Deinitialize](#) is called to deinitialize the driver instance.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the [DRV_PMP_Status](#) operation. The system must use [DRV_PMP_Status](#) to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Example

```
DRV_PMP_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the initialization structure
```

```

init.inputBuffer = PMP_INPUT_BUFFER_TTL;
init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS |
PMP_PMA14_PORT;
init.ports.readWriteStrobe = PORT_ENABLE;
init.ports.writeEnableStrobe = PORT_ENABLE;
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.pmpID = PMP_ID_0;
init.stopInIdle = false;
init.muxMode = PMP_MUX_NONE;

// Do something

objectHandle = DRV_PMP_Initialize(DRV_PMP_INDEX_0, (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}

```

5.1.7.7.1.3 DRV_PMP_Reinitialize Function

C

```

void DRV_PMP_Reinitialize(
    const SYS_MODULE_OBJ pmpDriverObject,
    const SYS_MODULE_INIT * const init
);

```

Description

This function reinitializes the driver and refreshes any associated hardware settings using the specified initialization data, but it will not interrupt any ongoing operations.

Preconditions

The [DRV_PMP_Initialize](#) function must have been called before calling this function and a valid [SYS_MODULE_OBJ](#) must have been returned.

Parameters

Parameters	Description
pmpDriverObject	Driver object handle, returned from the DRV_PMP_Initialize

Returns

None.

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to re-initialize, it will be reported by the [DRV_PMP_Status](#) operation. The system must use [DRV_PMP_Status](#) to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Example

```

DRV_PMP_INIT    init;
SYS_MODULE_OBJ  pmpDriverObject;
SYS_STATUS      pmpStatus;

// Populate the initialization structure
init.inputBuffer = PMP_INPUT_BUFFER_TTL;
init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS |
PMP_PMA14_PORT;
init.ports.readWriteStrobe = PORT_ENABLE;
init.ports.writeEnableStrobe = PORT_ENABLE;
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.pmpID             = PMP_ID_0;
init.stopInIdle        = false;
init.muxMode           = PMP_MUX_NONE;

DRV_PMP_Reinitialize(pmpDriverObject, (SYS_MODULE_INIT*)&init);

pmpStatus = DRV_PMP_Status(pmpDriverObject);
if (SYS_STATUS_BUSY == pmpStatus)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= pmpStatus)
{
    // Handle error
}

```

5.1.7.7.1.4 DRV_PMP_Status Function**C**

```

SYS_STATUS DRV_PMP_Status(
    const SYS_MODULE_OBJ pmpDriverObject
);

```

Description

This function provides the current status of the PMP driver module.

Preconditions

The [DRV_PMP_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
pmpDriverObject	Driver object handle, returned from the DRV_PMP_Initialize routine

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to [accept](#) new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than `SYS_STATUS_ERROR` is also an error state.

`SYS_MODULE_DEINITIALIZED` - Indicates that the driver has been deinitialized

This value is less than `SYS_STATUS_ERROR`.

This operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns `SYS_STATUS_BUSY`, a previous operation has not yet completed. Once the status operation returns `SYS_STATUS_READY`, any previous operations have completed.

The value of `SYS_STATUS_ERROR` is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Example

```
SYS_MODULE_OBJ    pmpDriverObject;    // Returned from DRV_PMP_Initialize
SYS_STATUS        status;

status = DRV_PMP_Status(pmpDriverObject);
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

5.1.7.7.1.5 DRV_PMP_Tasks Function

C

```
void DRV_PMP_Tasks(
    SYS_MODULE_OBJ pmpDriverObject
);
```

Description

This function is used to maintain the queue and execute the tasks stored in the queue. It resides in the ISR of the PMP for interrupt-driven implementations.

Preconditions

The [DRV_PMP_Initialize](#) function must have been called for the specified PMP driver instance.

Parameters

Parameters	Description
pmpDriverObject	Object handle for the specified driver instance (returned from DRV_PMP_Initialize)

Returns

None.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)) or by the appropriate raw ISR.

This function may excute in an ISR context and will never block or access any resources that may cause it to block.

Example

```
SYS_MODULE_OBJ    pmpDriverObject;    // Returned from DRV_PMP_Initialize

while (true)
{
```

```
    DRV_PMP_Tasks (pmpDriverObject);  
  
    // Do other tasks  
}
```

5.1.7.7.2 Client Interaction Functions

5.1.7.7.2.1 DRV_PMP_ClientStatus Function

C

```
DRV_PMP_CLIENT_STATUS DRV_PMP_ClientStatus(  
    DRV_HANDLE hClient  
);
```

Description

This function gets the client-specific status of the PMP driver associated with the specified handle.

Preconditions

The [DRV_PMP_Initialize](#) routine must have been called.

[DRV_PMP_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
hClient	A valid open-instance handle, returned from the driver's open routine

Returns

A [DRV_PMP_CLIENT_STATUS](#) value describing the current status of the driver.

Remarks

This function will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE hClient; // Returned from DRV_PMP_Open  
DRV_PMP_CLIENT_STATUS pmpClientStatus;  
  
pmpClientStatus = DRV_PMP_ClientStatus(hClient);  
if(DRV_PMP_CLIENT_STATUS_ERROR >= pmpClientStatus)  
{  
    // Handle the error  
}
```

5.1.7.7.2.2 DRV_PMP_Close Function

C

```
void DRV_PMP_Close(  
    const DRV_HANDLE hClient  
);
```

Description

This function closes an opened instance of the PMP driver, invalidating the handle.

Preconditions

The [DRV_PMP_Initialize](#) routine must have been called for the specified PMP driver instance.

[DRV_PMP_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
hClient	A valid open instance handle, returned from the driver's open routine

Returns

None

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling [DRV_PMP_Open](#) before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the [DRV_PMP_Status](#) operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Example

```
DRV_HANDLE hClient; // Returned from DRV_PMP_Open

DRV_PMP_Close(hClient);
```

5.1.7.7.2.3 DRV_PMP_ModeConfig Function**C**

```
void DRV_PMP_ModeConfig(
    DRV_HANDLE hClient,
    DRV_PMP_MODE_CONFIG config
);
```

Description

This function configures the modes for client in which it wants to operate. Different master-slave modes, 8/16 data bits selection, address increment/decrement, interrupt mode, wait states, etc., can be configured through this function.

Preconditions

Function [DRV_PMP_Initialize](#) must have been called. [DRV_PMP_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
hClient	Client handle obtained from DRV_PMP_Open API
config	Structure which will have all the required PMP modes configuration

Returns

None.

Remarks

This function will NEVER block waiting for hardware. If this API is called more than once for a particular client handle, previous config setting of that client will be overwritten.

Example

```
DRV_HANDLE hClient;
DRV_PMP_MODE_CONFIG config;
```



```

config.chipSelect = PMCS1_AND_PMCS2_AS_CHIP_SELECT;
config.endianMode = LITTLE_ENDIAN;
config.incrementMode = PMP_ADDRESS_AUTO_INCREMENT;
config.intMode = PMP_INTERRUPT_NONE;
config.pmpMode = PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT;
config.portSize = PMP_DATA_SIZE_8_BITS;
config.waitStates.dataHoldWait = PMP_DATA_HOLD_2;
config.waitStates.dataWait = PMP_DATA_WAIT_THREE;
config.waitStates.strobeWait = PMP_STROBE_WAIT_5;

DRV_PMP_ModeConfig ( hClient, config );

```

5.1.7.7.2.4 DRV_PMP_Open Function

C

```

DRV_HANDLE DRV_PMP_Open(
    const SYS_MODULE_INDEX index,
    const DRV_IO_INTENT intent
);

```

Description

This function opens the specified PMP driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Preconditions

The [DRV_PMP_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is [DRV_HANDLE_INVALID](#).

Remarks

The handle returned is valid until the [DRV_PMP_Close](#) routine is called.

This function will NEVER block waiting for hardware.

If the [DRV_IO_INTENT_BLOCKING](#) is requested and the driver was built appropriately to support blocking behavior, other client-level operations may block waiting on hardware until they are complete.

If [DRV_IO_INTENT_NON_BLOCKING](#) is requested the driver client can call the [DRV_PMP_ClientStatus](#) operation to find out when the module is in the ready state.

If the requested intent flags are not supported, the routine will return [DRV_HANDLE_INVALID](#).

Example

```

DRV_HANDLE hClient;

hClient = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == hClient)
{
    // Unable to open the driver
}

```

5.1.7.7.2.5 DRV_PMP_Read Function

C

```
QUEUE_ELEMENT_OBJECT* DRV_PMP_Read(
    DRV_HANDLE hClient,
    uint32_t address,
    uint16_t* buffer,
    uint32_t nBytes
);
```

Description

This function reads the given number of data bytes from the given address of the external device to the MCU buffer through the selected PMP instance. This function should be used for all the master and slave modes. Proper configuration should be done using [DRV_PMP_ModeConfig](#) before calling this function.

Preconditions

The [DRV_PMP_Initialize](#) routine must have been called. [DRV_PMP_Open](#) must have been called to obtain a valid opened device handle. [DRV_PMP_ModeConfig](#) must have been called to configure the desired mode

Parameters

Parameters	Description
hClient	A valid open-instance handle, returned from the driver's open routine
address	Starting address of the slave device from where data has to be read. It does not have any significance for legacy slave mode and buffer mode. In PMP enhanced slave mode i.e. addressable buffer slave mode, this parameter should be the buffer number to be used.
buffer	Pointer to the buffer into which the data read through the PMP instance will be placed. Even if only one word has to be transferred, pointer should be used.
nBytes	Number of bytes that need to be read through the PMP instance

Returns

Returns the position number of the queue, where the data element was stored. Returns '0' when there is no place in the queue to store the data.

Example

```
DRV_HANDLE hClient; // Returned from DRV_PMP_Open
uint32_t deviceAddress;
uint32_t nBytes;
unsigned char myBuffer[nBytes];
uint32_t transferID;

transferID = DRV_PMP_Read ( hClient, deviceAddress, &myBuffer, nBytes);
```

5.1.7.7.2.6 DRV_PMP_Write Function

C

```
QUEUE_ELEMENT_OBJECT* DRV_PMP_Write(
    DRV_HANDLE* hClient,
    bool address,
    uint32_t * buffer,
    uint32_t nBytes,
    uint32_t repeatCount
);
```

Description

This function transfer the given number of data bytes from the MCU buffer location to the defined address of the external device

through the selected PMP instance. It repeats the operation n (=repeatCount) number of times as well. This function should be used for all the master and slave modes. Proper configuration should be done using [DRV_PMP_ModeConfig](#) before calling this function.

Preconditions

The [DRV_PMP_Initialize](#) routine must have been called. [DRV_PMP_Open](#) must have been called to obtain a valid opened device handle. [DRV_PMP_ModeConfig](#) must have been called to configure the desired mode.

Parameters

Parameters	Description
hClient	A valid open-instance handle, returned from the driver's open routine
address	Starting address of the slave device where data has to be written. It does not have any significance for legacy slave mode and buffer mode. In PMP enhanced slave mode (i.e., addressable buffer slave mode), this parameter should be the buffer number to be used.
buffer	Pointer to MCU Buffer from which the data will be written through the PMP instance. even if only one word has to be tranferred, pointer should be used.
nBytes	Total number of bytes that need to be written through the PMP instance
repeatCount	Number of times the data set (nBytes of data) to be repetedly written. This value should be 0 if user does not want any repetation. If repeatCount is greater than 0, then after writing every nBytes of data, the buffer starts pointing to its first element. Ideally, PMP Address should be in auto increment/decrement mode for repeatCount greater than 0.

Returns

Returns a 32-bit ID with which status of the transfer can be checked later. Returns '0' when there is no place in the queue to store the data.

Example

```
DRV_HANDLE hClient; // Returned from DRV_PMP_Open
uint32_t deviceAddress;
uint32_t nBytes;
unsigned char myBuffer[nBytes];
uint32_t repeatCount;
uint32_t transferID;

transferID = DRV_PMP_MasterWrite ( hClient, deviceAddress, &myBuffer, nBytes, repeatCount);
```

5.1.7.7.3 Client Transfer Functions

5.1.7.7.3.1 DRV_PMP_TransferStatus Function

C

```
DRV_PMP_TRANSFER_STATUS DRV_PMP_TransferStatus(
    QUEUE_ELEMENT_OBJECT* queueObject
);
```

Description

This function returns the status of a particular transfer whose ID has been specified as input.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
seqID	A valid ID returned from read/write transfer functions

Returns

A [DRV_PMP_TRANSFER_STATUS](#) value describing the current status of the transfer.

Example

```
uint32_8 seqID;
DRV_PMP_TRANSFER_STATUS transferStatus;

transferStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, seqID);
```

5.1.7.7.4 Data Types and Constants

5.1.7.7.4.1 DRV_PMP_INDEX_COUNT Macro

C

```
#define DRV_PMP_INDEX_COUNT _PMP_EXISTS
```

Description

PMP Driver Module Index Count

This constant identifies the number of valid PMP driver indices.

Remarks

The value of "_PMP_EXISTS" is derived from device-specific header files defined as part of the peripheral libraries.

5.1.7.7.4.2 DRV_PMP_CHIPX_STROBE_MODE Enumeration

C

```
typedef enum {
    PMP_RW_STROBE_WITH_ENABLE_STROBE,
    PMP_READ_AND_WRITE_STROBES
} DRV_PMP_CHIPX_STROBE_MODE;
```

Description

PMP writeEnable/ReadWrite strobes

This enumeration provides ReadWrite/WriteEnable Strobe definitions.

Members

Members	Description
PMP_RW_STROBE_WITH_ENABLE_STROBE	One strobe for read/write and another for enable
PMP_READ_AND_WRITE_STROBES	Separate strobes for read and write operations

5.1.7.7.4.3 DRV_PMP_CLIENT_STATUS Enumeration

C

```
typedef enum {
    DRV_PMP_CLIENT_STATUS_INVALID,
    PMP_CLIENT_STATUS_CLOSED,
    DRV_PMP_CLIENT_STATUS_OPEN
} DRV_PMP_CLIENT_STATUS;
```

Description

PMP Client Status

This enumeration provides various client status possibilities.

5.1.7.7.4.4 DRV_PMP_ENDIAN_MODE Enumeration

C

```
typedef enum {  
    LITTLE,  
    BIG  
} DRV_PMP_ENDIAN_MODE;
```

Description

PMP Endian modes

This enumeration holds the endian configuration options.

Members

Members	Description
LITTLE	Little Endian
BIG	Big Endian

5.1.7.7.4.5 DRV_PMP_INDEX Enumeration

C

```
typedef enum {  
    DRV_PMP_INDEX_0,  
    DRV_PMP_INDEX_1  
} DRV_PMP_INDEX;
```

Description

PMP Driver Module Index Numbers

These constants provide PMP driver index definitions.

Members

Members	Description
DRV_PMP_INDEX_0	First PMP instance
DRV_PMP_INDEX_1	Second PMP instance (not available for now)

Remarks

These values should be passed into the [DRV_PMP_Initialize](#) and [DRV_PMP_Open](#) functions to identify the driver instance in use.

5.1.7.7.4.6 DRV_PMP_INIT Structure

C

```
typedef struct {  
    SYS_MODULE_INIT moduleInit;  
    PMP_MODULE_ID pmpID;  
    bool stopInIdle;  
    PMP_MUX_MODE muxMode;  
    PMP_INPUT_BUFFER_TYPE inputBuffer;  
    DRV_PMP_POLARITY_OBJECT polarity;  
    DRV_PMP_PORTS ports;  
} DRV_PMP_INIT;
```

Description

PMP Driver Initialize Data

This data type defines data required to initialize or reinitialize the PMP driver.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	module power state info
PMP_MODULE_ID pmpID;	module PLIB ID
bool stopInIdle;	Stop in Idle enable
PMP_MUX_MODE muxMode;	MUX mode
PMP_INPUT_BUFFER_TYPE inputBuffer;	Input buffer type to be used
DRV_PMP_POLARITY_OBJECT polarity;	Polarity settings
DRV_PMP_PORTS ports;	PMP port settings

Remarks

Not all the initialization features are available for all devices.

5.1.7.7.4.7 DRV_PMP_MODE_CONFIG Structure

C

```
typedef struct {
    PMP_OPERATION_MODE pmpMode;
    PMP_INTERRUPT_MODE intMode;
    PMP_INCREMENT_MODE incrementMode;
    DRV_PMP_ENDIAN_MODE endianMode;
    PMP_DATA_SIZE portSize;
    DRV_PMP_WAIT_STATES waitStates;
    PMP_CHIPSELECT_FUNCTION chipSelect;
} DRV_PMP_MODE_CONFIG;
```

Description

PMP modes configuration

This data type controls the configuration of PMP modes.

Members

Members	Description
PMP_OPERATION_MODE pmpMode;	PMP Usage Mode Type
PMP_INTERRUPT_MODE intMode;	Interrupt mode
PMP_INCREMENT_MODE incrementMode;	should be appropriately selected based on read/write requirements and operation mode setting */ address/buffer increment mode
DRV_PMP_ENDIAN_MODE endianMode;	it does not have any significance in PMP slave mode or 8bit data mode */ Endian modes
PMP_DATA_SIZE portSize;	Data Port Size
DRV_PMP_WAIT_STATES waitStates;	Wait states
PMP_CHIPSELECT_FUNCTION chipSelect;	use this when plib is fixed

5.1.7.7.4.8 DRV_PMP_POLARITY_OBJECT Structure

C

```
typedef struct {
    PMP_POLARITY_LEVEL addressLatchPolarity;
    PMP_POLARITY_LEVEL byteEnablePolarity;
    PMP_POLARITY_LEVEL rwStrobePolarity;
    PMP_POLARITY_LEVEL writeEnableStrobePolarity;
```

```

    PMP_POLARITY_LEVEL chipselect1Polarity;
    PMP_POLARITY_LEVEL chipselect2Polarity;
} DRV_PMP_POLARITY_OBJECT;

```

Description

PMP polarity object

This structure holds the polarities of different entities to be configured.

Members

Members	Description
PMP_POLARITY_LEVEL addressLatchPolarity;	Address latch polarity
PMP_POLARITY_LEVEL byteEnablePolarity;	ByteEnable port polarity
PMP_POLARITY_LEVEL rwStrobePolarity;	Read/Write strobe polarity
PMP_POLARITY_LEVEL writeEnableStrobePolarity;	Write/Enable strobe polarity
PMP_POLARITY_LEVEL chipselect1Polarity;	ChipSelect-1 Polarity
PMP_POLARITY_LEVEL chipselect2Polarity;	chipSelect-2 Polarity

5.1.7.7.4.9 DRV_PMP_PORT_CONTROL Enumeration

C

```

typedef enum {
    PORT_ENABLE,
    PORT_DISABLE
} DRV_PMP_PORT_CONTROL;

```

Description

PMP port enable/disable.

This enumeration provides port enable/disable values.

Members

Members	Description
PORT_ENABLE	Enable the given port
PORT_DISABLE	Disable the given port

5.1.7.7.4.10 DRV_PMP_PORTS Structure

C

```

typedef struct {
    PMP_ADDRESS_PORT addressPortsMask;
    PMP_PMBE_PORT byteEnablePort;
    DRV_PMP_PORT_CONTROL readWriteStrobe;
    DRV_PMP_PORT_CONTROL writeEnableStrobe;
} DRV_PMP_PORTS;

```

Description

PMP Ports

This structure holds the ports (including the address ports) to be configured by the application to function as general purpose I/O (GPIO) or part of the PMP.

Members

Members	Description
PMP_ADDRESS_PORT addressPortsMask;	User needs to put the address lines which he wants to use in OR'd fashion */ Address ports
PMP_PMBE_PORT byteEnablePort;	Byte enable ports
DRV_PMP_PORT_CONTROL readWriteStrobe;	READ/WRITE Strobe PORT
DRV_PMP_PORT_CONTROL writeEnableStrobe;	WRITE/ENABLE strobe port

5.1.7.7.4.11 DRV_PMP_QUEUE_ELEMENT_OBJ Type**C**

```
typedef struct _DRV_PMP_QUEUE_ELEMENT_OBJ DRV_PMP_QUEUE_ELEMENT_OBJ;
```

Description

PMP Driver Queue Element Object

This defines the object structure for each queue element of PMP. This object gets created for every Read/Write operations APIs.

Remarks

None

5.1.7.7.4.12 DRV_PMP_TRANSFER_STATUS Enumeration**C**

```
typedef enum {
    MASTER_8BIT_TRANSFER_IN_PROGRESS = PMP_DATA_SIZE_8_BITS,
    MASTER_16BIT_TRANSFER_IN_PROGRESS = PMP_DATA_SIZE_16_BITS,
    MASTER_8BIT_BUFFER_IN_PROGRESS,
    MASTER_16BIT_BUFFER_IN_PROGRESS,
    MASTER_8BIT_BUFFER_CONTINUE,
    QUEUED_BUT_PMP_TRANSFER_NOT_STARTED,
    PMP_TRANSFER_FINISHED
} DRV_PMP_TRANSFER_STATUS;
```

Description

Queue Element Transfer Status

This enumeration defines the PMP transfer status.

5.1.7.7.4.13 DRV_PMP_WAIT_STATES Structure**C**

```
typedef struct {
    PMP_DATA_HOLD_STATES dataHoldWait;
    PMP_STROBE_WAIT_STATES strobeWait;
    PMP_DATA_WAIT_STATES dataWait;
} DRV_PMP_WAIT_STATES;
```

Description

PMP wait states object

This structure holds the different wait states to be configured. Refer to the PMP PLIB help document for the possible values and meaning of the different wait states.

Members

Members	Description
PMP_DATA_HOLD_STATES dataHoldWait;	data hold wait states
PMP_STROBE_WAIT_STATES strobeWait;	read/write strobe wait states
PMP_DATA_WAIT_STATES dataWait;	data wait strobe wait sates

5.1.7.7.4.14 QUEUE_ELEMENT_OBJECT Type**C**

```
typedef struct _QUEUE_ELEMENT_OBJECT QUEUE_ELEMENT_OBJECT;
```

Description

Queue Element Object

This defines the structure required for maintaining the queue element.

Remarks

None

5.1.7.7.4.15 MAX_NONBUFFERED_BYTE_COUNT Macro**C**

```
#define MAX_NONBUFFERED_BYTE_COUNT 4 //After this number the PMP transfer should be polled to  
guarantee data transfer
```

Description

After this number the PMP transfer should be polled to guarantee data transfer

5.1.7.7.4.16 DRV_PMP_TRANSFER_TYPE Enumeration**C**

```
typedef enum {  
    ADDRESS,  
    READ,  
    WRITE,  
    BUFFERED_WRITE  
} DRV_PMP_TRANSFER_TYPE;
```

Description

This is type DRV_PMP_TRANSFER_TYPE.

Members

Members	Description
ADDRESS	PMP Address needs to be updated
READ	PMP Read Transfer
WRITE	PMP Write Transfer
BUFFERED_WRITE	PMP Array Write Transfer

5.1.7.7.5 Miscellaneous Functions

5.1.7.7.5.1 DRV_PMP_VersionGet Function

C

```
unsigned int DRV_PMP_VersionGet(  
    const SYS_MODULE_INDEX drvIndex  
);
```

Description

This function gets the PMP driver version. The version is encoded as major * 10000 + minor * 100 + patch. The stringed version can be obtained using [DRV_PMP_VersionStrGet\(\)](#).

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to get the version for

Returns

Current driver version in numerical format.

Remarks

None.

Example

```
unsigned int version;  
  
version = DRV_PMP_VersionGet( DRV_PMP_INDEX_0 );  
  
if(version < 110200)  
{  
    // Do Something  
}
```

5.1.7.7.5.2 DRV_PMP_VersionStrGet Function

C

```
char * DRV_PMP_VersionStrGet(  
    const SYS_MODULE_INDEX drvIndex  
);
```

Description

This function gets the PMP driver version. The version is returned as major.minor.path[type], where type is optional. The numerical version can be obtained using [DRV_PMP_VersionGet\(\)](#).

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to get the version for

Returns

Current PMP driver version in the string format.

Remarks

None.

Example

```
char *version;

version = DRV_PMP_VersionStrGet( DRV_PMP_INDEX_0 );

printf("%s", version);
```

5.1.7.8 Files

Files

Name	Description
drv_pmp.h	Parallel Master Port (PMP) device driver interface file.
drv_pmp_config.h	PMP driver configuration definitions template

Description

5.1.7.8.1 drv_pmp.h

PMP Device Driver Interface

The PMP device driver provides a simple interface to manage the Parallel Master and Slave ports. This file defines the interface definitions and prototypes for the PMP driver.

Enumerations

	Name	Description
	DRV_PMP_CHIPX_STROBE_MODE	PMP writeEnable/ReadWrite strobes.
	DRV_PMP_CLIENT_STATUS	PMP client status definitions.
	DRV_PMP_ENDIAN_MODE	PMP endian modes.
	DRV_PMP_INDEX	PMP driver index definitions.
	DRV_PMP_PORT_CONTROL	PMP port enable/disable definitions.
	DRV_PMP_TRANSFER_STATUS	Defines the PMP transfer status.
	DRV_PMP_TRANSFER_TYPE	This is type DRV_PMP_TRANSFER_TYPE.

Functions

	Name	Description
	DRV_PMP_ClientStatus	Gets the current client-specific status of the PMP driver.
	DRV_PMP_Close	Closes an opened instance of the PMP driver.
	DRV_PMP_Deinitialize	Deinitializes the specified instance of the PMP driver module
	DRV_PMP_Initialize	Initializes the PMP driver.
	DRV_PMP_ModeConfig	Configure the PMP modes.
	DRV_PMP_Open	Opens the specified PMP driver instance and returns a handle to it.
	DRV_PMP_Read	Read the data from external device.
	DRV_PMP_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings.
	DRV_PMP_Status	Provides the current status of the PMP driver module.
	DRV_PMP_Tasks	Maintains the driver's state machine and implements its ISR.

	DRV_PMP_TransferStatus	Returns the transfer status.
	DRV_PMP_VersionGet	Gets the PMP driver version in numerical format.
	DRV_PMP_VersionStrGet	Gets the PMP driver version in string format.
	DRV_PMP_Write	Transfers the data from the MCU to the external device.

Macros

	Name	Description
	DRV_PMP_INDEX_COUNT	Number of valid PMP driver indices.
	MAX_NONBUFFERED_BYTE_COUNT	After this number the PMP transfer should be polled to gurantee data trasnfer

Structures

	Name	Description
	DRV_PMP_INIT	Defines the PMP driver initialization data.
	DRV_PMP_MODE_CONFIG	PMP modes configuration.
	DRV_PMP_POLARITY_OBJECT	PMP polarity object.
	DRV_PMP_PORTS	PMP port configuration.
	DRV_PMP_WAIT_STATES	PMP wait states object.

Types

	Name	Description
	DRV_PMP_QUEUE_ELEMENT_OBJ	Defines the object for PMP queue element.
	QUEUE_ELEMENT_OBJECT	Defines the structure required for maintaining the queue element.

Company

Microchip Technology Inc.

FileName: drv_pmp.h

5.1.7.8.2 drv_pmp_config.h

PMP Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

Macros

	Name	Description
	DRV_PMP_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_PMP_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_PMP_QUEUE_SIZE	PMP queue size for different instances.

File Name

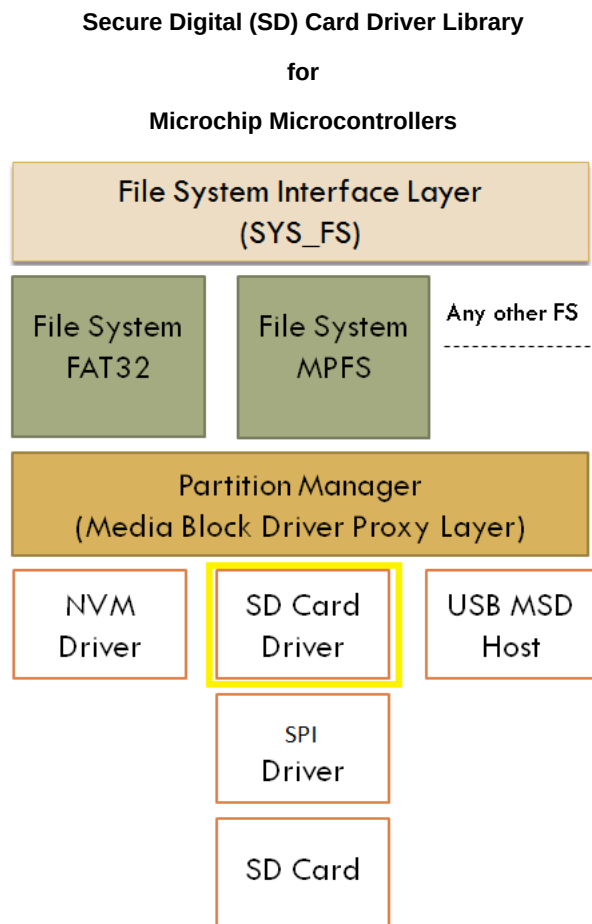
drv_pmp_config_template.h

Company

Microchip Technology Inc.

5.1.8 Secure Digital (SD) Card Driver Library

5.1.8.1 Introduction



The SD Card driver provides the necessary interfaces to interact with an SD card. It provides the necessary abstraction for the higher layer.

Description

A SD Card is a non-volatile memory (Flash memory) card designed to provide high-capacity memory in a small size. Its applications are like digital video camcorders, digital cameras, handheld computers, audio players and mobile phones.

5.1.8.2 Release Notes

MPLAB Harmony Version: v0.70b **SD Card Library Version :** **Release Date:** 18Nov2013

New This Release:

- Non-blocking operation
- Layered architecture
- Interoperability

Known Issues:

- Tested only with SDHC cards (Card memory more than 2 GB)
- CARD detection mechanism is not so fine-tuned
- Card maximum Frequency detection is not automatic. Different cards will have specific frequency of operation and that needs to be set in `system_config.h`
- Not tested the drivers operation with multiple SD cards
- Timeout and driver reset on failure case is not implemented

5.1.8.3 SW License Agreement

(c) 2013 Microchip Technology Inc. Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.8.4 Using the Library

This topic describes the basic architecture of the SD Card Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_sdcard.h](#)

The interface to the SD Card Driver library is defined in the "drv_sdcard.h" header file. This file is included by the "drv.h" file. Any C language source (.c) file that uses the SD Card Driver library should include "drv.h".

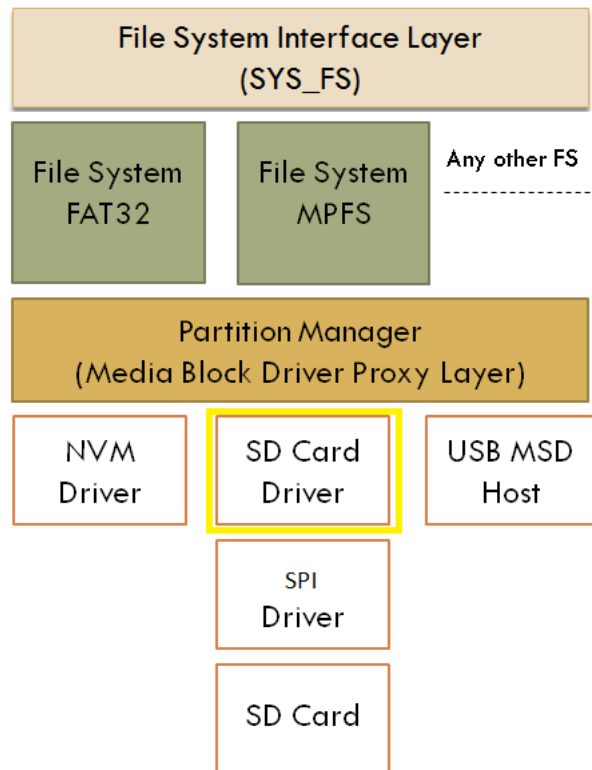
Library File:

The SD Card Driver library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the Driver interacts with the framework.

5.1.8.4.1 Abstraction Model

SD Card Driver Software Abstraction Block Diagram



- SD Card driver comes in the layer below the Partition Manager in MPLAB Harmony file system architecture.
- It uses the SPI driver to interact with SD card.

5.1.8.4.2 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the SD Card module.

Library Interface Section	Description
System Level Functions	Includes functions for initialize the module.
Client Level Functions	Functions to open and close a client.
Operation Functions	Functions for read and write operations
Module Information Functions	Functions for information about the module.
Version Information Functions	Functions to get the software version.

5.1.8.4.3 How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

5.1.8.5 Configuring the Library

The configuration of the SD Card driver is based on the file `sys_config.h`.

This header file contains the configuration selection for the SD Card driver. Based on the selections made, the SD Card driver will support or not support selected features. These configuration settings will apply to all instances of the SD Card.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.8.6 Building the Library

This section lists the files that are available in the `\src` of the SD Card driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.1.8.7 Library Interface

Data Types and Constants

	Name	Description
	DRV_SDCARD_INDEX_0	SD Card driver index definitions
	DRV_SDCARD_INDEX_COUNT	Number of valid SD Card driver indices
	DRV_SDCARD_BUFFER_HANDLE	Handle to a buffer added to the process queue.
	DRV_SDCARD_CLIENT_STATUS	Identifies the client-specific status of the SD card driver
	DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device
	SYS_MEDIA_EVENT	Defines the buffer status
	SDCARD_MAX_LIMIT	Maximum allowed SD card instances
	DRV_SDCARD_INDEX_1	This is macro <code>DRV_SDCARD_INDEX_1</code> .
	DRV_SDCARD_INDEX_2	This is macro <code>DRV_SDCARD_INDEX_2</code> .
	DRV_SDCARD_INDEX_3	This is macro <code>DRV_SDCARD_INDEX_3</code> .

Client Level Functions

	Name	Description
	DRV_SDCARD_ClientStatus	Gets current client-specific status the SD Card driver
	DRV_SDCARD_Close	Closes an opened-instance of the SD Card driver
	DRV_SDCARD_Open	Opens the specified SD Card driver instance and returns a handle to it

Module Information Functions

	Name	Description
	DRV_SDCARD_SectorsCountGet	Gets the number of sectors present in the selected SD card device.
	DRV_SDCARD_SectorSizeGet	Gets the sector size of the selected SD card device.
	DRV_SDCARD_WriteProtectionIsEnabled	Gets the status of write protection.

Operation Functions

	Name	Description
	DRV_SDCARD_SectorRead	Reads data from the sectors of the SD card.
	DRV_SDCARD_SectorWrite	Writes sector(s) of data to an SD card.

	DRV_SDCARD_BufferStatusGet	Gets the status of an an SD card operation (read/write)
	DRV_SDCARD_MediaStatusGet	Gets the status of the device.

System Level Functions

	Name	Description
	DRV_SDCARD_Deinitialize	Deinitializes the specified instance of the SD Card driver module
	DRV_SDCARD_Initialize	Initializes the SD Card driver
	DRV_SDCARD_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings
	DRV_SDCARD_Status	Provides the current status of the SD Card driver module
	DRV_SDCARD_Tasks	Maintains the driver's state machine

Version Information Functions

	Name	Description
	DRV_SDCARD_VersionGet	Gets SD card driver version in numerical format.
	DRV_SDCARD_VersionStrGet	Gets SD card driver version in string format.

Description

This section describes the Application Programming Interface (API) functions of the SD Card Driver.

Refer to each section for a detailed description.

5.1.8.7.1 System Level Functions

5.1.8.7.1.1 DRV_SDCARD_Deinitialize Function

C

```
void DRV_SDCARD_Deinitialize(
    SYS_MODULE_OBJ object
);
```

Description

Deinitializes the specified instance of the SD Card driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Preconditions

Function [DRV_SDCARD_Initialize](#) must have been called before calling this routine and a valid [SYS_MODULE_OBJ](#) must have been returned.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SDCARD_Initialize routine.

Returns

None.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This routine will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the [DRV_SDCARD_Status](#) operation. The system has to use [DRV_SDCARD_Status](#) to find out when the module is

in the ready state.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_SDCARD_Initialize
SYS_STATUS        status;

DRV_SDCARD_Deinitialize(object);

status = DRV_SDCARD_Status(object);
if (SYS_MODULE_UNINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

5.1.8.7.1.2 DRV_SDCARD_Initialize Function

C

```
SYS_MODULE_OBJ DRV_SDCARD_Initialize(
    const SYS_MODULE_INDEX index,
    const SYS_MODULE_INIT * const init
);
```

Description

This routine initializes the SD Card driver, making it ready for clients to open and use it.

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver. This pointer may be null if no data is required because static overrides have been provided.

Returns

If successful, returns a valid handle to a driver object. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#). The returned object must be passed as argument to [DRV_SDCARD_Reinitialize](#), [DRV_SDCARD_Deinitialize](#), [DRV_SDCARD_TaskRead](#), [DRV_SDCARD_TaskWrite](#) and [DRV_SDCARD_Status](#) routines.

Remarks

This routine must be called before any other SD Card routine is called.

This routine should only be called once during system initialization unless [DRV_SDCARD_Deinitialize](#) is called to de-initialize the driver instance.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to re-initialize, it will be reported by the [DRV_SDCARD_Status](#) operation. The system must use [DRV_SDCARD_Status](#) to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this routine.

Example

```
DRV_SDCARD_INIT    init;
SYS_MODULE_OBJ     objectHandle;
```

```
// Populate the SD Card initialization structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.baudRate         = 100000;
init.spiId            = SPI_ID_1;

// Do something

objectHandle = DRV_SDCARD_Initialize(DRV_SDCARD_INDEX_0, (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

5.1.8.7.1.3 DRV_SDCARD_Reinitialize Function

C

```
void DRV_SDCARD_Reinitialize(
    SYS_MODULE_OBJ object,
    const SYS_MODULE_INIT * const init
);
```

Description

This routine reinitializes the driver and refreshes any associated hardware settings using the initialization data given, but it will not interrupt any ongoing operations.

Preconditions

Function [DRV_SDCARD_Initialize](#) must have been called before calling this routine and a valid [SYS_MODULE_OBJ](#) must have been returned.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SDCARD_Initialize routine
init	Pointer to the initialization data structure

Returns

None

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to re-initialize, it will be reported by the [DRV_SDCARD_Status](#) operation. The system must use [DRV_SDCARD_Status](#) to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this routine.

Example

```
DRV_SDCARD_INIT    init;
SYS_MODULE_OBJ     objectHandle;

// Populate the SD Card initialization structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.baudRate         = 100000;
init.spiId            = SPI_ID_1;
```

```
// Do something
```

5.1.8.7.1.4 DRV_SDCARD_Status Function

C

```
SYS_STATUS DRV_SDCARD_Status(  
    SYS_MODULE_OBJ object  
);
```

Description

This routine provides the current status of the SD Card driver module.

Preconditions

Function [DRV_SDCARD_Initialize](#) must have been called before calling this

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SDCARD_Initialize routine

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Note Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to [accept](#) new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Remarks

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been de-initialized

This value is less than SYS_STATUS_ERROR

The this operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This routine will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_SDCARD_Initialize  
SYS_STATUS        status;  
  
status = DRV_SDCARD_Status(object);  
else if (SYS_STATUS_ERROR >= status)  
{  
    // Handle error  
}
```

5.1.8.7.1.5 DRV_SDCARD_Tasks Function

C

```
void DRV_SDCARD_Tasks(  
    SYS_MODULE_OBJ object  
)  
;
```

Description

This routine is used to maintain the driver's internal state machine.

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called for the specified SDCARD driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SDCARD_Initialize)

Returns

None

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_SDCARD_Initialize  
  
while (true)  
{  
    DRV_SDCARD_Tasks (object);  
  
    // Do other tasks  
}
```

5.1.8.7.2 Client Level Functions

5.1.8.7.2.1 DRV_SDCARD_ClientStatus Function

C

```
DRV_SDCARD_CLIENT_STATUS DRV_SDCARD_ClientStatus(  
    DRV_HANDLE handle  
)  
;
```

Description

This routine gets the client-specific status of the SD Card driver associated with the given handle.

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called.

[DRV_SDCARD_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

A [DRV_SDCARD_CLIENT_STATUS](#) value describing the current status of the driver

Remarks

This routine will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE sdcardHandle; // Returned from DRV_SDCARD_Open
DRV_SDCARD_CLIENT_STATUS sdcardClientStatus;

sdcardClientStatus = DRV_SDCARD_ClientStatus ( sdcardHandle );
if ( DRV_SDCARD_CLIENT_STATUS_ERROR >= sdcardClientStatus )
{
    // Handle the error
}
```

5.1.8.7.2.2 DRV_SDCARD_Close Function**C**

```
void DRV_SDCARD_Close(
    DRV_HANDLE handle
);
```

Description

This routine closes an opened-instance of the SD Card driver, invalidating the handle.

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called for the specified SD Card driver instance.

[DRV_SDCARD_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

None

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling [DRV_SDCARD_Open](#) before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the [DRV_SDCARD_Status](#) operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Example

```
DRV_HANDLE handle; // Returned from DRV_SDCARD_Open
```

```
DRV_SDCARD_Close ( handle );
```

5.1.8.7.2.3 DRV_SDCARD_Open Function

C

```
DRV_HANDLE DRV_SDCARD_Open(
    const SYS_MODULE_INDEX drvIndex,
    const DRV_IO_INTENT intent
);
```

Description

This routine opens the specified SD Card driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Preconditions

Function [DRV_SDCARD_Initialize](#) must have been called before calling this function.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "or'd" together to indicate the intended use of the driver

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is [DRV_HANDLE_INVALID](#).

Remarks

The handle returned is valid until the [DRV_SDCARD_Close](#) routine is called.

This routine will NEVER block waiting for hardware.

If the DRV_IO_INTENT_BLOCKING is requested and the driver was built appropriately to support blocking behavior, then other client-level operations may block waiting on hardware until they are complete.

If DRV_IO_INTENT_NON_BLOCKING is requested the driver client can call the [DRV_SDCARD_ClientStatus](#) operation to find out when the module is in the ready state.

If the requested intent flags are not supported, the routine will return [DRV_HANDLE_INVALID](#).

Example

```
DRV_HANDLE handle;

handle = DRV_SDCARD_Open ( DRV_SDCARD_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

if ( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

5.1.8.7.3 Operation Functions

5.1.8.7.3.1 DRV_SDCARD_SectorRead Function

C

```
SYS_FS_MEDIA_BUFFER_HANDLE DRV_SDCARD_SectorRead(
```

```

    DRV_HANDLE handle,
    uint8_t * buffer,
    uint32_t sector_addr,
    uint32_t sectorCount
);

```

Description

This function reads data from the sectors of the SD card starting from the sector address and stores it in the location pointed by 'buffer'.

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called.

[DRV_SDCARD_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
sector_addr	The address of the sector on the card.
sectorCount	Number of sectors to be read.
buffer	The buffer where the retrieved data will be stored. If buffer is NULL, do not store the data anywhere.

Returns

SYS_FS_MEDIA_BUFFER_HANDLE - Buffer handle.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Example

```

SYS_FS_MEDIA_BUFFER_HANDLE bufferHandle;

bufferHandle = DRV_SDCARD_SectorRead ( handle,
                                        , readData //buffer
                                        , 20 // Sector
                                        , 1 // Number of Sectors
                                        );

```

5.1.8.7.3.2 DRV_SDCARD_SectorWrite Function**C**

```

SYS_FS_MEDIA_BUFFER_HANDLE DRV_SDCARD_SectorWrite(
    DRV_HANDLE handle,
    uint32_t sector_addr,
    uint8_t * buffer,
    uint32_t sectorCount
);

```

Description

This function writes sector(s) of data (512 bytes) of data from the location pointed to by 'buffer' to the specified sector of the SD card.

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called.

[DRV_SDCARD_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
sector_addr	The address of the sector on the card.
buffer	The buffer with the data to write.

Returns

SYS_FS_MEDIA_BUFFER_HANDLE - Buffer handle.

Remarks

The card expects the address field in the command packet to be a byte address. The sector_addr value is converted to a byte address by shifting it left nine times (multiplying by 512).

Example

```
SYS_FS_MEDIA_BUFFER_HANDLE bufferHandle;  
  
bufferHandle = DRV_SDCARD_SectorWrite ( handle,  
                                         , readData //buffer  
                                         , 20 // Sector  
                                         , 1 // Number of Sectors  
                                         );
```

5.1.8.7.3.3 DRV_SDCARD_BufferStatusGet Function**C**

```
SYS_FS_MEDIA_BUFFER_STATUS DRV_SDCARD_BufferStatusGet(  
    DRV_HANDLE handle,  
    SYS_FS_MEDIA_BUFFER_HANDLE bufferHandle  
);
```

Description

This function gets the status of an SD card operation (read/write). To be called only after a read or write is scheduled.

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called.

[DRV_SDCARD_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
bufferHandle	Handle returned by a 'sector write' or a 'sector read' function.

Returns

SYS_FS_MEDIA_BUFFER_STATUS - Buffer status.

Remarks

None.

Example

```
int globalState = 0;  
SYS_FS_MEDIA_BUFFER_HANDLE bufferHandle;  
  
switch (globalState)  
{  
    case 0:
```

```

        bufferHandle = SYS_FS_MEDIA_MANAGER_SectorWrite ( handle,
                                                         , readData //buffer
                                                         , 20 // Sector
                                                         , 1 // Number of Sectors
                                                         );

        globalState++;
        break;
    case 1:
        if ( SYS_FS_MEDIA_MANAGER_BufferStatusGet(handle, bufferHandle) ==
            SYS_FS_MEDIA_BUFFER_COMPLETED)
        {
            //Write complete
        }
        break;

```

5.1.8.7.3.4 DRV_SDCARD_MediaStatusGet Function

C

```

SYS_FS_MEDIA_STATUS DRV_SDCARD_MediaStatusGet(
    DRV_HANDLE handle
);

```

Description

This function gets the status of the device (attached/detached).

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called.

[DRV_SDCARD_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

SYS_FS_MEDIA_STATUS - Status of the device.

Remarks

None.

Example

```

DRV_HANDLE sdcardHandle; // Returned from DRV_SDCARD_Open

if( DRV_SDCARD_MediaStatusGet ( handle ) == SYS_FS_MEDIA_ATTACHED )
{
    //Device is attached
}

```

5.1.8.7.4 Module Information Functions

5.1.8.7.4.1 DRV_SDCARD_SectorsCountGet Function

C

```

uint32_t DRV_SDCARD_SectorsCountGet(
    DRV_HANDLE handle
);

```

Description

This function gets the number of sectors present in the selected SD card device.

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called.

[DRV_SDCARD_Open](#) must have been called to obtain a valid opened device handle.

SD Card must have been detected and initialized by the task routine.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

The number of a sectors in the selected SD Card.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE sdcardHandle; // Returned from DRV_SDCARD_Open
DRV_SDCARD_CLIENT_STATUS sdcardClientStatus;
bool status;
```

5.1.8.7.4.2 DRV_SDCARD_SectorSizeGet Function**C**

```
uint32_t DRV_SDCARD_SectorSizeGet(
    DRV_HANDLE handle
);
```

Description

This function gets the card's sector size.

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called.

[DRV_SDCARD_Open](#) must have been called to obtain a valid opened device handle.

SD Card must have been detected and initialized by the task routine.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

The size of a sector in the selected SD Card.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE sdcardHandle; // Returned from DRV_SDCARD_Open
DRV_SDCARD_CLIENT_STATUS sdcardClientStatus;
bool status;
```

5.1.8.7.4.3 DRV_SDCARD_WriteProtectionIsEnabled Function

C

```
bool DRV_SDCARD_WriteProtectionIsEnabled(  
    DRV_HANDLE handle  
);
```

Description

This function returns 'true' if the write protection for the card is enabled.

Preconditions

The [DRV_SDCARD_Initialize](#) routine must have been called.

[DRV_SDCARD_Open](#) must have been called to obtain a valid opened device handle.

SD Card must have been detected and initialized by the task routine.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

true - Write protection is enabled. false - Write protection is disabled.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE sdcardHandle; // Returned from DRV_SDCARD_Open  
bool writeProtectStatus;  
  
//Call DRV_SDCARD_Open  
writeProtectStatus = DRV_SDCARD_WriteProtectionIsEnabled(sdcardHandle);
```

5.1.8.7.5 Version Information Functions

5.1.8.7.5.1 DRV_SDCARD_VersionGet Function

C

```
unsigned int DRV_SDCARD_VersionGet(  
    const SYS_MODULE_INDEX drvIndex  
);
```

Description

This routine gets the SD card driver version. The version is encoded as major * 10000 + minor * 100 + patch. The stringized version can be obtained using [DRV_SDCARD_VersionStrGet\(\)](#)

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for.

Returns

Current driver version in numerical format.

Remarks

None.

Example

```
unsigned int version;

version = DRV_SDCARD_VersionGet( DRV_SDCARD_INDEX_1 );

if(version < 110200)
{
    // Do Something
}
```

5.1.8.7.5.2 DRV_SDCARD_VersionStrGet Function

C

```
char* DRV_SDCARD_VersionStrGet(
    const SYS_MODULE_INDEX drvIndex
);
```

Description

This routine gets the SD card driver version. The version is returned as major.minor.path[type], where type is optional. The numerical version can be obtained using [DRV_SDCARD_VersionGet\(\)](#)

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for.

Returns

Current SD card driver version in the string format.

Remarks

None.

Example

```
char *version;

version = DRV_SDCARD_VersionStrGet( DRV_SDCARD_INDEX_1 );

printf("%s", version);
```

5.1.8.7.6 Data Types and Constants

5.1.8.7.6.1 DRV_SDCARD_INDEX_0 Macro

C

```
#define DRV_SDCARD_INDEX_0 0
```

Description

SD Card Driver Module Index Numbers

These constants provide SD Card driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the [DRV_SDCARD_Initialize](#) and [DRV_SDCARD_Open](#) routines to identify the driver instance in use.

5.1.8.7.6.2 DRV_SDCARD_INDEX_COUNT Macro

C

```
#define DRV_SDCARD_INDEX_COUNT DRV_SDCARD_INDEX_MAX
```

Description

SD Card Driver Module Index Count

This constant identifies number of valid SD Card driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

5.1.8.7.6.3 DRV_SDCARD_BUFFER_HANDLE Type

C

```
typedef uintptr_t DRV_SDCARD_BUFFER_HANDLE;
```

Description

SD Card Buffer Handle

This handle identifies the a buffer in the queue(read or write).

5.1.8.7.6.4 DRV_SDCARD_CLIENT_STATUS Enumeration

C

```
typedef enum {
    DRV_SDCARD_CLIENT_STATUS_INVALID,
    DRV_SDCARD_CLIENT_STATUS_ERROR,
    DRV_SDCARD_CLIENT_STATUS_CLOSED,
    DRV_SDCARD_CLIENT_STATUS_BUSY,
    DRV_SDCARD_CLIENT_STATUS_READY,
    DRV_SDCARD_CLIENT_STATUS_STOPPED
} DRV_SDCARD_CLIENT_STATUS;
```

Description

SD Card Driver Client Status

This enumeration identifies the client-specific status of the SD card driver.

Members

Members	Description
DRV_SDCARD_CLIENT_STATUS_INVALID	Client in an invalid state

DRV_SDCARD_CLIENT_STATUS_ERROR	Unspecified error condition
DRV_SDCARD_CLIENT_STATUS_CLOSED	Client is not open
DRV_SDCARD_CLIENT_STATUS_BUSY	An operation is currently in progress
DRV_SDCARD_CLIENT_STATUS_READY	Up and running, no operations running
DRV_SDCARD_CLIENT_STATUS_STOPPED	SD Card stopped (not running), but ready to accept commands

Remarks

None.

5.1.8.7.6.5 DRV_SDCARD_INIT Type

C

```
typedef struct _DRV_SDCARD_INIT DRV_SDCARD_INIT;
```

Description

SD Card Device Driver Initialization Data

This structure contains all the data necessary to initialize the SD Card device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the [DRV_SDCARD_Initialize](#) routine.

5.1.8.7.6.6 SYS_MEDIA_EVENT Enumeration

C

```
typedef enum {
    SYS_MEDIA_EVENT_SDCARD_ATTACHED,
    SYS_MEDIA_EVENT_SDCARD_DETACHED
} SYS_MEDIA_EVENT;
```

Description

SD Card Buffer Status

Defines the states that an SD Card buffer can be in during its lifetime

Members

Members	Description
SYS_MEDIA_EVENT_SDCARD_ATTACHED	The media event is SD Card attach
SYS_MEDIA_EVENT_SDCARD_DETACHED	The media event is SD Card detach

Remarks

As buffers may have a limited life span, so too have the associated handles and status info. Once a buffer transfer is completed (and possibly notified and acknowledged) it will be discarded. The status of a discarded buffer is no longer maintained by the driver.

5.1.8.7.6.7 SDCARD_MAX_LIMIT Macro

C

```
#define SDCARD_MAX_LIMIT 2
```

Description

SD Card Driver Maximum allowed limit

This constant identifies number of valid SD Card driver indices.

Remarks

- This constant should be used in place of hard-coded numeric literals.
- This value is derived from part-specific header files defined as part of the peripheral libraries.

5.1.8.7.6.8 DRV_SDCARD_INDEX_1 Macro

C

```
#define DRV_SDCARD_INDEX_1 1
```

Description

This is macro DRV_SDCARD_INDEX_1.

5.1.8.7.6.9 DRV_SDCARD_INDEX_2 Macro

C

```
#define DRV_SDCARD_INDEX_2 2
```

Description

This is macro DRV_SDCARD_INDEX_2.

5.1.8.7.6.10 DRV_SDCARD_INDEX_3 Macro

C

```
#define DRV_SDCARD_INDEX_3 3
```

Description

This is macro DRV_SDCARD_INDEX_3.

5.1.8.8 Files

Files

Name	Description
drv_sdcard.h	SD Card Device Driver Interface File

Description

5.1.8.8.1 drv_sdcard.h

- SD Card Device Driver Interface
- The SD Card device driver provides a simple interface to manage the "SD Card" peripheral. This file defines the interface definitions and prototypes for the SD Card driver.

Enumerations

	Name	Description
	DRV_SDCARD_CLIENT_STATUS	Identifies the client-specific status of the SD card driver
	SYS_MEDIA_EVENT	Defines the buffer status

Functions

	Name	Description
≡	DRV_SDCARD_BufferStatusGet	Gets the status of an SD card operation (read/write)
≡	DRV_SDCARD_ClientStatus	Gets current client-specific status the SD Card driver
≡	DRV_SDCARD_Close	Closes an opened-instance of the SD Card driver
≡	DRV_SDCARD_Deinitialize	Deinitializes the specified instance of the SD Card driver module
≡	DRV_SDCARD_Initialize	Initializes the SD Card driver
≡	DRV_SDCARD_MediaStatusGet	Gets the status of the device.
≡	DRV_SDCARD_Open	Opens the specified SD Card driver instance and returns a handle to it
≡	DRV_SDCARD_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings
≡	DRV_SDCARD_SectorRead	Reads data from the sectors of the SD card.
≡	DRV_SDCARD_SectorsCountGet	Gets the number of sectors present in the selected SD card device.
≡	DRV_SDCARD_SectorSizeGet	Gets the sector size of the selected SD card device.
≡	DRV_SDCARD_SectorWrite	Writes sector(s) of data to an SD card.
≡	DRV_SDCARD_Status	Provides the current status of the SD Card driver module
≡	DRV_SDCARD_Tasks	Maintains the driver's state machine
≡	DRV_SDCARD_VersionGet	Gets SD card driver version in numerical format.
≡	DRV_SDCARD_VersionStrGet	Gets SD card driver version in string format.
≡	DRV_SDCARD_WriteProtectionIsEnabled	Gets the status of write protection.

Macros

	Name	Description
≡	DRV_SDCARD_INDEX_0	SD Card driver index definitions
≡	DRV_SDCARD_INDEX_1	This is macro DRV_SDCARD_INDEX_1.
≡	DRV_SDCARD_INDEX_2	This is macro DRV_SDCARD_INDEX_2.
≡	DRV_SDCARD_INDEX_3	This is macro DRV_SDCARD_INDEX_3.
≡	DRV_SDCARD_INDEX_COUNT	Number of valid SD Card driver indices
≡	SDCARD_MAX_LIMIT	Maximum allowed SD card instances

Types

	Name	Description
≡	DRV_SDCARD_BUFFER_HANDLE	Handle to a buffer added to the process queue.
≡	DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device

Company

Microchip Technology Incorporated

FileName: drv_sdcard.h

5.1.9 SPI Driver Library

5.1.9.1 Introduction

SPI Driver Library

for

Microchip Microcontrollers

This library provides an interface to manage the Serial Peripheral Interface (SPI) module on the Microchip family of microcontrollers in different modes of operation.

Description

Overview

The SPI module is a full duplex synchronous serial interface useful for communicating with other peripherals or microcontrollers in master/slave relationship and it can transfer data over short distances at high speeds. The peripheral devices may be serial EEPROMs, shift registers, display drivers, analog-to-digital converters, etc. The SPI module is compatible with Motorola's SPI and SIOP interfaces.

During data transfer devices can work either in master or in Slave mode. The source of synchronization is the system clock, which is generated by the master. The SPI module allows to connect one or more slave devices to a single master device via the same bus.

The SPI serial interface consists of four pins, which are further sub-divided into data and control lines:

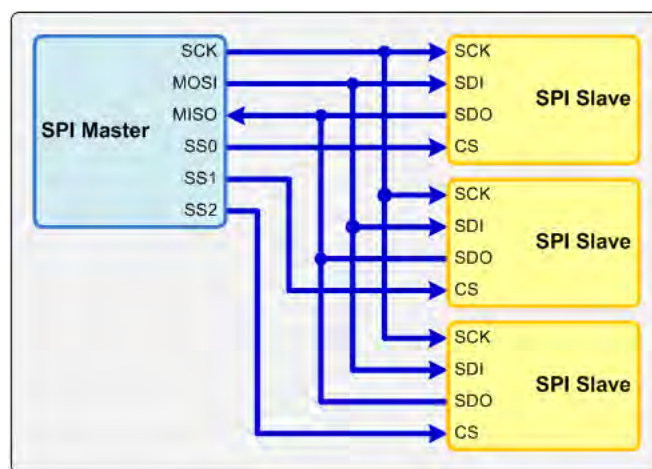
Data lines:

- MOSI – Master Data Output, Slave Data Input
- MISO – Master Data Input, Slave Data Output

Control lines:

- SCLK – Serial Clock
- /SS – Slave Select (no addressing)

SPI Master-Slave Relationship



The SPI module can be configured to operate using 2, 3 or 4 pins. In the 3-pin mode, slave select line is not used. In the 2-pin mode, both MOSI and SS lines are not used.

Note: Third-party trademarks are property of their respective owners.

5.1.9.2 Release Notes

MPLAB Harmony Version: v0.70b **SPI Driver Library Version :** 0.01 **Release Date:** 18Nov2013

New This Release:

- Configurable baud rate at client level
- Interoperability

Known Issues:

- Does not work in the Slave mode of operation
- DMA is not supported

5.1.9.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED AS IS MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.9.4 Using the Library

This topic describes the basic architecture of the SPI Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_spi.h](#)

The interface to the SPI Driver library is defined in the "[drv_spi.h](#)" header file. Any C language source (.c) file that uses the SPI Driver library should include this header.

Library File:

The SPI Driver library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the Driver interacts with the framework.

5.1.9.4.1 Abstraction Model

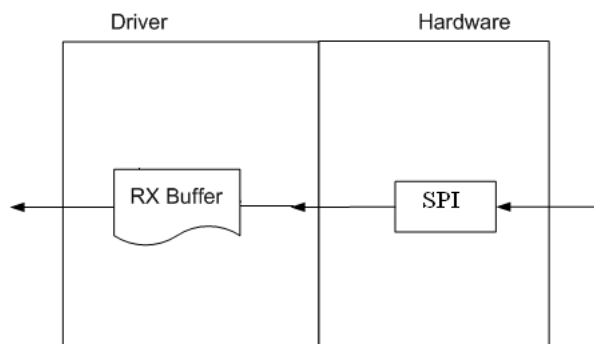
Different types of SPIs are available on Microchip microcontrollers. Some have an internal buffer mechanism and some do not.

The buffer depth varies across part families. The SPI driver abstracts out these differences and provides a unified model for data transfer across different types of SPIs available.

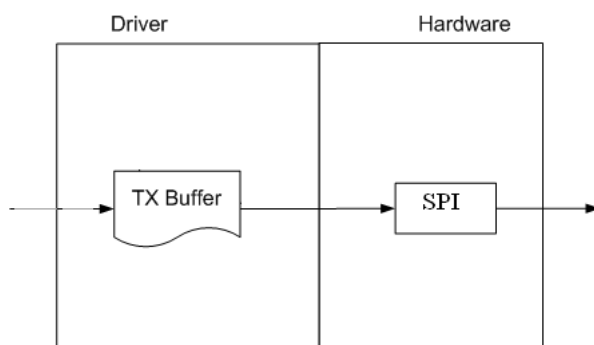
Both transmitter and receiver provides a buffer in the driver which transmits and receives data to/from the hardware. The SPI driver provides a set of interfaces to perform the read and the write.

The diagrams below illustrates the model used by the SPI driver for transmitter and receiver.

Receiver Abstraction Model



Transmitter Abstraction Model



5.1.9.4.2 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

The library interface routines are divided into various subsections, each of the sub section addresses one of the blocks or the overall operation of the SPI module.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open, close, status and other setup functions.
Chip Selects	Provides chip select functions available in the configuration.
Client Data Transfer - Core	Provides core data transfer functions available in the configuration. [Standard/Single buffer transfer]

Client Data Transfer - Advanced	Provides advanced data transfer functions available in the configuration. [Enhanced/Multi buffer transfers]
Miscellaneous	Provides driver miscellaneous functions, data transfer status function, version identification functions etc.

5.1.9.4.3 How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

Note: Not all modes are available on all microcontrollers, please refer to the specific device data sheet to the modes that are supported for your device.

5.1.9.4.3.1 System Access

System Initialization and Re-initialization

The system performs the initialization and the re-initialization of the device driver with settings that affect only the instance of the device that is being initialized or re-initialized. During system initialization each instance of the SPI module would be initialized with the following configuration settings (either passed dynamically at run time using [DRV_SPI_INIT](#) or by using Initialization Overrides) that are supported by the specific SPI device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to the Data Types section in this guide
- The actual peripheral ID enumerated as the PLIB level module ID. e.g SPI_ID_2.
- Defining the respective interrupt sources for Tx, Rx, and Error Interrupt.

The [DRV_SPI_Initialize](#) API returns an object handle of the type [SYS_MODULE_OBJ](#). There on, the object handle returned by the Initialize interface would be used by the other system interfaces like [DRV_SPI_Reinitialize](#), [DRV_SPI_Deinitialize](#), [DRV_SPI_Status](#) and [DRV_SPI_Tasks](#).

Note: The system initialization and the re-initialization settings, only effect the instance of the peripheral that is being initialized or re-initialized.

Example:

```

DRV_SPI_INIT          spiInitData;
SYS_MODULE_OBJ        objectHandle;

spiInitData.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
spiInitData.spiId           = SPI_ID_1;
spiInitData.spiMode         = DRV_SPI_MODE_MASTER;
spiInitData.spiProtocolType = DRV_SPI_PROTOCOL_TYPE_STANDARD;
spiInitData.commWidth       = SPI_COMMUNICATION_8BIT_WIDE;
spiInitData.baudRate        = 5000;
spiInitData.bufferType      = DRV_SPI_BUFFER_TYPE_STANDARD;
spiInitData.inputSamplePhase = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
spiInitData.clockMode       = DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE;
spiInitData.txInterruptSource = INT_SOURCE_SPI_1_TRANSMIT; // INT_SOURCE_SPI_1_EVENT
spiInitData.rxInterruptSource = INT_SOURCE_SPI_1_RECEIVE;  // INT_SOURCE_SPI_1_EVENT
spiInitData.errInterruptSource = INT_SOURCE_SPI_1_ERROR;
spiInitData.pinConfig.sdiPin  = PIN_6; // RPI38
spiInitData.pinConfig.sdoPin  = PIN_10; // RP21
spiInitData.pinConfig.sckPin  = PIN_11; // RP26

objectHandle = DRV_SPI_Initialize(DRV_SPI_INDEX_1, (SYS_MODULE_INIT*)&spiInitData);

```

```
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Tasks Routine

The system will either call [DRV_SPI_Tasks](#), from System Task Service (in a polled environment) or [DRV_SPI_Tasks](#) will be called from interrupt service routine of the SPI.

5.1.9.4.3.2 Client Access

General Client Operation

For the application to start using an instance of the module, it must call the [DRV_SPI_Open](#) function. This provides the configuration required to open the SPI instance for operation. If the driver is de-initialized using the function [DRV_SPI_Deinitialize](#), the application must call the [DRV_SPI_Open](#) function again to setup the instance of the SPI. The function [DRV_SPI_Open](#) need not be called again if the system is re-initialized using the [DRV_SPI_Reinitialize](#) function.

For the various options available for IO INTENT please refer to the Data Types section in this guide.

Example:

```
DRV_HANDLE handle;

// Configure the instance DRV_SPI_INDEX_1 with the configuration
handle = DRV_SPI_Open(DRV_SPI_INDEX_1, DRV_IO_INTENT_READWRITE);

if(handle == DRV_HANDLE_INVALID)
{
    // Client cannot open the instance.
}
```

Communication Setup:

Communication entities like the SPI modes, input clock frequency, baud rates, clock mode, ports setup etc. can be configured with the help of the driver function [DRV_SPI_CommunicationSetup](#). This is a mandatory setup before doing any data transfer. The callback function registered during the communication setup gets called after a successful data transfer.

Example:

```
DRV_HANDLE handle; // Returned from DRV_SPI_Open
DRV_SPI_COMM_CONFIG config;

// Populate the communication setup configuration structure
config.spiMode      = DRV_SPI_MODE_MASTER;
config.baudRate     = 5000;
config.inputSamplePhase = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
config.clockMode    = DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE;
config.callback      = &callbackFunction; // callback registered
DRV_SPI_CommunicationSetup( handle, &config );
```

Chip Select Control:

User can control the chip/slave select signals using the driver APIs [DRV_SPI_ChipSelectActivate](#) and [DRV_SPI_ChipSelectDeactivate](#) with appropriate parameters.

Example:

```
// Activates/Enables Slave Select connected to Port A, bit 1
DRV_SPI_ChipSelectActivate( handle, PORT_CHANNEL_A, PORTS_BIT_POS_1 );
...
...
...
// Deactivates/Disables Slave Select connected to Port A, bit 1
```

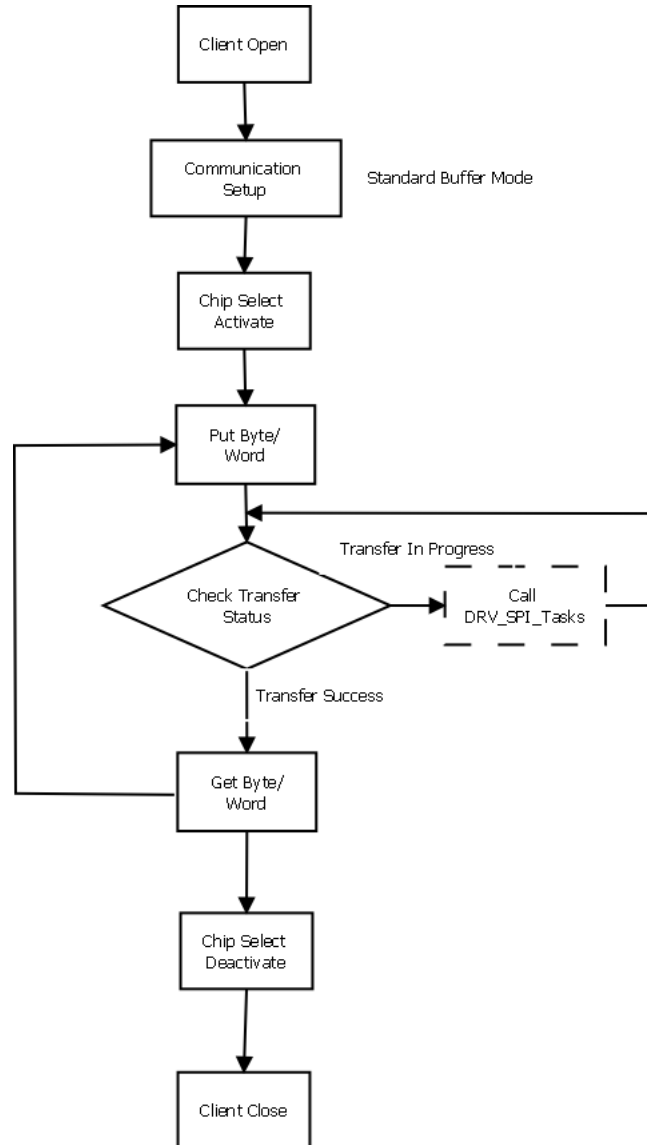
```
DRV_SPI_ChipSelectDeactivate( handle, PORT_CHANNEL_A, PORTS_BIT_POS_1 );
```

5.1.9.4.3.3 Client Transfer - Core

Client basic functionality provides a extremely basic interface for the driver operation.

This client transfer mechanism needs to be used for the devices which support the 'Standard' SPI mode of operation [Single Buffer]. This mechanism needs to be repeated in case of buffer based data transfers for transferring multi bytes/words.

The diagram below illustrates the byte/word model used for the data transfer.



Note: It is not necessary to close and re-open the client between multiple transfers.

Client Byte/Word transfer Functionality

Application using the SPI byte/word functionality, needs to perform the following:

1. The system should have completed necessary initialization and the [DRV_SPI_Tasks](#) should either be running in polled

environment, or in an interrupt environment.

2. Open the driver using [DRV_SPI_Open](#) with the necessary intent
3. Setup the communication using the driver function DRV_SPI_CommunicationSetup
4. Enable/activate the slave using the function DRV_SPI_ChipSelectActivate
5. Put/Write a byte using DRV_SPI_Put, dummy byte in case of read only
6. Check for the current transfer status using DRV_SPI_TransferStatus until the transfer is in progress
DRV_SPI_TRANSFER_STATUS_IN_PROGRESS, call the [DRV_SPI_Tasks](#) in case of a polled environment until the transfer is success
7. Get/Read the dummy/valid data using the function DRV_SPI_Get
8. Repeat from step 5 till 7 to handle buffer transmission and reception using standard mode
9. Disable/deactivate the slave using the function DRV_SPI_ChipSelectDeactivate
10. The client will be able to close the driver using the [DRV_SPI_Close](#) when required.

Example:

```

DRV_HANDLE          handle;      // Returned from DRV_SPI_Open
SPI_BUFFER           myBuffer[MY_BUFFER_SIZE];
unsigned int         numBytes;
DRV_SPI_COMM_CONFIG  config;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

while( 1 )
{
    switch( state )
    {
        case APP_STATE_INIT:
            /* Initialize the SPI Driver */
            object = DRV_SPI_Initialize( sysIndex, ( SYS_MODULE_INIT * )&initConf );

            /* Check for the System Status */
            if( SYS_STATUS_READY != DRV_SPI_Status( object ) )
                return 0;

            /* Open the Driver */
            clHandle = DRV_SPI_Open( sysIndex, DRV_IO_INTENT_EXCLUSIVE );

            /* Check for the Client Open Status */
            if( DRV_SPI_CLIENT_STATUS_READY != DRV_SPI_ClientStatus( clHandle ) )
                return 0;

            /* Set up the communication medium */
            DRV_SPI_CommunicationSetup( clHandle, &commConf );

            /* Update the state to transfer data */
            state = APP_STATE_DATA_PUT;
            break;

        case APP_STATE_DATA_PUT:
            /* Enable/Activate the CS */
            DRV_SPI_ChipSelectActivate( handle, PORT_CHANNEL_A, PORTS_BIT_POS_1 );
            /* Initiate the data transfer */
            DRV_SPI_Put( clHandle, myBuffer[i++] );
            /* Deactivates/Disables CS */
            DRV_SPI_ChipSelectDeactivate( handle, PORT_CHANNEL_A, PORTS_BIT_POS_1 );
            /* Update the state to status check */
            state = APP_STATE_STATUS_CHECK;
            break;

        case APP_STATE_STATUS_CHECK:

```



```

    /* Check for the successful data transfer */
    if( DRV_SPI_TRANSFER_STATUS_IN_PROGRESS & DRV_SPI_TransferStatus( clHandle ) )
    {
        DRV_SPI_Tasks (object);
        /* Stay here until the data transfer is executed */
        state = APP_STATE_STATUS_CHECK;
    }
    else
    {
        /* Data transfer was successfull, get the data */
        state = APP_STATE_DATA_GET;
    }
    break;

case APP_STATE_DATA_GET:
    /* Read the data from the peripheral */
    rxd_data = DRV_SPI_Get( clHandle );
    /* Restart the process by jumping to the beginning state */
    state = APP_STATE_DATA_PUT;
    break;

default:
    break;
}
}

```

Note: 'bufferType' attribute of the initialization structure or the configuration macro '[DRV_SPI_BUFFER_USAGE_TYPE](#)' needs to be updated as DRV_SPI_BUFFER_TYPE_STANDARD

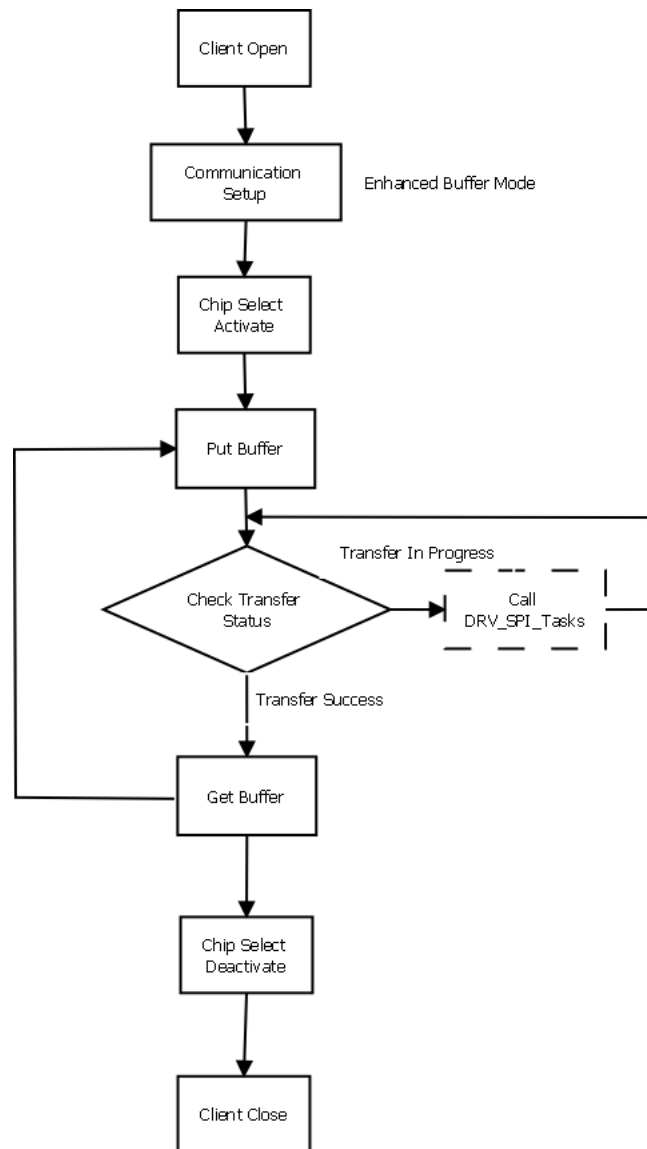
5.1.9.4.3.4 Client Transfer - Advanced

Client basic functionality provides a extremely basic interface for the driver operation.

This client transfer mechanism needs to be used for the devices which support the

'Enhanced' SPI mode of operation [Internal FIFO or buffer].

The diagram below illustrates the buffer model used for the data transfer.



Note: It is not necessary to close and re-open the client between multiple transfers.

Client Byte/Word transfer Functionality

Application using the SPI byte/word functionality, needs to perform the following:

1. The system should have completed necessary initialization and the [DRV_SPI_Tasks](#) should either be running in polled environment, or in an interrupt environment.
2. Open the driver using [DRV_SPI_Open](#) with the necessary intent
3. Setup the communication using the driver function [DRV_SPI_CommunicationSetup](#)
4. Enable/activate the slave using the function [DRV_SPI_ChipSelectActivate](#)
5. Put/Write a byte using [DRV_SPI_PutBuffer](#), dummy byte in case of read only
6. Check for the current transfer status using [DRV_SPI_TransferStatus](#) until the transfer is in progress [DRV_SPI_TRANSFER_STATUS_IN_PROGRESS](#), call the [DRV_SPI_Tasks](#) in case of a polled environment until the transfer is success
7. Get/Read the dummy/valid data using the function [DRV_SPI_GetBuffer](#)

8. Repeat from step 5 till 7 to handle multi buffer transmission and reception using enhanced mode
9. Disable/deactivate the slave using the function `DRV_SPI_ChipSelectDeactivate`
10. The client will be able to close the driver using the `DRV_SPI_Close` when required.

Example:

```

DRV_HANDLE    handle;    // Returned from DRV_SPI_Open
SPI_DATA_TYPE tx_buffer[DRV_SPI_BUFFER_SIZE] = { 0x0001, 0x0001, 0x0001, 0x0001 };
SPI_DATA_TYPE rx_buffer[DRV_SPI_BUFFER_SIZE];
unsigned int   numBytes;
DRV_SPI_COMM_CONFIG config;

// Pre-initialize tx_buffer with DRV_SPI_BUFFER_SIZE bytes of valid data.

while( 1 )
{
    switch( state )
    {
        case APP_STATE_INIT:
            /* Initialize the SPI Driver */
            object = DRV_SPI_Initialize( sysIndex, ( SYS_MODULE_INIT * )&initConf );

            /* Check for the System Status */
            if( SYS_STATUS_READY != DRV_SPI_Status( object ) )
                return 0;

            /* Open the Driver */
            clHandle = DRV_SPI_Open( sysIndex, DRV_IO_INTENT_EXCLUSIVE );

            /* Check for the Client Open Status */
            if( DRV_SPI_CLIENT_STATUS_READY != DRV_SPI_ClientStatus( clHandle ) )
                return 0;

            /* Set up the communication medium */
            DRV_SPI_CommunicationSetup( clHandle, &commConf );

            /* Update the state to transfer data */
            state = APP_STATE_BUFFER_PUT;
            break;

        case APP_STATE_BUFFER_PUT:
            /* Initiate the data transfer */
            DRV_SPI_PutBuffer( clHandle, &tx_buffer, sizeof( tx_buffer )/2 );
            /* Update the state to status check */
            state = APP_STATE_BUFFER_CHECK;
            break;

        case APP_STATE_BUFFER_CHECK:
            /* Check for the successful data transfer */
            if( DRV_SPI_TRANSFER_STATUS_IN_PROGRESS & DRV_SPI_TransferStatus( clHandle ) )
            {
                DRV_SPI_Tasks( object );
                /* Stay here until the data transfer is executed */
                state = APP_STATE_BUFFER_CHECK;
            }
            else
            {
                /* Data transfer was successfull, get the data */
                state = APP_STATE_BUFFER_GET;
            }
            break;

        case APP_STATE_BUFFER_GET:
            /* Read the data from the peripheral */
            DRV_SPI_GetBuffer( clHandle, &rx_buffer, sizeof( rx_buffer )/2 );

```

```

        /* Restart the process by jumping to the beginning state */
        state = APP_STATE_BUFFER_PUT;
        break;

    default:
        break;
}
}

```

Note: 'bufferType' attribute of the initialization structure or the configuration macro '[DRV_SPI_BUFFER_USAGE_TYPE](#)' needs to be updated as DRV_SPI_BUFFER_TYPE_ENHANCED

5.1.9.5 Configuring the Library

The configuration of the SPI driver is based on the file sys_config.h

This header file contains the configuration selection for the SPI driver. Based on the selections made, the SPI driver will support or not support selected features. These configuration settings will apply to all instances of the SPI driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.9.5.1 Client Configuration

5.1.9.5.2 System Configuration

5.1.9.5.3 Miscellaneous Configuration

5.1.9.6 Building the Library

This section list the files that are available in the \src of the SPI driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.1.9.7 Library Interface

Data Types and Constants

	Name	Description
	DRV_SPI_BAUD_RATE	Controls the baud rate value of the SPI driver.
	DRV_SPI_BAUD_RATE_CLOCK	Selects the SPI baud rate clock.
	DRV_SPI_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_SPI_BUFFER_SIZE	This is macro DRV_SPI_BUFFER_SIZE.
	DRV_SPI_BUFFER_USAGE_TYPE	Controls the buffer type of the SPI driver.

	DRV_SPI_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_SPI_CLOCK_OPERATION_MODE	Controls the clock mode of the SPI driver.
	DRV_SPI_COMMUNICATION_WIDTH	Controls the communication width of the SPI driver.
	DRV_SPI_FRAME_SYNC_PULSE_COUNT	Selects the SPI frame sync pulse count.
	DRV_SPI_FRAME_SYNC_PULSE_DIRECTION	Selects the SPI frame sync pulse direction.
	DRV_SPI_FRAME_SYNC_PULSE_EDGE	Selects the SPI frame sync pulse edge.
	DRV_SPI_FRAME_SYNC_PULSE_POLARITY	Selects the SPI frame sync pulse polarity.
	DRV_SPI_FRAME_SYNC_PULSE_WIDTH	Selects the SPI frame sync pulse width.
	DRV_SPI_INDEX	SPI static index selection.
	DRV_SPI_INDEX_0	SPI driver index definitions.
	DRV_SPI_INDEX_COUNT	Number of valid SPI driver indices.
	DRV_SPI_INPUT_SAMPLE_PHASE	Selects the SPI input sample phase.
	DRV_SPI_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver .
	DRV_SPI_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_SPI_INTERRUPT_SOURCE_ERROR	Defines the error interrupt source in case of static driver
	DRV_SPI_INTERRUPT_SOURCE_RX	Defines the receive interrupt source in case of static driver.
	DRV_SPI_INTERRUPT_SOURCE_TX	Defines the transmit/receive or transmit alone interrupt source in case of static driver.
	DRV_SPI_PERIPHERAL_ID	SPI peripheral ID selection.
	DRV_SPI_PORTS_REMAP_USAGE	Selects the SPI pins remap.
	DRV_SPI_POWER_STATE	Controls the power state of the SPI driver.
	DRV_SPI_PROTOCOL	Controls the protocol type of the SPI driver.
	DRV_SPI_USAGE_MODE	Controls the usage mode of the SPI driver.
	DRV_SPI_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_SPI_BUFFER_EVENT_HANDLER	Pointer to a SPI Driver Buffer Event handler function
	DRV_SPI_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_SPI_BUFFER_TYPE	Identifies the various buffer types of the SPI module.
	DRV_SPI_CLIENT_SETUP	Defines the data required to initialize an SPI client.
	DRV_SPI_CLOCK_MODE	Identifies the various clock modes of the SPI module.
	DRV_SPI_INIT	Defines the data required to initialize or reinitialize the SPI driver
	DRV_SPI_MODE	Identifies the various usage modes of the SPI module.
	DRV_SPI_PROTOCOL_TYPE	Identifies the various protocols of the SPI module.

Client Setup Functions

	Name	Description
	DRV_SPI_BufferStatus	Returns the transmitter and receiver transfer status.
	DRV_SPI_ClientSetup	Initializes a client.
	DRV_SPI_Close	Closes an opened instance of the SPI driver
	DRV_SPI_Open	Opens the specified SPI driver instance and returns a handle to it.

Miscellaneous Functions

	Name	Description
	DRV_SPI_BufferAddRead	Registers a buffer for a read operation. Actual transfer will happen in the Task function.

	DRV_SPI_BufferAddWrite	Registers a buffer for a read operation. Actual transfer will happen in the Task function.
	DRV_SPI_BufferAddWriteRead	Registers a buffer for a read operation. Actual transfer will happen in the Task function.
	DRV_SPI_VersionGet	Gets the SPI driver version in numerical format.
	DRV_SPI_VersionStrGet	Gets the SPI driver version in string format.
	DRV_SPI_INDEX_1	This is macro DRV_SPI_INDEX_1.
	DRV_SPI_INDEX_2	This is macro DRV_SPI_INDEX_2.
	DRV_SPI_INDEX_3	This is macro DRV_SPI_INDEX_3.
	DRV_SPI_INDEX_4	This is macro DRV_SPI_INDEX_4.
	DRV_SPI_INDEX_5	This is macro DRV_SPI_INDEX_5.
	DRV_SPI_RX_FIFO_INTERRUPT_MODE	This is macro DRV_SPI_RX_FIFO_INTERRUPT_MODE.
	DRV_SPI_TX_FIFO_INTERRUPT_MODE	This is macro DRV_SPI_TX_FIFO_INTERRUPT_MODE.

System Interaction Functions

	Name	Description
	DRV_SPI_Deinitialize	De-initializes the specified instance of the SPI driver module.
	DRV_SPI_Initialize	Initializes the SPI driver.
	DRV_SPI_Status	Provides the current status of the SPI driver module.
	DRV_SPI_Tasks	Maintains the driver's state machine and implements its ISR.
	DRV_SPI_Read	Gets/Reads byte(s) from SPI. The function will wait until the specified number of bytes is received.
	DRV_SPI_Write	Writes the data to SPI. The function will wait until all the bytes are transmitted.

Description

This section describes the API functions of the SPI driver library.

Refer to each section for a detailed description.

5.1.9.7.1 System Interaction Functions

5.1.9.7.1.1 DRV_SPI_Deinitialize Function

C

```
void DRV_SPI_Deinitialize(
    SYS_MODULE_OBJ object
);
```

Description

De-initializes the specified instance of the SPI driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Preconditions

Function [DRV_SPI_Initialize](#) must have been called before calling this routine and a valid [SYS_MODULE_OBJ](#) must have been returned.

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_SPI_Initialize

Returns

None.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the [DRV_SPI_Status](#) operation. The system has to use [DRV_SPI_Status](#) to find out when the module is in the ready state.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_SPI_Initialize
SYS_STATUS        status;

DRV_SPI_Deinitialize ( object );

status = DRV_SPI_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is de-initialized.
}
```

5.1.9.7.1.2 DRV_SPI_Initialize Function**C**

```
SYS_MODULE_OBJ DRV_SPI_Initialize(
    const SYS_MODULE_INDEX index,
    const SYS_MODULE_INIT * const init
);
```

Description

This function initializes the SPI driver, making it ready for clients to open and use.

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver. This pointer may be null if no data is required because static overrides have been provided.

Returns

If successful, a valid handle to a driver object is returned. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#). The returned object must be passed as an argument to the [DRV_SPI_Reinitialize](#), [DRV_SPI_Deinitialize](#), [DRV_SPI_Tasks](#), and [DRV_SPI_Status](#) functions.

Remarks

This function must be called before any other SPI routine is called.

This function should only be called once during system initialization unless [DRV_SPI_Deinitialize](#) is called to de-initialize the driver instance.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be

reported by the [DRV_SPI_Status](#) operation. The system must use [DRV_SPI_Status](#) to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Example

```
DRV_SPI_INIT      init;
SYS_MODULE_OBJ    objectHandle;

// Populate the SPI initialization structure
init.spiId        = SPI_ID_1;
init.spiMode      = DRV_SPI_MODE_MASTER;
init.spiProtocolType = DRV_SPI_PROTOCOL_TYPE_STANDARD;
init.commWidth    = SPI_COMMUNICATION_8BIT_WIDE;
init.baudRate     = 5000;
init.bufferType   = DRV_SPI_BUFFER_TYPE_STANDARD;
init.inputSamplePhase = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
init.clockMode    = DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE;
init.txInterruptSource = INT_SOURCE_SPI_1_TRANSMIT; // INT_SOURCE_SPI_1_EVENT
init.rxInterruptSource = INT_SOURCE_SPI_1_RECEIVE;  // INT_SOURCE_SPI_1_EVENT
init.errInterruptSource = INT_SOURCE_SPI_1_ERROR;

// Do something

objectHandle = DRV_SPI_Initialize( DRV_SPI_INDEX_0, (SYS_MODULE_INIT*)&init );
if( SYS_MODULE_OBJ_INVALID == objectHandle )
{
    // Handle error
}
```

5.1.9.7.1.3 DRV_SPI_Status Function

C

```
SYS_STATUS DRV_SPI_Status(
    SYS_MODULE_OBJ object
);
```

Description

This function provides the current status of the SPI driver module.

Preconditions

The [DRV_SPI_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_SPI_Initialize

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to [accept](#) new operations.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been de-initialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns `SYS_STATUS_BUSY`, the previous operation has not yet completed. Once the status operation returns `SYS_STATUS_READY`, any previous operations have completed.

The value of `SYS_STATUS_ERROR` is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the de-initialize operation will need to be called, followed by the initialize operation to return to normal operations.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_SPI_Initialize
SYS_STATUS        status;

status = DRV_SPI_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

5.1.9.7.1.4 DRV_SPI_Tasks Function

C

```
void DRV_SPI_Tasks(
    SYS_MODULE_OBJ object
);
```

Description

This routine is used to maintain the driver's internal state machine and implement its transmit ISR for interrupt-driven implementations. In polling mode, this function should be called from the `SYS_Tasks()` function. In interrupt mode, this function should be called in the transmit interrupt service routine of the USART that is associated with this USART driver hardware instance.

Preconditions

The `DRV_SPI_Initialize` routine must have been called for the specified SPI driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from <code>DRV_SPI_Initialize</code>)

Returns

None.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (`SYS_Tasks`) or by the appropriate raw ISR.

This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_SPI_Initialize

while( true )
{
    DRV_SPI_Tasks ( object );

    // Do other tasks
}
```

5.1.9.7.1.5 DRV_SPI_Read Function

C

```
size_t DRV_SPI_Read(
    DRV_HANDLE handle,
    void* buffer,
    size_t noOfBytes
);
```

Description

This function gets/reads byte(s) from SPI. The function will wait until the specified number of bytes is received.

Preconditions

The [DRV_SPI_Initialize](#) routine must have been called for the specified SPI driver instance.

[DRV_SPI_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_SPI_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer to which the data should be written to.
noOfBytes	Number of bytes to be received.

Returns

The number of bytes read.

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_SPI_Open](#) function and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_SPI_Open](#) function or the driver was built to only support non-blocking behavior the call will return immediately. If data is not available, a zero ('0') value will be returned and an underrun error status will be captured. To ensure that valid data is returned, the caller must first check the return value to [DRV_SPI_BufferStatus](#), as shown in the example.

Example

```
DRV_HANDLE    handle;    // Returned from DRV_SPI_Open
SPI_DATA_TYPE myBuffer[MY_BUFFER_SIZE];
unsigned int  numberOfBytesRead;

numberOfBytesRead = DRV_SPI_Read ( handle, myBuffer, 10 );

if ( numberOfBytesWritten == 0 )
{
    // Check for the error...
}
```

5.1.9.7.1.6 DRV_SPI_Write Function

C

```
size_t DRV_SPI_Write(
    DRV_HANDLE handle,
    void* buffer,
    size_t noOfBytes
);
```

Description

This function Writes the data to SPI. The function will wait until all the bytes are transmitted.

Preconditions

The [DRV_SPI_Initialize](#) routine must have been called for the specified SPI driver instance.

[DRV_SPI_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_SPI_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer which contains the data.
noOfBytes	Number of bytes to be transmitted.

Returns

The number of bytes written.

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_SPI_Open](#) function and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_SPI_Open](#) function or the driver was built to only support non-blocking behavior the call will return immediately. If the driver was not able to [accept](#) the data, an overrun error status will be captured. To ensure that the driver is ready to [accept](#) data, the caller must first check the return value to [DRV_SPI_BufferStatus](#), as shown in the example.

Example

```

DRV_HANDLE      handle;      // Returned from DRV_SPI_Open
SPI_DATA_TYPE   myBuffer[MY_BUFFER_SIZE]; // fill the data
unsigned int    numberOfBytesWritten;

numberOfBytesWritten = DRV_SPI_Write ( handle, myBuffer, 10 );

if ( numberOfBytesWritten == 0 )
{
    // Check the error...
}

```

5.1.9.7.2 Client Setup Functions

5.1.9.7.2.1 DRV_SPI_BufferStatus Function

C

```

DRV_SPI_BUFFER_EVENT DRV_SPI_BufferStatus(
    DRV_SPI_BUFFER_HANDLE bufferHandle
);

```

Description

This returns the transmitter and receiver transfer status.

Preconditions

The [DRV_SPI_Initialize](#) routine must have been called for the specified SPI driver instance.

[DRV_SPI_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

A DRV_SPI_BUFFER_STATUS value describing the current status of the transfer.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_SPI_BUFFER_STATUS enumeration set. The caller should perform an AND with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Example

```
DRV_HANDLE handle;    // Returned from DRV_SPI_Open

if( DRV_SPI_BUFFER_STATUS_SUCCESS & DRV_SPI_BufferStatus( handle ) )
{
    // All transmitter data has been sent.
}
```

5.1.9.7.2.2 DRV_SPI_ClientSetup Function**C**

```
void DRV_SPI_ClientSetup(
    DRV_HANDLE handle,
    const DRV_SPI_CLIENT_SETUP * const clientSetup
);
```

Description

This function sets up the device communication parameters. Using this API user can have configurations like chip select and baud rate specific to a client. The driver will be able to switch the between clients dynamically. 'DRV_SPI_CLIENT_SPECIFIC_CONTROL' must be defined in 'system_config.h'. This function is only useful for instance which has got multiple slaves.

Preconditions

The [DRV_SPI_Initialize](#) routine must have been called.

[DRV_SPI_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open routine
clientSetup	Communication parameters identified by DRV_SPI_CLIENT_SETUP

Returns

None

Remarks

This is an optional call. It is necessary only two clients using needs different configurations.

Example

```
DRV_HANDLE handle;    // Returned from DRV_SPI_Open
DRV_SPI_CLIENT_SETUP clientSetup;

// Populate the communication setup configuration structure
```

```

clientSetup.baudRate      = 5000;
clientSetup.inputSamplePhase = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
clientSetup.clockMode      = DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE;
clientSetup.callback       = &callbackFunction; // callback registered

DRV_SPI_ClientSetup ( handle, &clientSetup );

```

5.1.9.7.2.3 DRV_SPI_Close Function

C

```

void DRV_SPI_Close(
    DRV_HANDLE handle
);

```

Description

This function closes an opened instance of the SPI driver, invalidating the handle.

Preconditions

The [DRV_SPI_Initialize](#) routine must have been called for the specified SPI driver instance.

[DRV_SPI_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

None

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling [DRV_USART_Open](#) before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually there is no need for the driver client to verify that the Close operation has completed.

Example

```

DRV_HANDLE handle; // Returned from DRV_SPI_Open

DRV_SPI_Close ( handle );

```

5.1.9.7.2.4 DRV_SPI_Open Function

C

```

DRV_HANDLE DRV_SPI_Open(
    const SYS_MODULE_INDEX drvIndex,
    const DRV_IO_INTENT intent
);

```

Description

This routine opens the specified USART driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options additionally affect the behavior of the [DRV_USART_Read\(\)](#) and [DRV_USART_Write\(\)](#) functions. If the ioIntent is DRV_IO_INTENT_NONBLOCKING, then these function will not block even if the required amount of data could not be processed. If the ioIntent is DRV_IO_INTENT_BLOCKING, these functions will block till the required amount of data is processed.

If `ioIntent` is `DRV_IO_INTENT_READ`, the client will only be read from the driver. If `ioIntent` is `DRV_IO_INTENT_WRITE`, the client will only be able to write to the driver. If the `ioIntent` is `DRV_IO_INTENT_READWRITE`, the client will be able to do both, read and write.

Specifying a `DRV_IO_INTENT_EXCLUSIVE` will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Preconditions

The [DRV_SPI_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
<code>drvIndex</code>	Identifier for the object instance to be opened
<code>intent</code>	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is [DRV_HANDLE_INVALID](#). Error can occur

- if the number of client objects allocated via [DRV_USART_CLIENTS_NUMBER](#) is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.

Remarks

The handle returned is valid until the [DRV_USART_Close](#) routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return [DRV_HANDLE_INVALID](#). This function is thread safe in a RTOS application. It should not be called in an ISR.

Example

```
DRV_HANDLE  handle;

handle = DRV_SPI_Open( DRV_SPI_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );
if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

5.1.9.7.3 Miscellaneous Functions

5.1.9.7.3.1 DRV_SPI_BufferAddRead Function

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddRead(
    DRV_HANDLE handle,
    void * rxBuffer,
    size_t size
);
```

Description

Registers a buffer for a read operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using [DRV_SPI_BufferStatus](#) function.

Preconditions

The [DRV_SPI_Initialize](#) routine must have been called for the specified SPI driver instance.

[DRV_SPI_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_SPI_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
rxBuffer	The buffer to which the data should be written to.
size	Number of bytes to be read.

Returns

None.

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_SPI_Open](#) function and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_SPI_Open](#) function or the driver was built to only support non-blocking behavior the call will return immediately. If the driver was not able to [accept](#) the data, an overrun error status will be captured. To ensure that the driver is ready to [accept](#) data, the caller must first check the return value to [DRV_SPI_BufferStatus](#), as shown in the example.

Example

```
DRV_HANDLE    handle;    // Returned from DRV_SPI_Open
SPI_DATA_TYPE myBuffer[MY_BUFFER_SIZE];

// PUT API returns data in any case, upto the user to use it
DRV_SPI_BufferAddRead( handle, myBuffer, 10 );

if( DRV_SPI_BUFFER_STATUS_SUCCESS & DRV_SPI_BufferStatus( handle ) )
{
    // All transmitter data has been sent.
}
```

5.1.9.7.3.2 DRV_SPI_BufferAddWrite Function**C**

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWrite(
    DRV_HANDLE handle,
    void * txBuffer,
    size_t size
);
```

Description

Registers a buffer for a read operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using [DRV_SPI_BufferStatus](#) function.

Preconditions

The [DRV_SPI_Initialize](#) routine must have been called for the specified SPI driver instance.

[DRV_SPI_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_SPI_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
rxBuffer	The buffer to which the data should be written to.
size	Number of bytes to be read.

Returns

None.

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_SPI_Open](#) function and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_SPI_Open](#) function or the driver was built to only support non-blocking behavior the call will return immediately. If the driver was not able to [accept](#) the data, an overrun error status will be captured. To ensure that the driver is ready to [accept](#) data, the caller must first check the return value to [DRV_SPI_BufferStatus](#), as shown in the example.

Example

```
DRV_HANDLE    handle;    // Returned from DRV_SPI_Open
SPI_DATA_TYPE myBuffer[MY_BUFFER_SIZE];
unsigned int   numBytes;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;

// PUT API returns data in any case, upto the user to use it
DRV_SPI_BufferAddRead( handle, myBuffer, 10 );

if( DRV_SPI_BUFFER_STATUS_SUCCESS & DRV_SPI_BufferStatus( handle ) )
{
    // All transmitter data has been sent.
}
```

5.1.9.7.3.3 DRV_SPI_BufferAddWriteRead Function

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWriteRead(
    DRV_HANDLE handle,
    void * txBuffer,
    void * rxBuffer,
    size_t size
);
```

Description

Registers a buffer for a read operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using [DRV_SPI_BufferStatus](#) function.

Preconditions

The [DRV_SPI_Initialize](#) routine must have been called for the specified SPI driver instance.

[DRV_SPI_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_SPI_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
rxBuffer	The buffer to which the data should be written to.
size	Number of bytes to be read.

Returns

None.

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_SPI_Open](#) function and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_SPI_Open](#) function or the driver was built to only support non-blocking behavior the call will return immediately. If the driver was not able to [accept](#) the data, an overrun error status will be captured. To ensure that the driver is ready to [accept](#) data, the caller must first check the return value to [DRV_SPI_BufferStatus](#), as shown in the example.

Example

```
DRV_HANDLE      handle;      // Returned from DRV_SPI_Open
SPI_DATA_TYPE   myBuffer[MY_BUFFER_SIZE];
unsigned int     numBytes;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;

// PUT API returns data in any case, upto the user to use it
DRV_SPI_BufferAddWriteRead( handle, myTxBuffer, myRxBuffer, 10 );

if( DRV_SPI_BUFFER_STATUS_SUCCESS & DRV_SPI_BufferStatus( handle ) )
{
    // All transmitter data has been sent.
}
```

5.1.9.7.3.4 DRV_SPI_VersionGet Function**C**

```
unsigned int DRV_SPI_VersionGet(
    const SYS_MODULE_INDEX drvIndex
);
```

Description

This function gets the SPI driver version. The version is encoded as major * 10000 + minor * 100 + patch. The stringed version can be obtained using [DRV_SPI_VersionStrGet](#).

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for

Returns

Current driver version in numerical format.

Remarks

None.

Example

```
unsigned int version;

version = DRV_SPI_VersionGet( DRV_SPI_INDEX_0 );

if(version < 110200)
{
    // Do Something
}
```

5.1.9.7.3.5 DRV_SPI_VersionStrGet Function**C**

```
char * DRV_SPI_VersionStrGet(
    const SYS_MODULE_INDEX drvIndex
);
```

Description

This function gets the SPI driver version. The version is returned as major.minor.path[type], where type is optional. The numerical version can be obtained using [DRV_SPI_VersionGet](#).

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for

Returns

Current the SPI driver version in the string format.

Remarks

None.

Example

```
char *version;

version = DRV_SPI_VersionStrGet( DRV_SPI_INDEX_0 );

printf("%s", version);
```

5.1.9.7.3.6 DRV_SPI_INDEX_1 Macro**C**

```
#define DRV_SPI_INDEX_1 1
```

Description

This is macro DRV_SPI_INDEX_1.

5.1.9.7.3.7 DRV_SPI_INDEX_2 Macro**C**

```
#define DRV_SPI_INDEX_2 2
```

Description

This is macro DRV_SPI_INDEX_2.

5.1.9.7.3.8 DRV_SPI_INDEX_3 Macro

C

```
#define DRV_SPI_INDEX_3 3
```

Description

This is macro DRV_SPI_INDEX_3.

5.1.9.7.3.9 DRV_SPI_INDEX_4 Macro

C

```
#define DRV_SPI_INDEX_4 4
```

Description

This is macro DRV_SPI_INDEX_4.

5.1.9.7.3.10 DRV_SPI_INDEX_5 Macro

C

```
#define DRV_SPI_INDEX_5 5
```

Description

This is macro DRV_SPI_INDEX_5.

5.1.9.7.3.11 DRV_SPI_RX_FIFO_INTERRUPT_MODE Macro

C

```
#define DRV_SPI_RX_FIFO_INTERRUPT_MODE SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_FULL
```

Description

This is macro DRV_SPI_RX_FIFO_INTERRUPT_MODE.

5.1.9.7.3.12 DRV_SPI_TX_FIFO_INTERRUPT_MODE Macro

C

```
#define DRV_SPI_TX_FIFO_INTERRUPT_MODE SPI_FIFO_INTERRUPT_WHEN_TRANSMISSION_IS_COMPLETE
```

Description

This is macro DRV_SPI_TX_FIFO_INTERRUPT_MODE.

5.1.9.7.4 Data Types and Constants**5.1.9.7.4.1 DRV_SPI_BAUD_RATE Macro**

C

```
#define DRV_SPI_BAUD_RATE 8000000
```

Description

SPI baud rate value configuration

This macro controls the baud rate value of the SPI driver.

Remarks

None.

5.1.9.7.4.2 DRV_SPI_BAUD_RATE_CLOCK Macro**C**

```
#define DRV_SPI_BAUD_RATE_CLOCK SPI_BAUD_RATE_CLOCK_MCLK
```

Description

SPI baud rate clock selection

This macro selects the SPI baud rate clock.

Remarks

None.

5.1.9.7.4.3 DRV_SPI_BUFFER_HANDLE_INVALID Macro**C**

```
#define DRV_SPI_BUFFER_HANDLE_INVALID ((DRV_SPI_BUFFER_HANDLE)(-1))
```

Description

SPI Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by [DRV_SPI_BufferAddRead\(\)](#) and [DRV_SPI_BufferAddWrite\(\)](#) function if the buffer add request was not successful.

Remarks

None

5.1.9.7.4.4 DRV_SPI_BUFFER_SIZE Macro**C**

```
#define DRV_SPI_BUFFER_SIZE 8
```

Description

This is macro DRV_SPI_BUFFER_SIZE.

5.1.9.7.4.5 DRV_SPI_BUFFER_USAGE_TYPE Macro**C**

```
#define DRV_SPI_BUFFER_USAGE_TYPE DRV_SPI_BUFFER_TYPE_STANDARD
```

Description

SPI buffer type configuration

This macro controls the buffer type of the SPI driver.

Remarks

None.

5.1.9.7.4.6 DRV_SPI_CLIENTS_NUMBER Macro**C**

```
#define DRV_SPI_CLIENTS_NUMBER 1
```

Description

SPI maximum number of clients

This definition selects the maximum number of clients that the SPI driver can support at run time.

Remarks

None.

5.1.9.7.4.7 DRV_SPI_CLOCK_OPERATION_MODE Macro**C**

```
#define DRV_SPI_CLOCK_OPERATION_MODE DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE
```

Description

SPI clock mode configuration

This macro controls the clock mode of the SPI driver.

Remarks

None.

5.1.9.7.4.8 DRV_SPI_COMMUNICATION_WIDTH Macro**C**

```
#define DRV_SPI_COMMUNICATION_WIDTH SPI_COMMUNICATION_8BIT_WIDE
```

Description

SPI communication width configuration

This macro controls the communication width of the SPI driver.

Remarks

None.

5.1.9.7.4.9 DRV_SPI_FRAME_SYNC_PULSE_COUNT Macro**C**

```
#define DRV_SPI_FRAME_SYNC_PULSE_COUNT SPI_FRAME_SYNC_PULSE_16_CHAR
```

Description

SPI frame sync pulse count selection

This macro selects the SPI frame sync pulse count.

Remarks

None.

5.1.9.7.4.10 DRV_SPI_FRAME_SYNC_PULSE_DIRECTION Macro

C

```
#define DRV_SPI_FRAME_SYNC_PULSE_DIRECTION SPI_FRAME_PULSE_DIRECTION_INPUT
```

Description

SPI frame sync pulse direction selection

This macro selects the SPI frame sync pulse direction.

Remarks

None.

5.1.9.7.4.11 DRV_SPI_FRAME_SYNC_PULSE_EDGE Macro

C

```
#define DRV_SPI_FRAME_SYNC_PULSE_EDGE SPI_FRAME_PULSE_EDGE_PRECEDES_FIRST_CLOCK
```

Description

SPI frame sync pulse edge selection

This macro selects the SPI frame sync pulse edge.

Remarks

None.

5.1.9.7.4.12 DRV_SPI_FRAME_SYNC_PULSE_POLARITY Macro

C

```
#define DRV_SPI_FRAME_SYNC_PULSE_POLARITY SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH
```

Description

SPI frame sync pulse polarity selection

This macro selects the SPI frame sync pulse polarity.

Remarks

None.

5.1.9.7.4.13 DRV_SPI_FRAME_SYNC_PULSE_WIDTH Macro

C

```
#define DRV_SPI_FRAME_SYNC_PULSE_WIDTH SPI_FRAME_PULSE_WIDTH_TO_ONE_WORD
```

Description

SPI frame sync pulse width selection

This macro selects the SPI frame sync pulse width.

Remarks

None.

5.1.9.7.4.14 DRV_SPI_INDEX Macro

C

```
#define DRV_SPI_INDEX DRV_SPI_INDEX_2
```

Description

SPI static index selection

This definition selects the SPI static index for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

5.1.9.7.4.15 DRV_SPI_INDEX_0 Macro

C

```
#define DRV_SPI_INDEX_0 0
```

Description

SPI Driver Module Index Numbers

These constants provide the SPI driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the [DRV_SPI_Initialize](#) and [DRV_SPI_Open](#) functions to identify the driver instance in use.

5.1.9.7.4.16 DRV_SPI_INDEX_COUNT Macro

C

```
#define DRV_SPI_INDEX_COUNT SPI_NUMBER_OF_MODULES
```

Description

SPI Driver Module Index Count

This constant identifies the number of valid SPI driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

5.1.9.7.4.17 DRV_SPI_INPUT_SAMPLE_PHASE Macro

C

```
#define DRV_SPI_INPUT_SAMPLE_PHASE SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE
```

Description

SPI input sample phase selection

This macro selects the SPI input sample phase.

Remarks

None.

5.1.9.7.4.18 DRV_SPI_INSTANCES_NUMBER Macro

C

```
#define DRV_SPI_INSTANCES_NUMBER 1
```

Description

SPI hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

None.

5.1.9.7.4.19 DRV_SPI_INTERRUPT_MODE Macro

C

```
#define DRV_SPI_INTERRUPT_MODE false
```

Description

SPI interrupt and polled mode operation control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of SPI operation is desired
 - false - Select if polling mode of SPI operation is desired
- Not defining this option to true or false will result in build error.

Remarks

None.

5.1.9.7.4.20 DRV_SPI_INTERRUPT_SOURCE_ERROR Macro

C

```
#define DRV_SPI_INTERRUPT_SOURCE_ERROR INT_SOURCE_SPI_1_ERROR
```

Description

SPI error interrupt source

Macro to define the error interrupt source in case of static driver.

Remarks

Refer to the INT PLIB document for more information on [INT_SOURCE](#) enumeration

5.1.9.7.4.21 DRV_SPI_INTERRUPT_SOURCE_RX Macro

C

```
#define DRV_SPI_INTERRUPT_SOURCE_RX INT_SOURCE_SPI_1_EVENT
```

Description

SPI receive interrupt source

This macro defines the receive interrupt source of the static driver.

Remarks

Refer to the Interrupt Peripheral Library documentation for more information on [INT_SOURCE](#) enumeration.

This should be specified in case the device does not have a dedicated receive interrupt source.

5.1.9.7.4.22 DRV_SPI_INTERRUPT_SOURCE_TX Macro**C**

```
#define DRV_SPI_INTERRUPT_SOURCE_TX INT_SOURCE_SPI_1_EVENT
```

Description

SPI transmit/receive or transmit alone interrupt source

This macro defines the transmit/receive or transmit alone interrupt source of the static driver.

Remarks

Refer to the Interrupt Peripheral Library documentation for more information on the [INT_SOURCE](#) enumeration.

Some devices have a single interrupt source for both transmit and receive communication, while some have a dedicated transmit interrupt source. These are handled through the DRV_SPI_INTERRUPT_SOURCE configuration switch.

5.1.9.7.4.23 DRV_SPI_PERIPHERAL_ID Macro**C**

```
#define DRV_SPI_PERIPHERAL_ID SPI_ID_1
```

Description

SPI peripheral ID selection

This macro selects the SPI peripheral ID selection. This is an initialization override of the spiID member of the initialization configuration.

Remarks

Some devices also support SPI_ID_0.

5.1.9.7.4.24 DRV_SPI_PORTS_REMAP_USAGE Macro**C**

```
#define DRV_SPI_PORTS_REMAP_USAGE false
```

Description

SPI pins remap selection

This macro selects the SPI pins remap. The possible values of this macro are:

- true - Remap is required
- false - Remap is not required

Remarks

None.

5.1.9.7.4.25 DRV_SPI_POWER_STATE Macro

C

```
#define DRV_SPI_POWER_STATE SYS_MODULE_POWER_IDLE_STOP
```

Description

SPI power state configuration

This macro controls the power state of the SPI driver.

Remarks

This feature may not be available in the device or the SPI module selected.

5.1.9.7.4.26 DRV_SPI_PROTOCOL Macro

C

```
#define DRV_SPI_PROTOCOL DRV_SPI_PROTOCOL_TYPE_STANDARD
```

Description

SPI protocol type configuration

This macro controls the protocol type of the SPI driver.

Remarks

None.

5.1.9.7.4.27 DRV_SPI_USAGE_MODE Macro

C

```
#define DRV_SPI_USAGE_MODE DRV_SPI_MODE_MASTER
```

Description

SPI usage mode configuration

This macro controls the usage mode of the SPI driver.

Remarks

None.

5.1.9.7.4.28 DRV_SPI_BUFFER_EVENT Enumeration

C

```
typedef enum {  
    DRV_SPI_BUFFER_EVENT_PENDING,  
    DRV_SPI_BUFFER_EVENT_COMPLETE,  
    DRV_SPI_BUFFER_EVENT_ERROR  
} DRV_SPI_BUFFER_EVENT;
```

Description

SPI Driver Buffer Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_NVM_BufferAddRead or DRV_NVM_BufferAddWrite functions.

Members

Members	Description
DRV_SPI_BUFFER_EVENT_PENDING	Buffer is pending to get processed
DRV_SPI_BUFFER_EVENT_COMPLETE	All data from or to the buffer was transferred successfully.
DRV_SPI_BUFFER_EVENT_ERROR	There was an error while processing the buffer transfer request.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_NVM_BufferEventHandlerSet function when a buffer transfer request is completed.

5.1.9.7.4.29 DRV_SPI_BUFFER_EVENT_HANDLER Type**C**

```
typedef void (* DRV_SPI_BUFFER_EVENT_HANDLER)(DRV_SPI_BUFFER_EVENT event,
DRV_SPI_BUFFER_HANDLE bufferHandle, uintptr_t context);
```

Description

SPI Driver Buffer Event Handler Function Pointer

This data type defines the required function signature for the SPI driver buffer event handling callback function. A client must register a pointer to a buffer event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values and return value are described here and a partial example implementation is provided.

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

Returns

None.

Remarks

If the event is DRV_SPI_BUFFER_EVENT_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_SPI_BUFFER_EVENT_ERROR, it means that the data was not transferred successfully.

The bufferHandle parameter contains the buffer handle of the buffer that failed.

The context parameter contains the a handle to the client context, provided at the time the event handling function was registered using the DRV_SPI_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in an interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive operations within this function.

Example

```
void APP_MyBufferEventHandler( DRV_SPI_BUFFER_EVENT event,
                             DRV_SPI_BUFFER_HANDLE bufferHandle,
                             uintptr_t context )
{
```

```

MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

switch(event)
{
    case DRV_SPI_BUFFER_EVENT_COMPLETE:

        // Handle the completed buffer.
        break;

    case DRV_SPI_BUFFER_EVENT_ERROR:
    default:

        // Handle error.
        break;
}
}

```

5.1.9.7.4.30 DRV_SPI_BUFFER_HANDLE Type

C

```
typedef uintptr_t DRV_SPI_BUFFER_HANDLE;
```

Description

SPI Driver Buffer Handle

A buffer handle value is returned by a call to the [DRV_SPI_BufferAddRead\(\)](#)/ [DRV_SPI_BufferAddWrite\(\)](#) or [DRV_SPI_BufferAddReadWrite\(\)](#) functions. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

5.1.9.7.4.31 DRV_SPI_BUFFER_TYPE Enumeration

C

```
typedef enum {
    DRV_SPI_BUFFER_TYPE_STANDARD,
    DRV_SPI_BUFFER_TYPE_ENHANCED
} DRV_SPI_BUFFER_TYPE;
```

Description

SPI Buffer Type Selection

This enumeration identifies the various buffer types of the SPI module.

Members

Members	Description
DRV_SPI_BUFFER_TYPE_STANDARD	SPI Buffer Type Standard
DRV_SPI_BUFFER_TYPE_ENHANCED	SPI Enhanced Buffer Type

Remarks

None.

5.1.9.7.4.32 DRV_SPI_CLIENT_SETUP Type

C

```
typedef struct _DRV_SPI_CLIENT_SETUP DRV_SPI_CLIENT_SETUP;
```

Description

SPI Client Initialization Data

This structure lists the elements needed to set up an SPI client. Used by the function [DRV_SPI_ClientSetup](#) function. This data type defines the data required to initialize an SPI client.

Remarks

None.

5.1.9.7.4.33 DRV_SPI_CLOCK_MODE Enumeration

C

```
typedef enum {
    DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE,
    DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL,
    DRV_SPI_CLOCK_MODE_IDLE_HIGH_EDGE_FALL,
    DRV_SPI_CLOCK_MODE_IDLE_HIGH_EDGE_RISE
} DRV_SPI_CLOCK_MODE;
```

Description

SPI Clock Mode Selection

This enumeration identifies the various clock modes of the SPI module.

Members

Members	Description
DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE	SPI Clock Mode 0 - Idle State Low & Sampling on Rising Edge
DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL	SPI Clock Mode 1 - Idle State Low & Sampling on Falling Edge
DRV_SPI_CLOCK_MODE_IDLE_HIGH_EDGE_FALL	SPI Clock Mode 2 - Idle State High & Sampling on Falling Edge
DRV_SPI_CLOCK_MODE_IDLE_HIGH_EDGE_RISE	SPI Clock Mode 3 - Idle State High & Sampling on Rising Edge

Remarks

None.

5.1.9.7.4.34 DRV_SPI_INIT Type

C

```
typedef struct _DRV_SPI_INIT DRV_SPI_INIT;
```

Description

SPI Driver Initialization Data

This data type defines the data required to initialize or reinitialize the SPI driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

5.1.9.7.4.35 DRV_SPI_MODE Enumeration

C

```
typedef enum {  
    DRV_SPI_MODE_MASTER,  
    DRV_SPI_MODE_SLAVE  
} DRV_SPI_MODE;
```

Description

SPI Usage Modes Enumeration

This enumeration identifies the various usage modes of the SPI module.

Members

Members	Description
DRV_SPI_MODE_MASTER	SPI Mode Master
DRV_SPI_MODE_SLAVE	SPI Mode Slave

Remarks

None.

5.1.9.7.4.36 DRV_SPI_PROTOCOL_TYPE Enumeration

C

```
typedef enum {  
    DRV_SPI_PROTOCOL_TYPE_STANDARD,  
    DRV_SPI_PROTOCOL_TYPE_FRAMED,  
    DRV_SPI_PROTOCOL_TYPE_AUDIO  
} DRV_SPI_PROTOCOL_TYPE;
```

Description

SPI Protocols Enumeration

This enumeration identifies the various protocols of the SPI module.

Members

Members	Description
DRV_SPI_PROTOCOL_TYPE_STANDARD	SPI Protocol Type Standard
DRV_SPI_PROTOCOL_TYPE_FRAMED	SPI Protocol Type Framed
DRV_SPI_PROTOCOL_TYPE_AUDIO	SPi Protocol Type Audio

Remarks

None.

5.1.9.8 Files

Files

Name	Description
drv_spi.h	SPI device driver interface file.
drv_spi_config_template.h	SPI Driver configuration definitions template.

Description

5.1.9.8.1 drv_spi.h

SPI Driver Interface

The SPI driver provides a simple interface to manage the SPI module. This file defines the interface definitions and prototypes for the SPI driver.

Enumerations

	Name	Description
	<u>DRV_SPI_BUFFER_EVENT</u>	Identifies the possible events that can result from a buffer add request.
	<u>DRV_SPI_BUFFER_TYPE</u>	Identifies the various buffer types of the SPI module.
	<u>DRV_SPI_CLOCK_MODE</u>	Identifies the various clock modes of the SPI module.
	<u>DRV_SPI_MODE</u>	Identifies the various usage modes of the SPI module.
	<u>DRV_SPI_PROTOCOL_TYPE</u>	Identifies the various protocols of the SPI module.

Functions

	Name	Description
	<u>DRV_SPI_BufferAddRead</u>	Registers a buffer for a read operation. Actual transfer will happen in the Task function.
	<u>DRV_SPI_BufferAddWrite</u>	Registers a buffer for a read operation. Actual transfer will happen in the Task function.
	<u>DRV_SPI_BufferAddWriteRead</u>	Registers a buffer for a read operation. Actual transfer will happen in the Task function.
	<u>DRV_SPI_BufferStatus</u>	Returns the transmitter and receiver transfer status.
	<u>DRV_SPI_ClientSetup</u>	Initializes a client.
	<u>DRV_SPI_Close</u>	Closes an opened instance of the SPI driver
	<u>DRV_SPI_Deinitialize</u>	De-initializes the specified instance of the SPI driver module.
	<u>DRV_SPI_Initialize</u>	Initializes the SPI driver.
	<u>DRV_SPI_Open</u>	Opens the specified SPI driver instance and returns a handle to it.
	<u>DRV_SPI_Read</u>	Gets/Reads byte(s) from SPI. The function will wait until the specified number of bytes is received.
	<u>DRV_SPI_Status</u>	Provides the current status of the SPI driver module.
	<u>DRV_SPI_Tasks</u>	Maintains the driver's state machine and implements its ISR.
	<u>DRV_SPI_VersionGet</u>	Gets the SPI driver version in numerical format.
	<u>DRV_SPI_VersionStrGet</u>	Gets the SPI driver version in string format.
	<u>DRV_SPI_Write</u>	Writes the data to SPI. The function will wait until all the bytes are transmitted.

Macros

	Name	Description
	<u>DRV_SPI_BUFFER_HANDLE_INVALID</u>	Definition of an invalid buffer handle.
	<u>DRV_SPI_INDEX_0</u>	SPI driver index definitions.
	<u>DRV_SPI_INDEX_1</u>	This is macro DRV_SPI_INDEX_1.
	<u>DRV_SPI_INDEX_2</u>	This is macro DRV_SPI_INDEX_2.
	<u>DRV_SPI_INDEX_3</u>	This is macro DRV_SPI_INDEX_3.
	<u>DRV_SPI_INDEX_4</u>	This is macro DRV_SPI_INDEX_4.

	DRV_SPI_INDEX_5	This is macro DRV_SPI_INDEX_5.
	DRV_SPI_INDEX_COUNT	Number of valid SPI driver indices.

Types

	Name	Description
	DRV_SPI_BUFFER_EVENT_HANDLER	Pointer to a SPI Driver Buffer Event handler function
	DRV_SPI_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_SPI_CLIENT_SETUP	Defines the data required to initialize an SPI client.
	DRV_SPI_INIT	Defines the data required to initialize or reinitialize the SPI driver

File Name

drv_spi.h

Company

Microchip Technology Inc.

5.1.9.8.2 drv_spi_config_template.h

SPI Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

Macros

	Name	Description
	DRV_SPI_BAUD_RATE	Controls the baud rate value of the SPI driver.
	DRV_SPI_BAUD_RATE_CLOCK	Selects the SPI baud rate clock.
	DRV_SPI_BUFFER_SIZE	This is macro DRV_SPI_BUFFER_SIZE.
	DRV_SPI_BUFFER_USAGE_TYPE	Controls the buffer type of the SPI driver.
	DRV_SPI_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_SPI_CLOCK_OPERATION_MODE	Controls the clock mode of the SPI driver.
	DRV_SPI_COMMUNICATION_WIDTH	Controls the communication width of the SPI driver.
	DRV_SPI_FRAME_SYNC_PULSE_COUNT	Selects the SPI frame sync pulse count.
	DRV_SPI_FRAME_SYNC_PULSE_DIRECTION	Selects the SPI frame sync pulse direction.
	DRV_SPI_FRAME_SYNC_PULSE_EDGE	Selects the SPI frame sync pulse edge.
	DRV_SPI_FRAME_SYNC_PULSE_POLARITY	Selects the SPI frame sync pulse polarity.
	DRV_SPI_FRAME_SYNC_PULSE_WIDTH	Selects the SPI frame sync pulse width.
	DRV_SPI_INDEX	SPI static index selection.
	DRV_SPI_INPUT_SAMPLE_PHASE	Selects the SPI input sample phase.
	DRV_SPI_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver .
	DRV_SPI_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_SPI_INTERRUPT_SOURCE_ERROR	Defines the error interrupt source in case of static driver
	DRV_SPI_INTERRUPT_SOURCE_RX	Defines the receive interrupt source in case of static driver.
	DRV_SPI_INTERRUPT_SOURCE_TX	Defines the transmit/receive or transmit alone interrupt source in case of static driver.
	DRV_SPI_PERIPHERAL_ID	SPI peripheral ID selection.
	DRV_SPI_PORTS_REMAP_USAGE	Selects the SPI pins remap.
	DRV_SPI_POWER_STATE	Controls the power state of the SPI driver.

	DRV_SPI_PROTOCOL	Controls the protocol type of the SPI driver.
	DRV_SPI_RX_FIFO_INTERRUPT_MODE	This is macro DRV_SPI_RX_FIFO_INTERRUPT_MODE.
	DRV_SPI_TX_FIFO_INTERRUPT_MODE	This is macro DRV_SPI_TX_FIFO_INTERRUPT_MODE.
	DRV_SPI_USAGE_MODE	Controls the usage mode of the SPI driver.

File Name

drv_spi_config_template.h

Company

Microchip Technology Inc.

5.1.10 Timer Driver Library

5.1.10.1 Introduction

Timer Driver Library

for

Microchip Microcontrollers

This library provides an interface to manage the Timer module on the Microchip family of microcontrollers during different modes of operation.

Description**Overview**

Timers are useful for generating accurate time based periodic interrupts for software application or real time operating systems. Other uses include counting external pulses or accurate timing measurement of external events using the timer's gate functions and accurate hardware delays.

Please refer to the the Data Sheet for the part that is being used.

Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software.

5.1.10.2 Release Notes

MPLAB Harmony Version: v0.70b **Timer Driver Library Version :** 0.02 Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.1.10.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.10.4 Using the Library

This topic describes the basic architecture of the Timer Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_tmr.h](#)

The interface to the Timer Driver Library is defined in the "[drv_tmr.h](#)" header file.

Please refer to the [MPLAB Harmony Overview](#) for how the driver interacts with the framework.

5.1.10.4.1 Abstraction Model

Microchip devices have two type of timers:

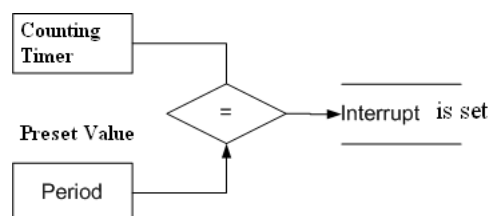
- Period-Match - When the counting timer matches the configured period, this timer is said to complete one iteration and an interrupt is set
- Overflow - When the counting timer overflows/rolls over, interrupt is set and restarts from its previously set counter value. In this case, the preset period value would be the maximum possible count value.

The Timer Driver abstracts this difference and provides a unified model for both of these types of timers.

Description

Abstraction Model

The abstraction model of the Timer Driver is explained in the following diagram:



The core functionality of the Timer allows access to both the counter and the period values.

The unified model allows user not to worry about whether the Timer currently being used is a overflow or period match-based. The user simply updates the period and the driver automatically takes care of adjusting the respective Timer's periodicity. In case a user needs any modification into the counter's initial value (i.e., to adjust for any errors in periodicity), they are allowed to do so using the counter control interfaces.

5.1.10.4.2 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

The table below lists the interface section and its brief description.

Library Interface Section	Description
Configuration	Provides macros for configuring the system. It is required that the system configures the driver to build correctly by choosing appropriate configuration options as listed in this section. These macros enable different features or modes of the timer peripheral.
System Interaction Functions	Provides interfaces to system layer to initialize, deinitialize and reinitialize the module. This section also describes functions to query the status of the module.
Core Functions	Provides interfaces for core functionality of the driver.
Alarm Functions	Provides interfaces to handle alarm features, if alarm functionality is enabled.
Period Functions	Provides interfaces to control the periodicity of the timers.
Counter Control Functions	Provides interfaces to update the counter values.
Miscellaneous Functions	Provides interfaces to get the version information, timer tick and operating frequencies.

5.1.10.4.3 How the Library Works

The library provides interfaces to support:

- System Interaction
- Sync Mode Setup
- Period Modification
- Counter Modification
- Client Core Functionality
- Client Alarm Functionality (optional function, enabled using configuration options)
- Other Optional Functionality (enabled using configuration options)

Note: Any code segment pertaining to the driver interfaces will work for both the static or dynamic configurations. It is not necessary to modify the code to move from one configuration to the other (i.e., from static or dynamic or static-multi).

5.1.10.4.3.1 System Interaction

This section describes Timer initialization and reinitialization.

Description

Initialization and Reinitialization

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or re-initialized. During system initialization, each instance of the Timer module will be initialized with the following configuration settings (either passed dynamically at run time using [DRV_TMR_INIT](#) or by using initialization overrides) that are supported by the specific Timer module hardware:

Initialization member	Configuration Name	Description
moduleInit	DRV_TMR_POWER_STATE	System module initialization of the power state.
tmrId	DRV_TMR_PERIPHERAL_ID	Timer hardware module ID (peripheral library-level ID).
clockSource	DRV_TMR_CLOCK_SOURCE	Clock Source Selection, Enables either the internal or the external clock, using one of the possible values from TMR_CLOCK_SOURCE .
timerPeriod	DRV_TMR_TIMER_PERIOD	Timer period value.
prescale	DRV_TMR_PRESCALE	Sets up the prescaler, using one of the possible values from TMR_PRESCALE .
sourceEdge	DRV_TMR_SOURCE_EDGE	Source edge selection, if supported by the hardware, selects either the rising or falling edge, using one of the possible values from TMR_CLOCK_SOURCE_EDGE .
postscale	DRV_TMR_POSTSCALE	Sets up the postscaler, if supported by the hardware, using one of the possible values from TMR_POSTSCALE .
syncMode	DRV_TMR_SYNCHRONIZATION_MODE	Selects the operation mode of the instance using one of the possible values from DRV_TMR_SYNC_MODE .
interruptSource	DRV_TMR_INTERRUPT_SOURCE	Interrupt source identifier for the Timer module being initialized, which is available from the respective device's processor headers.

The [DRV_TMR_Initialize](#) function returns an object handle of the type [SYS_MODULE_OBJ](#). After this, the object handle returned by the Initialize interface would be used by the other system interfaces like [DRV_TMR_Reinitialize](#), [DRV_TMR_Deinitialize](#), [DRV_TMR_Status](#) and [DRV_TMR_Tasks](#).

Example: Timer Initialization Through [DRV_TMR_INIT](#)

```

DRV_TMR_INIT      init;
SYS_MODULE_OBJ    object;
SYS_STATUS        tmrStatus;

// populate the TMR init configuration structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId            = TMR_ID_2;
init.clockSource      = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale         = TMR_PRESCALE_TX_VALUE_256;
init.syncMode         = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource  = INT_SOURCE_TIMER_2;
init.timerPeriod      = 0xABCD;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);

tmrStatus = DRV_TMR_Status(object);

if ( SYS_STATUS_READY != tmrStatus)
{
    // Handle error
}

```

Example: Timer initialization Through Configuration

Source file:

```
SYS_MODULE_OBJ    object;
SYS_STATUS        tmrStatus;
```

```
int main( void )
{
    object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)NULL);

    tmrStatus = DRV_TMR_Status(object);

    if ( SYS_STATUS_READY != tmrStatus)
    {
        // Handle error
    }
}
```

Configuration file: sys_config.h

```
#define DRV_TMR_POWER_STATE          SYS_MODULE_POWER_RUN_FULL
#define DRV_TMR_PERIPHERAL_ID        TMR_ID_3 // TMR_MODULE_ID
#define DRV_TMR_CLOCK_SOURCE          TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK
#define DRV_TMR_PRESCALE              TMR_PRESCALE_TX_VALUE_256
#define DRV_TMR_SYNCHRONIZATION_MODE DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL
#define DRV_TMR_INTERRUPT_SOURCE      INT_SOURCE_TIMER_3
#define DRV_TMR_TIMER_PERIOD          0x0000F424 // 200ms
```

Deinitialization

Once the Initialize operation has been called, the Deinitialize operation must be called before the initialize operation can be called again. This routine may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the routine will never block for hardware Timer access. If the operation requires time to allow the hardware to complete, this will be reported by the [DRV_TMR_Status](#) operation.

Status

Timer status is available to query the module state before, during, and after initialization, deinitialization, and reinitialization.

Tasks Routine

The interface [DRV_TMR_Tasks](#) can be called in two ways:

- By the system task service in a polled environment
- By the interrupt service routine of the timer in an interrupt based system

Example: Polling

```
int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_TMR_Initialize( DRV_TMR_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_TMR_Status( object ) )
        return 0;

    while (1)
    {
        DRV_TMR_Tasks (object);
    }
}
```

Example: Interrupt

```
int main( void )
{
```

```

SYS_MODULE_OBJ object;
object = DRV_TMR_Initialize( DRV_TMR_INDEX_0, (SYS_MODULE_INIT *) &initConf );

if( SYS_STATUS_READY != DRV_TMR_Status( object ) )
    return 0;

while (1);
}

/* Sample interrupt routine not specific to any device family */
void ISR T1Interrupt(void)
{
    //Call the Timer Tasks routine
    DRV_TMR_Tasks(object);
}

```

5.1.10.4.3.2 Client Interaction

This section describes general client operation.

Description

General Client Operation

For the application to begin using an instance of the Timer module, it must call the [DRV_TMR_Open](#) function. This provides the configuration required to open the Timer instance for operation. If the driver is deinitialized using the function [DRV_TMR_Deinitialize](#), the application must call the [DRV_TMR_Open](#) function again to set up the instance of the Timer. The function [DRV_TMR_Open](#) need not be called again if the system is reinitialized using the [DRV_TMR_Reinitialize](#) function.

The Timer Driver supports only the 'DRV_IO_INTENT_EXCLUSIVE' IO_INTENT.

Example:

```

DRV_HANDLE handle;

// Configure the instance DRV_TMR_INDEX_1 with the configuration
handle = DRV_TMR_Open(DRV_TMR_INDEX_1, DRV_IO_INTENT_EXCLUSIVE);

if( handle == DRV_HANDLE_INVALID )
{
    // Client cannot open the instance.
}

```

The function [DRV_TMR_Close](#) closes an already opened instance of the Timer Driver, invalidating the handle. [DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Example:

```

DRV_HANDLE handle;

// Configure the instance DRV_TMR_INDEX_1 with the configuration
handle = DRV_TMR_Open(DRV_TMR_INDEX_1, DRV_IO_INTENT_EXCLUSIVE);

/*...*/

DRV_TMR_Close( handle );

```

The client has the option to check the status through the interface [DRV_TMR_ClientStatus](#).

Example:

```

DRV_HANDLE handle;

// Configure the instance DRV_TMR_INDEX_1 with the configuration
handle = DRV_TMR_Open(DRV_TMR_INDEX_1, DRV_IO_INTENT_EXCLUSIVE);

```

```
if ( DRV_TMR_CLIENT_STATUS_READY != DRV_TMR_ClientStatus( handle ) )
    return 0;
```

5.1.10.4.3.3 Sync Mode Selection

This section describes the Sync mode selection process.

Description

Sync Mode Selection

The Timer Driver can operate in various modes as described below. They can be changed using [DRV_TMR_SyncModeSet](#), and the current mode being using by the Timer can be retrieved using [DRV_TMR_SyncModeGet](#).

DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL - Synchronous Internal Clock Counter

- This mode can be used to provide the following capabilities:
 - Create elapsed time measurements
 - Create time delays
 - Create periodic timer interrupts
- The input clock to the Timer is provided from the internal system clock
- The Timer increments in the effective delta time specified by the prescaler and the selected source clock

DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL_GATED - Synchronous Internal Gated Counter

- In this mode, the operation is dependent on an external signal applied to a pin assigned as a gate. When the gate signal remains high, the Timer increments. When the gate signal goes low, the operation terminates.
- Input clock to the timer is provided from the internal system clock
- The Timer increments in the effective delta time specified by the prescaler and the selected source clock
- Gated Timer mode is not possible when the Timer is set to use external clock source

DRV_TMR_SYNC_MODE_SYNCHRONOUS_EXTERNAL_WITHOUT_CLKSYNC - Synchronous External Clock Counter

- The Synchronous External Clock Counter operation provides the following capabilities:
 - Counting periodic or non-periodic pulses
 - Uses the external clock as the time base for Timers
- The selected Timer increments on every rising edge of the clock input on the TxCK pin

DRV_TMR_SYNC_MODE_SYNCHRONOUS_EXTERNAL_WITH_CLKSYNC - Synchronous External Clock Counter

- The clock source for the Timer is provided externally
- The selected Timer increments on every rising edge of the input on the TxCK pin
- External Clock sync is enabled to operate in the Synchronous mode

DRV_TMR_SYNC_MODE_ASYNCHRONOUS_EXTERNAL_WITHOUT_CLKSYNC - Asynchronous External Clock Counter

- The Asynchronous Timer operation provides the following capabilities:
 - The Timer can operate during Sleep mode and can generate an interrupt on a period register match that will wake the processor from Sleep or Idle modes.
 - The Timer can be clocked from the secondary oscillator for real-time clock applications
- The clock source for the Timer is provided externally
- The selected Timer increments on every rising edge of the clock input on the TxCK pin
- Synchronization with the external clock is disabled

DRV_TMR_SYNC_MODE_ASYNCHRONOUS_EXTERNAL_WITH_CLKSYNC - Asynchronous External Clock Counter

- The clock source for the Timer is provided externally
- The selected Timer increments on every rising edge of the clock input on the TxCK pin
- Synchronization with the external clock is enabled

Changing the Sync Mode

Perform the following steps before changing the sync mode.

Preconditions: [DRV_TMR_Open](#) should have been called before calling this function and should have returned a valid handle.

1. Stop the TMR module using [DRV_TMR_Stop](#).
2. Change the sync mode using [DRV_TMR_SyncModeSet](#).
3. Update the counter using [DRV_TMR_Counter\[8Bit/16Bit/32Bit\]Set](#), if applicable.
4. Start the TMR module using [DRV_TMR_Start](#).

Example:

```
DRV_HANDLE handle;
/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

/* Stop the timer */
DRV_TMR_Stop( handle );
/* Change the sync mode */
DRV_TMR_SyncModeSet( handle, DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL );
/* Re-start the timer */
DRV_TMR_Start( handle );
```

Note: Not all modes are supported by all the Timer modules. Please refer the specific device data sheet to determine the supported modes for your device.

5.1.10.4.3.4 Period Modification

This section describes Period modification for the different types of Timers (i.e., 8-/16-/32-bit).

Description

These set of functions help modify the Timer periodicity at the client level, regardless of whether it is an overflow or a period match-based Timer.

This interface can be used to alter the already set periodicity at the system level interface [DRV_TMR_Initialize](#).

Period Modification

Periodicity for the type of Timer (8-/16-/32-bit) can be controlled as follows:

- 8-bit timer periodicity can be modified using [DRV_TMR_Period8BitSet](#) and the current period value can be obtained using [DRV_TMR_Period8BitGet](#)
- 16-bit timer periodicity can be modified using [DRV_TMR_Period16BitSet](#) and the current period value can be obtained using [DRV_TMR_Period16BitGet](#)
- 32-bit timer periodicity can be modified using [DRV_TMR_Period32BitSet](#) and the current period value can be obtained using [DRV_TMR_Period32BitGet](#)

Example:

```
DRV_HANDLE handle;
/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

/* ... */
```



```
/* Update the new period */
DRV_TMR_Period16BitSet( handle, 0x0000C350);
```

5.1.10.4.3.5 Counter Modification

This section describes counter modification for the different types of Timers (i.e., 8-/16-/32-bit).

Description

These set of functions help modify the initial value of the Timer counters to help adjust any errors in the periodicity.

Counter Modification

Initial counter values can be controlled as follows:

- 8-bit timer initial value can be modified using [DRV_TMR_Counter8BitSet](#) and the current counter value can be obtained using [DRV_TMR_Counter8BitGet](#)
- 16-bit timer initial value can be modified using [DRV_TMR_Counter16BitSet](#) and the current counter value can be obtained using [DRV_TMR_Counter16BitGet](#)
- 32-bit timer initial value can be modified using [DRV_TMR_Counter32BitSet](#) and the current counter value can be obtained using [DRV_TMR_Counter32BitGet](#)

Example:

```
DRV_HANDLE handle;
/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

/* ... */

/* Update the counter value */
/* Following code updates the initial value from 0x0000 to 0x0010
   to cover up any error in the previously set periodicity */

DRV_TMR_Counter16BitSet( handle, 0x0010);
```

5.1.10.4.3.6 Core Functionality

This section describes core functionality of the Timer Driver.

Description

Core functionality provides a extremely basic interface for the driver operation.

Applications using the core Timer core functionality, need to perform the following:

1. The system should have completed the necessary initialization and [DRV_TMR_Tasks](#) should either be running in a polled environment, or in an interrupt environment.
2. Open the driver using [DRV_TMR_Open](#). The Timer Driver supports exclusive access only.
3. The Timer period will have to be updated using [DRV_TMR_Period\[8Bit/16Bit/32Bit\]Set](#) if the client intends to override the already preset value during the initialization. The previously set value can be retrieved using [DRV_TMR_Period\[8Bit/16Bit/32Bit\]Get](#).
4. Start the driver using [DRV_TMR_Start](#).
5. Poll for the elapse status using [DRV_TMR_ElapsedStatusGetAndClear](#) and handle the rest of the processing based on the status.
6. The client will be able to stop the started Timer instance using [DRV_TMR_Stop](#) at any time and will be able to close it using [DRV_TMR_Close](#) when it is no longer required.

Example :

```
DRV_HANDLE handle;
```

```

/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

while (1)
{
    // Interrupt mode & hence the DRV_TMR_Tasks not being called

    // client handle returned by the Open API
    if (true == DRV_TMR_ElapsedStatusGetAndClear(handle))
    {
        // Do something
    }
}

```

Notes:

1. The user needs to stop the Timer before any updates on the counter or period and restart it later.
2. The Timer elapsed status gets reset after any counter or period update.
3. The Timer elapse status remains unchanged if the user tries to stop and restart later.

5.1.10.4.3.7 Alarm Functionality

This section describes the Timer Driver alarm functionality.

Description

The Timer Driver provides alarm functionality. This is an optional advanced feature and needs to be enabled using the configuration macros.

Applications using the Timer alarm functionality, need to perform the following:

1. The system should have completed the necessary initialization and [DRV_TMR_Tasks](#) should either be running in a polled environment, or in an interrupt environment.
2. Open the driver using [DRV_TMR_Open](#). The Timer Driver supports exclusive access only.
3. Configure the alarm using [DRV_TMR_AlarmSet](#) with [DRV_TMR_ALARM_CONFIG](#).
4. Start the driver using [DRV_TMR_Start](#).
5. If a callback is configured, the Timer Driver will call the callback function when the alarm expires.
6. The client will be able to stop the started Timer module instance using [DRV_TMR_Stop](#) at any time and will be able to close it using [DRV_TMR_Close](#) when it is no longer required.

Precondition: For the callback to work, the switch [DRV_TMR_ALARM_ENABLE](#) needs to be defined in the configuration header.

Example:

```

DRV_HANDLE          handle;
DRV_TMR_ALARM_CONFIG alarmConfig;
uint32_t            alarmCount;

/* Open the client */
handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

/* Configure the timer alarm feature */
/* Callbacks are only allowed when the configuration switch
   DRV_TMR_ALARM_ENABLE is defined */
alarmConfig.type = DRV_TMR_ALARM_TYPE_PERIODIC;
alarmConfig.callback = &Callback;
DRV_TMR_AlarmSet (handle, &alarmConfig);

// Get all the alarm counters based on the number of alarm objects configured

```

```
alarmCount = DRV_TMR_AlarmCountGet (handle);

// Clear the Alarm counters associated with the HW instance
DRV_TMR_AlarmCountClear (handle);
```

The callback function in the application will look like:

```
void Callback (void)
{
    // Do something specific on an alarm event trigger
}
```

Note: The driver tasks function calls the client registered callback after the alarm expires.

5.1.10.4.3.8 Optional Interfaces

This section describes additional/optional client interfaces.

Description

Additional/Optional client interfaces include the following.

Get Operating Frequency

The function [DRV_TMR_OperatingFrequencyGet](#) provides the client with the information on the Timer operating frequency.

Example:

```
DRV_HANDLE handle;
uint32_t freq;

/* Open the client */
handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

freq = DRV_TMR_OperatingFrequencyGet (handle);
```

Get Tick Frequency

The function [DRV_TMR_TickFrequencyGet](#) provides the client with the information on the Timer tick frequency.

Example:

```
DRV_HANDLE handle;
uint32_t freq;

/* Open the client */
handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

freq = DRV_TMR_TickFrequencyGet (handle);
```

5.1.10.4.3.9 Example Usage of the Timer Driver

This section describes typical usage of the Timer Driver for various Timer modules in polling/interrupt advanced/core modes.

Description

Notes:

1. The interface calls remain the same whether using dynamic or static drivers.
2. There is a possibility that the user can pass NULL to the driver initialize interface. However, the respective configurations matching the init [[DRV_TMR_INIT](#)] members need to be configured in the correct manner.

Example 1:

```
// Polling mode under 32-bit count mode for a PIC32 device using the alarm feature
```

```

SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;
DRV_TMR_ALARM_CONFIG alConf;

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId             = TMR_ID_2;
init.clockSource       = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale          = TMR_PRESCALE_TX_VALUE_256;
init.syncMode          = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource   = INT_SOURCE_TIMER_3;
init.timerPeriod       = 0xFF00;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_STATUS_READY != DRV_TMR_ClientStatus(handle))
    return 0;

alConf.alType = DRV_TMR_ALARM_TYPE_PERIODIC;
alConf.alCallback = &TestAlarmCallback;
DRV_TMR_AlarmSet (handle, &alConf);

DRV_TMR_Start (handle);

while (1)
{
    DRV_TMR_Tasks (object);
}

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_STATUS_INVALID != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Deinitialize (object);
// end main

void TestAlarmCallback (void)
{
    // Do something specific on an alarm event trigger
}

```

Example 2:

// Polling mode under 16-bit count mode for a PIC32/PIC24 device using the alarm feature

```

SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;
DRV_TMR_ALARM_CONFIG alConf;

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId             = TMR_ID_3;
init.clockSource       = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale          = TMR_PRESCALE_TX_VALUE_256;
init.syncMode          = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource   = INT_SOURCE_TIMER_3;
init.timerPeriod       = 0xFF00;

```

```

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_STATUS_READY != DRV_TMR_ClientStatus(handle))
    return 0;

alConf.alType = DRV_TMR_ALARM_TYPE_PERIODIC;
alConf.alCallback = &TestAlarmCallback;
DRV_TMR_AlarmSet (handle, &alConf);

DRV_TMR_Start (handle);

while (1)
{
    DRV_TMR_Tasks (object);
}

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_STATUS_INVALID != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Deinitialize (object);
// end main

void TestAlarmCallback (void)
{
    // Do something specific on an alarm event trigger
}

```

Example 3:

// Polling mode under 16-bit count mode for a PIC18 device, with Timer3 using the alarm feature

```

SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;
DRV_TMR_ALARM_CONFIG alConf;

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId             = TMR_ID_3;
init.clockSource       = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale          = TMR_PRESCALE_T1_VALUE_8;
init.syncMode          = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource   = INT_SOURCE_TIMER_3;
init.timerPeriod       = 0xFF00;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_STATUS_READY != DRV_TMR_ClientStatus(handle))
    return 0;

alConf.alType = DRV_TMR_ALARM_TYPE_PERIODIC;
alConf.alCallback = &TestAlarmCallback;
DRV_TMR_AlarmSet (handle, &alConf);

DRV_TMR_Start (handle);

while (1)

```

```

{
    DRV_TMR_Tasks (object);
}

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_STATUS_INVALID != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Deinitialize (object);
// end main

void TestAlarmCallback (void)
{
    // Do something specific on an alarm event trigger
}

```

Example 4:

// Interrupt mode under 32-bit count mode for a PIC32 device using the alarm feature

```

SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;
DRV_TMR_ALARM_CONFIG alConf;

SYS_INT_Initialize ();

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId             = TMR_ID_2;
init.clockSource       = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale          = TMR_PRESCALE_TX_VALUE_256;
init.syncMode          = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource   = INT_SOURCE_TIMER_3;
init.timerPeriod       = 0xFF00;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_STATUS_READY != DRV_TMR_ClientStatus(handle))
    return 0;

alConf.alType = DRV_TMR_ALARM_TYPE_PERIODIC;
alConf.alCallback = &TestAlarmCallback;
DRV_TMR_AlarmSet (handle, &alConf);

SYS_INT_PrioritySet (PLIB_INT_SOURCE_TIMER_3, PLIB_INT_PRIORITY_LEVEL1);
SYS_INT_SubprioritySet (PLIB_INT_SOURCE_TIMER_3, PLIB_INT_SUBPRIORITY_LEVEL1);

DRV_TMR_Start (handle);

while (1);

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_STATUS_INVALID != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Deinitialize (object);
// end main

```

```

void TestAlarmCallback (void)
{
    // Do something specific on an alarm event trigger
}

```

Example 5:

// Interrupt mode under 16-bit count mode for a PIC24 device using the alarm feature

```

SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;
DRV_TMR_ALARM_CONFIG alConf;

SYS_INT_Initialize ();

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId             = TMR_ID_2;
init.clockSource       = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale          = TMR_PRESCALE_TX_VALUE_256;
init.syncMode          = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource   = INT_SOURCE_TIMER_2;
init.timerPeriod       = 0xFF00;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_STATUS_READY != DRV_TMR_ClientStatus(handle))
    return 0;

alConf.alType = DRV_TMR_ALARM_TYPE_PERIODIC;
alConf.alCallback = &TestAlarmCallback;
DRV_TMR_AlarmSet (handle, &alConf);

SYS_INT_PrioritySet (PLIB_INT_SOURCE_TIMER_2, PLIB_INT_PRIORITY_LEVEL1);

DRV_TMR_Start (handle);

while (1);

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_STATUS_INVALID != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Deinitialize (object);
//end main

void TestAlarmCallback (void)
{
    // Do something specific on an alarm event trigger
}

```

Example 6:

// Polling mode under 32-bit count mode for a PIC32 device using core functionality

```

SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;

```

```

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId             = TMR_ID_2;
init.clockSource       = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale          = TMR_PRESCALE_TX_VALUE_256;
init.syncMode          = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource   = INT_SOURCE_TIMER_3;
init.timerPeriod       = 0xFF00;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_STATUS_READY != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Start (handle);

while (1)
{
    DRV_TMR_Tasks (object);
    if (true == DRV_TMR_ElapsedStatusGetAndClear(handle))
        Callback();
}

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_STATUS_INVALID != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Deinitialize (object);
// end main

void Callback (void)
{
    // Do something specific
}

```

Example 7:

// Interrupt mode under 32-bit count mode for a PIC32 device using core functionality

```

SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;
DRV_TMR_ALARM_CONFIG alConf;

SYS_INT_Initialize ();

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId             = TMR_ID_2;
init.clockSource       = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale          = TMR_PRESCALE_TX_VALUE_256;
init.syncMode          = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource   = INT_SOURCE_TIMER_3;
init.timerPeriod       = 0xFF00;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_STATUS_READY != DRV_TMR_ClientStatus(handle))

```



```

    return 0;

SYS_INT_PrioritySet (PLIB_INT_SOURCE_TIMER_3, PLIB_INT_PRIORITY_LEVEL1);
SYS_INT_SubprioritySet (PLIB_INT_SOURCE_TIMER_3, PLIB_INT_SUBPRIORITY_LEVEL1);

DRV_TMR_Start (handle);

while (1)
{
    if (true == DRV_TMR_ElapsedStatusGetAndClear(handle))
        Callback();
}

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_STATUS_INVALID != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Deinitialize (object);
// end main

void Callback (void)
{
    // Do something specific
}

```

Example 8:

// Polling mode under 16-bit count mode for a PIC18 device, with Timer3 using core functionality

```

SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;
DRV_TMR_ALARM_CONFIG alConf;

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId             = TMR_ID_3;
init.clockSource       = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale          = TMR_PRESCALE_T1_VALUE_8;
init.syncMode          = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource   = INT_SOURCE_TIMER_3;
init.timerPeriod       = 0xFF00;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_STATUS_READY != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Start (handle);

while (1)
{
    DRV_TMR_Tasks (object);
    if (true == DRV_TMR_ElapsedStatusGetAndClear(handle))
        Callback();
}

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_STATUS_INVALID != DRV_TMR_ClientStatus(handle))

```

```

    return 0;

DRV_TMR_Deinitialize (object);
// end main

void Callback (void)
{
    // Do something specific
}

```

5.1.10.5 Configuring the Library

The configuration of the Timer Device Library is based on the file sys_config.h

This header file contains the configuration selection for the Timer Driver Library build. Based on the selections made here and the system setup, the Timer Driver driver may or may not support selected features. These configuration settings will apply to all instances of the driver.

This header can be placed anywhere in the application-specific folders and the path of this header needs to be presented to the include search for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.10.5.1 Initialization Overrides

Macros

	Name	Description
	DRV_TMR_CLOCK_SOURCE	Timer module clock source selection.
	DRV_TMR_PERIPHERAL_ID	Timer peripheral ID selection.
	DRV_TMR_POSTSCALE	Macro controls post scale of the TMR
	DRV_TMR_PRESCALE	Timer module prescale value.
	DRV_TMR_SOURCE_EDGE	Timer module source edge.
	DRV_TMR_SYNCHRONIZATION_MODE	Controls Synchronization mode of the Timer.
	DRV_TMR_TIMER_PERIOD	Timer period value for an overflow or a period match-based Timer.

Description

This section provides initialization overrides for the run time options that can be selected using [DRV_TMR_INIT](#) when passed into [DRV_TMR_Initialize](#) and [DRV_TMR_Reinitialize](#). When any of these options are defined, the build time override will be used.

5.1.10.5.1.1 DRV_TMR_CLOCK_SOURCE Macro

C

```
#define DRV_TMR_CLOCK_SOURCE TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK
```

Description

TMR Clock Source

This macro selects the Timer module clock source based on the enumeration.

Remarks

If defined in the configuration, this takes priority over the initialization option.

Refer to the processor peripheral header for more information.

5.1.10.5.1.2 DRV_TMR_PERIPHERAL_ID Macro

C

```
#define DRV_TMR_PERIPHERAL_ID TMR_ID_1
```

Description

TMR Peripheral ID Selection

This macro selects the Timer peripheral ID. This is an initialization override of the `tmrID` member of the initialization configuration.

Remarks

Some devices also support `TMR_ID_0`.

5.1.10.5.1.3 DRV_TMR_POSTSCALE Macro

C

```
#define DRV_TMR_POSTSCALE TMR_POSTSCALE_1_TO_1
```

Description

TMR post scale configuration

This macro controls the post scale of the TMR

Remarks

This feature is not available in all modules/devices. Refer to the specific device data sheet for more information.

5.1.10.5.1.4 DRV_TMR_PRESCALE Macro

C

```
#define DRV_TMR_PRESCALE TMR_PRESCALE_TX_1_TO_256
```

Description

TMR Prescale

This macro defines the Timer module prescale value.

Remarks

If defined in the configuration, this takes priority over the initialization option.

Refer to the processor peripheral header for more information.

5.1.10.5.1.5 DRV_TMR_SOURCE_EDGE Macro

C

```
#define DRV_TMR_SOURCE_EDGE TMR_SOURCE_EDGE_INCREMENT_ON_LOW_TO_HIGH
```

Description

TMR Source Edge

This macro defines the Timer module source edge.

Remarks

If defined in the configuration, this takes priority over the initialization option.

Refer to the processor peripheral header for more information.

5.1.10.5.1.6 DRV_TMR_SYNCHRONIZATION_MODE Macro

C

```
#define DRV_TMR_SYNCHRONIZATION_MODE DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL
```

Description

TMR Synchronization mode configuration

This macro controls the Synchronization mode of the Timer. It accepts one of the possible values of the enumeration [DRV_TMR_SYNC_MODE](#).

Remarks

If defined in the configuration, this takes priority over the initialization option.

5.1.10.5.1.7 DRV_TMR_TIMER_PERIOD Macro

C

```
#define DRV_TMR_TIMER_PERIOD 0x4567
```

Description

TMR period value

This macro defines the Timer period value for an overflow or a period match-based Timer.

Remarks

If defined in the configuration, this macro takes priority over the initialization option. In addition, the limit is the maximum possible Timer count width.

The user is required to provide the count value. The driver will internally adjust the value as required for the Timer hardware based on the overflow or period match functionality

5.1.10.5.2 System Config

Macros

	Name	Description
	DRV_TMR_COUNT_WIDTH	Controls the mode of the Timer.
	DRV_TMR_INDEX	Timer static index selection.
	DRV_TMR_INDEX_0	Timer driver index definitions
	DRV_TMR_INDEX_1	This is macro DRV_TMR_INDEX_1.
	DRV_TMR_INDEX_2	This is macro DRV_TMR_INDEX_2.
	DRV_TMR_INDEX_3	This is macro DRV_TMR_INDEX_3.
	DRV_TMR_INDEX_4	This is macro DRV_TMR_INDEX_4.
	DRV_TMR_INDEX_5	This is macro DRV_TMR_INDEX_5.
	DRV_TMR_INDEX_COUNT	Number of valid Timer driver indices.
	DRV_TMR_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_TMR_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.

Description

5.1.10.5.2.1 DRV_TMR_COUNT_WIDTH Macro

C

```
#define DRV_TMR_COUNT_WIDTH 16
```

Description

TMR mode configuration

This macro controls the mode of the Timer, and selects whether the timer is to support 8-bit, 16-bit or 32-bit mode. The acceptable values of this configuration are:

- 8, (Use this for 8-bit timers)
- 16 (Use this for 16-bit timers)
- 32 (Use this for 32-bit timers)

Remarks

This is a mandatory configuration option, which enables the Timer count width.

5.1.10.5.2.2 DRV_TMR_INDEX Macro

C

```
#define DRV_TMR_INDEX DRV_TMR_INDEX_2
```

Description

TMR Static Index Selection

This definition selects the Timer Static Index for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

5.1.10.5.2.3 DRV_TMR_INDEX_0 Macro

C

```
#define DRV_TMR_INDEX_0 0
```

Description

Timer Driver Module Index Numbers

These constants provide Timer driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the [DRV_TMR_Initialize](#) and [DRV_TMR_Open](#) functions to identify the driver instance in use.

5.1.10.5.2.4 DRV_TMR_INDEX_1 Macro

C

```
#define DRV_TMR_INDEX_1 1
```

Description

This is macro DRV_TMR_INDEX_1.

5.1.10.5.2.5 DRV_TMR_INDEX_2 Macro

C

```
#define DRV_TMR_INDEX_2 2
```

Description

This is macro DRV_TMR_INDEX_2.

5.1.10.5.2.6 DRV_TMR_INDEX_3 Macro

C

```
#define DRV_TMR_INDEX_3 3
```

Description

This is macro DRV_TMR_INDEX_3.

5.1.10.5.2.7 DRV_TMR_INDEX_4 Macro

C

```
#define DRV_TMR_INDEX_4 4
```

Description

This is macro DRV_TMR_INDEX_4.

5.1.10.5.2.8 DRV_TMR_INDEX_5 Macro

C

```
#define DRV_TMR_INDEX_5 5
```

Description

This is macro DRV_TMR_INDEX_5.

5.1.10.5.2.9 DRV_TMR_INDEX_COUNT Macro

C

```
#define DRV_TMR_INDEX_COUNT TMR_NUMBER_OF_MODULES
```

Description

Timer Driver Module Index Count

This constant identifies Timer driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is device-specific.

5.1.10.5.2.10 DRV_TMR_INSTANCES_NUMBER Macro

C

```
#define DRV_TMR_INSTANCES_NUMBER 1
```

Description

Hardware instances support

This definition sets up the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

None

5.1.10.5.2.11 DRV_TMR_INTERRUPT_MODE Macro**C**

```
#define DRV_TMR_INTERRUPT_MODE true
```

Description

TMR Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of timer operation is desired
- false - Select if polling mode of timer operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

5.1.10.5.3 Alarm Config**Macros**

	Name	Description
	DRV_TMR_ALARM_ENABLE	Enables alarm functionality.
	DRV_TMR_ALARM_PERIODIC	Alarm type configuration.

Description**5.1.10.5.3.1 DRV_TMR_ALARM_ENABLE Macro****C**

```
#define DRV_TMR_ALARM_ENABLE
```

Description

Alarm feature configuration option

This feature enables the alarm functionality inside the Timer driver.

Remarks

This is an optional feature. If the user wants to be called after a time elapse, alarm callback registration is mandatory.

5.1.10.5.3.2 DRV_TMR_ALARM_PERIODIC Macro

C

```
#define DRV_TMR_ALARM_PERIODIC true
```

Description

Alarm type configuration option

This macro if defined, makes the alarm periodic. This macro can [accept](#) the following values:

- true - Configures the alarm in periodic mode
- false - Configures the alarm in one shot mode

Remarks

This is an optional feature, which is enabled when using this configuration.

5.1.10.5.4 Client Config

Macros

	Name	Description
	DRV_TMR_CLIENTS_NUMBER	Selects the maximum number of clients

Description

5.1.10.5.4.1 DRV_TMR_CLIENTS_NUMBER Macro

C

```
#define DRV_TMR_CLIENTS_NUMBER 1
```

Description

Client support

This definition select the maximum number of clients that the Timer driver can support at run time.

Remarks

None.

5.1.10.5.5 Miscellaneous Config

Macros

	Name	Description
	DRV_TMR_ASYNC_WRITE_ENABLE	Controls Asynchronous Write mode of the Timer.
	DRV_TMR_PRESCALER_ENABLE	Controls prescaler assignment of the Timer.

Description

These configuration options control the functionality and setup of the Timer Driver.

5.1.10.5.5.1 DRV_TMR_ASYNC_WRITE_ENABLE Macro

C

```
#define DRV_TMR_ASYNC_WRITE_ENABLE false
```

Description

TMR Asynchronous write mode configuration

This macro controls the Asynchronous Write mode of the Timer. This macro accepts the following values:

- true - Configures the Timer to enable async write control
- false - Configures the Timer to disable async write control
- [DRV_CONFIG_NOT_SUPPORTED](#) - When the feature is not supported on the instance.

Remarks

This feature is not available in all modules/devices. Refer to the specific device data sheet for more information.

5.1.10.5.5.2 DRV_TMR_PRESCALER_ENABLE Macro

C

```
#define DRV_TMR_PRESCALER_ENABLE false
```

Description

TMR prescaler assignment configuration

This macro controls the prescaler assignment of the TMR. This macro accepts the following values:

- true - Configures the Timer for enable prescaler assignment
- false - Configures the Timer for disable prescaler assignment
- [DRV_CONFIG_NOT_SUPPORTED](#) - When the feature is not supported on the instance

Remarks

This feature is not available in all modules/devices. Refer to the specific device data sheet for more information.

5.1.10.5.6 Alarm Functionality Examples

Files

Name	Description
drv_tmr_config_alarm.h	Timer driver configuration definitions for the alarm feature.

Description

5.1.10.5.6.1 drv_tmr_config_alarm.h

TMR Driver Configuration Definitions for Alarm feature

These definitions set up the driver for the alarm feature.

File Name

drv_tmr_config_alarm.h

Company

Microchip Technology Inc.

5.1.10.6 Building the Library

This section list the files that are available in the \src of the timer driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.1.10.7 Library Interface

Data Types and Constants

	Name	Description
	DRV_TMR_ALARM_CONFIG	Timer module alarm configuration.
	DRV_TMR_ALARM_TYPE	Timer driver type enumeration.
	DRV_TMR_CALLBACK	Pointer to a Timer driver callback function data type.
	DRV_TMR_CLIENT_STATUS	Identifies the client-specific status of the Timer driver
	DRV_TMR_INIT	Defines the Timer driver initialization data.
	DRV_TMR_SYNC_MODE	Identifies the synchronous modes for the Timer driver.

Macros

	Name	Description
	DRV_TMR_INTERRUPT_SOURCE	Defines the interrupt source of the static driver.
	DRV_TMR_POWER_STATE	Controls the power state of the Timer.
	DRV_TMR_UNIFIED	Sets up the driver functionality in either split or the unified mode.

Alarm Functions

	Name	Description
	DRV_TMR_AlarmCountClear	Clears the alarm elapse counter.
	DRV_TMR_AlarmCountGet	Provides the number of times the alarm has elapsed since the last time it was cleared.
	DRV_TMR_AlarmSet	Initializes an alarm.

Core Functions

	Name	Description
	DRV_TMR_ClientStatus	Gets current client-specific status of the Timer driver.
	DRV_TMR_Close	Closes an opened instance of the Timer driver
	DRV_TMR_ElapsedStatusGetAndClear	Provides and clears the Timer's elapse count.
	DRV_TMR_Open	Opens the specified Timer driver instance and returns a handle to it.
	DRV_TMR_Start	Starts the Timer counting.
	DRV_TMR_Stop	Stops the Timer from counting.
	DRV_TMR_SyncModeGet	Provides the Timer's current synchronization mode.
	DRV_TMR_SyncModeSet	Changes the Timer's synchronization mode.

Counter Control Functions

	Name	Description
≡	DRV_TMR_Counter16BitGet	Provides the 16-bit Timer's current counter value.
≡	DRV_TMR_Counter16BitSet	Updates the 16-bit Timer's initial count value.
≡	DRV_TMR_Counter32BitGet	Provides the 32-bit Timer's current counter value.
≡	DRV_TMR_Counter32BitSet	Updates the 32-bit Timer's initial count value.
≡	DRV_TMR_Counter8BitGet	Provides the 8-bit Timer's current counter value.
≡	DRV_TMR_Counter8BitSet	Updates the 8-bit Timer's initial count value

Miscellaneous Functions

	Name	Description
≡	DRV_TMR_OperatingFrequencyGet	Provides the Timer operating frequency.
≡	DRV_TMR_TickFrequencyGet	Provides the Timer's counter "tick" frequency.
≡	DRV_TMR_VersionGet	Gets the Timer driver version in numerical format.
≡	DRV_TMR_VersionStrGet	Gets the Timer driver version in string format.

Period Functions

	Name	Description
≡	DRV_TMR_Period16BitGet	Provides the 16-bit Timer's period
≡	DRV_TMR_Period16BitSet	Updates the 16-bit Timer's period.
≡	DRV_TMR_Period32BitGet	Provides the 32-bit Timer's period
≡	DRV_TMR_Period32BitSet	Updates the 32-bit Timer's period.
≡	DRV_TMR_Period8BitGet	Provides the 8-bit Timer's period.
≡	DRV_TMR_Period8BitSet	Updates the 8-bit timer's period

System Interaction Functions

	Name	Description
≡	DRV_TMR_Deinitialize	Deinitializes the specified instance of the Timer driver.
≡	DRV_TMR_Initialize	Initializes the Timer driver .
≡	DRV_TMR_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings.
≡	DRV_TMR_Status	Provides the current status of the Timer driver.
≡	DRV_TMR_Tasks	Maintains the driver's state machine and implements its ISR.

Description

This section describes the functions of the Timer Driver Library.

Refer to each section for a detailed description.

5.1.10.7.1 System Interaction Functions**5.1.10.7.1.1 DRV_TMR_Deinitialize Function****C**

```
void DRV_TMR_Deinitialize(
    SYS_MODULE_OBJ object
);
```

Description

Deinitializes the specified instance of the Timer driver, disabling its operation (and any hardware). All internal data is invalidated.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called before calling this function and a valid [SYS_MODULE_OBJ](#) must have been returned.

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TMR_Initialize

Returns

None.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the [DRV_TMR_Status](#) operation. The system has to use [DRV_TMR_Status](#) to find out when the module is in the ready state.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_TMR_Initialize
SYS_STATUS        status;

DRV_TMR_Deinitialize(object);

status = DRV_TMR_Status(object);
if (SYS_MODULE_DEINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

5.1.10.7.1.2 DRV_TMR_Initialize Function**C**

```
SYS_MODULE_OBJ DRV_TMR_Initialize(
    const SYS_MODULE_INDEX drvIndex,
    const SYS_MODULE_INIT * const init
);
```

Description

This function initializes the Timer driver, making it ready for clients to open and use it.

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver. This pointer may be null if the user wants to provide the information via the respective configuration macros.

Returns

If successful, returns a valid handle to a driver object. Otherwise, it returns `SYS_MODULE_OBJ_INVALID`. The returned object must be passed as argument to `DRV_TMR_Reinitialize`, `DRV_TMR_Deinitialize`, `DRV_TMR_Tasks` and `DRV_TMR_Status` functions.

Remarks

This function must be called before any other Timer function is called.

This function should only be called once during system initialization unless `DRV_TMR_Deinitialize` is called to deinitialize the driver instance.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the `DRV_TMR_Status` operation. The system must use `DRV_TMR_Status` to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Example

```
DRV_TMR_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the timer initialization structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId            = TMR_ID_2;
init.clockSource      = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale         = TMR_PRESCALE_TX_VALUE_256;
init.syncMode         = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource  = INT_SOURCE_TIMER_2;
init.timerPeriod      = 0xABCD;

// Do something

objectHandle = DRV_TMR_Initialize(DRV_TMR_INDEX_0, (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

5.1.10.7.1.3 DRV_TMR_Reinitialize Function

C

```
void DRV_TMR_Reinitialize(
    SYS_MODULE_OBJ object,
    const SYS_MODULE_INIT* const init
);
```

Description

This function reinitializes the driver and refreshes any associated hardware settings using the initialization data given, but it will not interrupt any ongoing operations.

Preconditions

The `DRV_TMR_Initialize` function must have been called before calling this function and a valid `SYS_MODULE_OBJ` must have been returned.

Parameters

Parameters	Description
object	Driver object handle, returned from <code>DRV_TMR_Initialize</code>

init	Pointer to the initialization data structure

Returns

None

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the [DRV_TMR_Status](#) operation. The system must use [DRV_TMR_Status](#) to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Example

```

DRV_TMR_INIT    init;
SYS_MODULE_OBJ  objectHandle;  // Returned from DRV_TMR_Initialize
SYS_STATUS      tmrStatus;

// Iopulate the timer initialization structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId             = TMR_ID_2;
init.clockSource       = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale          = TMR_PRESCALE_TX_VALUE_256;
init.syncMode          = DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL;
init.interruptSource    = INT_SOURCE_TIMER_3;
init.timerPeriod       = 0xABCD;

DRV_TMR_Reinitialize(objectHandle, (SYS_MODULE_INIT*)&init);

tmrStatus = DRV_TMR_Status(object);
if (SYS_STATUS_BUSY == tmrStatus)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= tmrStatus)
{
    // Handle error
}

```

5.1.10.7.1.4 DRV_TMR_Status Function**C**

```

SYS_STATUS DRV_TMR_Status(
    SYS_MODULE_OBJ object
);

```

Description

This function provides the current status of the Timer driver.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TMR_Initialize

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to [accept](#) new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

The this operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Example

```

SYS_MODULE_OBJ    object;    // Returned from DRV_TMR_Initialize
SYS_STATUS        tmrStatus;

tmrStatus = DRV_TMR_Status(object);
else if (SYS_STATUS_ERROR >= tmrStatus)
{
    // Handle error
}

```

5.1.10.7.1.5 DRV_TMR_Tasks Function**C**

```

void DRV_TMR_Tasks(
    SYS_MODULE_OBJ object
);

```

Description

This function is used to maintain the driver's internal state machine and implement its ISR for interrupt-driven implementations.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called for the specified Timer driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TMR_Initialize)

Returns

None

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)) or by the appropriate raw ISR.

This function may excute in an ISR context and will never block or access any resources that may cause it to block.

Example

```
SYS_MODULE_OBJ      object;      // Returned from DRV_TMR_Initialize

while (true)
{
    DRV_TMR_Tasks (object);

    // Do other tasks
}
```

5.1.10.7.2 Core Functions

5.1.10.7.2.1 DRV_TMR_ClientStatus Function

C

```
DRV_TMR_CLIENT_STATUS DRV_TMR_ClientStatus(
    DRV_HANDLE handle
);
```

Description

This function gets the client-specific status of the Timer driver associated with the specified handle.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

A [DRV_TMR_CLIENT_STATUS](#) value describing the current status of the driver

Remarks

This function will not block for hardware access and will immediately return the current status.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
DRV_TMR_CLIENT_STATUS tmrClientStatus;

tmrClientStatus = DRV_TMR_ClientStatus(tmrHandle);
if(DRV_TMR_CLIENT_STATUS_ERROR >= tmrClientStatus)
{
    // Handle the error
}
```


5.1.10.7.2.2 DRV_TMR_Close Function

C

```
void DRV_TMR_Close(  
    DRV_HANDLE handle  
)  
;
```

Description

This function closes an opened instance of the Timer driver, invalidating the handle.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called for the specified Timer driver instance.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

None

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver functions. A new handle must be obtained by calling [DRV_TMR_Open](#) before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior, the call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the [DRV_TMR_Status](#) operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open  
  
DRV_TMR_Close(handle);
```

5.1.10.7.2.3 DRV_TMR_ElapsedStatusGetAndClear Function

C

```
bool DRV_TMR_ElapsedStatusGetAndClear(  
    DRV_HANDLE handle  
)  
;
```

Description

This function provides the Timer's elapse count and resets it to zero (0).

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid handle, returned from the DRV_TMR_Open

Returns

- true - The Timer has elapsed
- false - The Timer has not yet elapsed

Remarks

None.

Example

```
DRV_HANDLE    handle; // Returned from DRV_TMR_Open
bool          elapseStatus;

elapseStatus = DRV_TMR_ElapsedStatusGetAndClear(handle);
if (elapseStatus > 0)
{
    // The counter period has elapsed.
}
```

5.1.10.7.2.4 DRV_TMR_Open Function**C**

```
DRV_HANDLE DRV_TMR_Open(
    const SYS_MODULE_INDEX drvIndex,
    const DRV_IO_INTENT intent
);
```

Description

This function opens the specified Timer driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Returns

If successful, the function returns a valid open instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is [DRV_HANDLE_INVALID](#).

Remarks

The handle returned is valid until the [DRV_TMR_Close](#) function is called.

This function will NEVER block waiting for hardware.

If the DRV_IO_INTENT_BLOCKING is requested and the driver was built appropriately to support blocking behavior, other client-level operations may block waiting on hardware until they are complete.

If DRV_IO_INTENT_NON_BLOCKING is requested the driver client can call the [DRV_TMR_ClientStatus](#) operation to find out

when the module is in the ready state.

If the requested intent flags are not supported, the function will return [DRV_HANDLE_INVALID](#).

The Timer driver does not support DRV_IO_INTENT_SHARED. It must be opened with DRV_IO_INTENT_EXCLUSIVE in the "intent" parameter.

Example

```
DRV_HANDLE handle;

handle = DRV_TMR_Open(DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

5.1.10.7.2.5 DRV_TMR_Start Function

C

```
void DRV_TMR_Start(
    DRV_HANDLE handle
);
```

Description

This function starts the Timer counting.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open

DRV_TMR_Start (handle);
```

5.1.10.7.2.6 DRV_TMR_Stop Function

C

```
void DRV_TMR_Stop(
    DRV_HANDLE handle
);
```

Description

This function stops the running Timer from counting.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
DRV_TMR_Stop(handle);
```

5.1.10.7.2.7 DRV_TMR_SyncModeGet Function

C

```
DRV_TMR_SYNC_MODE DRV_TMR_SyncModeGet(
    DRV_HANDLE handle
);
```

Description

This function provides the Timer's current synchronization mode.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

Possible values from the enumeration [DRV_TMR_SYNC_MODE](#).

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
DRV_TMR_SYNC_MODE mode;

mode = DRV_TMR_SyncModeGet(handle);
```

5.1.10.7.2.8 DRV_TMR_SyncModeSet Function

C

```
void DRV_TMR_SyncModeSet(
    DRV_HANDLE handle,
    DRV_TMR_SYNC_MODE newMode
);
```

Description

This function changes the Timer's synchronization mode.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
mode	Possible values from the enumeration DRV_TMR_SYNC_MODE

Returns

None.

Remarks

The caller should stop the Timer/Counter before changing the synchronization mode and restart it afterward.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open

DRV_TMR_Stop(handle);
DRV_TMR_SyncModeSet(handle, DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL);
```

5.1.10.7.3 Alarm Functions

5.1.10.7.3.1 DRV_TMR_AlarmCountClear Function

C

```
void DRV_TMR_AlarmCountClear(
    DRV_HANDLE handle
);
```

Description

This function clears the alarm elapse counter.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
```

```
DRV_TMR_AlarmCountClear(handle);
```

5.1.10.7.3.2 DRV_TMR_AlarmCountGet Function

C

```
unsigned int DRV_TMR_AlarmCountGet(  
    DRV_HANDLE handle  
);
```

Description

This function provides the number of times the alarm is generated since the last time it was cleared.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open

Returns

Alarm count value.

Remarks

None.

Example

```
DRV_HANDLE handle;           // Returned from DRV_TMR_Open  
unsigned int alarmCount;  
  
alarmCount = DRV_TMR_AlarmCountGet (handle);
```

5.1.10.7.3.3 DRV_TMR_AlarmSet Function

C

```
void DRV_TMR_AlarmSet(  
    DRV_HANDLE handle,  
    const DRV_TMR_ALARM_CONFIG * config  
);
```

Description

This function initializes an alarm, allowing the client to receive a callback from the driver when the counter period elapses. Alarms can be one-shot or periodic.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open
config	Pointer to the alarm configuration structure containing any data necessary to initialize timer alarms

Returns

None

Remarks

None.

Example

```
DRV_HANDLE      handle; // Returned from DRV_TMR_Open
DRV_TMR_ALARM_CONFIG  alarmConfig;

alarmConfig.type      = DRV_TMR_ALARM_TYPE_PERIODIC; // Periodic Alarm
alarmConfig.callback = &callBackFunction;           // callback registered

DRV_TMR_AlarmSet(handle, config);
```

5.1.10.7.4 Period Functions

5.1.10.7.4.1 DRV_TMR_Period16BitGet Function

C

```
uint16_t DRV_TMR_Period16BitGet(
    DRV_HANDLE handle
);
```

Description

This function provides the 16-bit tTmer's period.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

16-bit timer period value

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
uint16_t value;
value = DRV_TMR_Period16BitGet(handle);
```

5.1.10.7.4.2 DRV_TMR_Period16BitSet Function

C

```
void DRV_TMR_Period16BitSet(
    DRV_HANDLE handle,
    uint16_t value
);
```

Description

This function updates the 16-bit Timer's period.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
value	16-bit Period value

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open

DRV_TMR_Period16BitSet(handle, 0x1000);
```

5.1.10.7.4.3 DRV_TMR_Period32BitGet Function**C**

```
uint32_t DRV_TMR_Period32BitGet(
    DRV_HANDLE handle
);
```

Description

This function provides the 32-bit Timer's period.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

32-bit Timer period value.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
uint32_t value;
value = DRV_TMR_Period32BitGet(handle);
```


5.1.10.7.4.4 DRV_TMR_Period32BitSet Function

C

```
void DRV_TMR_Period32BitSet(  
    DRV_HANDLE handle,  
    uint32_t value  
);
```

Description

This function updates the 32-bit Timer's period.

Preconditions

The [DRV_TMR_Initialize](#) uncton must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
value	32-bit Period value

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open  
  
DRV_TMR_Period32BitSet(handle, 0xFFFFFFFF);
```

5.1.10.7.4.5 DRV_TMR_Period8BitGet Function

C

```
uint8_t DRV_TMR_Period8BitGet(  
    DRV_HANDLE handle  
);
```

Description

This function provides the 8-bit Timer's period.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

8-bit timer period value.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
uint8_t value;
value = DRV_TMR_Period8BitGet(handle);
```

5.1.10.7.4.6 DRV_TMR_Period8BitSet Function**C**

```
void DRV_TMR_Period8BitSet(
    DRV_HANDLE handle,
    uint8_t value
);
```

Description

This function updates the 8-bit timer's period.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
value	8-bit Period value

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open

DRV_TMR_Period8BitSet(handle, 0x10);
```

5.1.10.7.5 Counter Control Functions**5.1.10.7.5.1 DRV_TMR_Counter16BitGet Function****C**

```
uint16_t DRV_TMR_Counter16BitGet(
    DRV_HANDLE handle
);
```

Description

This function provides the 16-bit Timer's current counter value.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

16-bit Timer's current counter value

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
uint16_t value;
value = DRV_TMR_Counter16BitGet(handle);
```

5.1.10.7.5.2 DRV_TMR_Counter16BitSet Function**C**

```
void DRV_TMR_Counter16BitSet(
    DRV_HANDLE handle,
    uint16_t value
);
```

Description

This function updates the 16-bit Timer's initial count value.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
value	16-bit Counter value

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open

DRV_TMR_Counter16BitSet(handle, 0x1000);
```

5.1.10.7.5.3 DRV_TMR_Counter32BitGet Function**C**

```
uint32_t DRV_TMR_Counter32BitGet(
    DRV_HANDLE handle
);
```

Description

This function provides the 32-bit Timer's current counter value.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

32-bit Timer's current counter value.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
uint32_t value;
value = DRV_TMR_Counter32BitGet(handle);
```

5.1.10.7.5.4 DRV_TMR_Counter32BitSet Function**C**

```
void DRV_TMR_Counter32BitSet(
    DRV_HANDLE handle,
    uint32_t value
);
```

Description

This function updates the 32-bit Timer's initial count value.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
value	32-bit Counter value

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open

DRV_TMR_Counter32BitSet(handle, 0xFFFFFFFF);
```

5.1.10.7.5.5 DRV_TMR_Counter8BitGet Function**C**

```
uint8_t DRV_TMR_Counter8BitGet(
```

```
    DRV_HANDLE handle  
);
```

Description

This function provides the 8-bit Timer's current counter value.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

8-bit Timer's current counter value.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open  
uint8_t value;  
value = DRV_TMR_Counter8BitGet(handle);
```

5.1.10.7.5.6 DRV_TMR_Counter8BitSet Function

C

```
void DRV_TMR_Counter8BitSet(  
    DRV_HANDLE handle,  
    uint8_t value  
);
```

Description

This function updates the 8-bit Timer's initial count value.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
value	8-bit Counter value

Returns

None.

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open  
  
DRV_TMR_Counter8BitSet(handle, 0x10);
```

5.1.10.7.6 Miscellaneous Functions

5.1.10.7.6.1 DRV_TMR_OperatingFrequencyGet Function

C

```
uint32_t DRV_TMR_OperatingFrequencyGet(  
    DRV_HANDLE handle  
);
```

Description

This function provides the Timer operating frequency.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

32-bit value corresponding to the running frequency.

Remarks

On most processors, the Timer's base frequency is the same as the peripheral bus clock.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open  
uint32_t clkFreq;  
  
clkFreq = DRV_TMR_OperatingFrequencyGet(handle);
```

5.1.10.7.6.2 DRV_TMR_TickFrequencyGet Function

C

```
uint32_t DRV_TMR_TickFrequencyGet(  
    DRV_HANDLE handle  
);
```

Description

This function provides the Timer's counter "tick" frequency.

Preconditions

The [DRV_TMR_Initialize](#) function must have been called.

[DRV_TMR_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid handle, returned from the DRV_TMR_Open

Returns

Clock frequency value in cycles per second (Hertz).

Remarks

None.

Example

```
DRV_HANDLE handle; // Returned from DRV_TMR_Open
uint32_t clkFreq;

clkFreq = DRV_TMR_TickFrequencyGet(handle);
```

5.1.10.7.6.3 DRV_TMR_VersionGet Function**C**

```
unsigned int DRV_TMR_VersionGet(
    const SYS_MODULE_INDEX drvIndex
);
```

Description

This function gets the Timer driver version. The version is encoded as major * 10000 + minor * 100 + patch. The stringed version can be obtained using [DRV_TMR_VersionStrGet\(\)](#)

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for

Returns

Current driver version in numerical format.

Remarks

None.

Example

```
unsigned int version;

version = DRV_TMR_VersionGet( DRV_TMR_INDEX_0 );

if(version < 110200)
{
    // Do Something
}
```

5.1.10.7.6.4 DRV_TMR_VersionStrGet Function**C**

```
char * DRV_TMR_VersionStrGet(
    const SYS_MODULE_INDEX drvIndex
);
```

Description

This function gets the Timer driver version. The version is returned as major.minor.patch[type], where type is optional. The numerical version can be obtained using [DRV_TMR_VersionGet](#).

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for

Returns

Current Timer driver version in the string format.

Remarks

None.

Example

```
char *version;

version = DRV_TMR_VersionStrGet( DRV_TMR_INDEX_0 );

printf("%s", version);
```

5.1.10.7.7 Data Types and Constants

5.1.10.7.7.1 DRV_TMR_ALARM_CONFIG Structure

C

```
typedef struct {
    DRV_TMR_ALARM_TYPE type;
    DRV_TMR_CALLBACK callback;
} DRV_TMR_ALARM_CONFIG;
```

Description

TMR Alarm configuration

This data type controls the configuration of the alarm.

Members

Members	Description
DRV_TMR_ALARM_TYPE type;	Type of the alarm one shot or periodic
DRV_TMR_CALLBACK callback;	Alarm callback

Remarks

Not all modes are available on all devices.

5.1.10.7.7.2 DRV_TMR_ALARM_TYPE Enumeration

C

```
typedef enum {
    DRV_TMR_ALARM_TYPE_ONE_SHOT,
    DRV_TMR_ALARM_TYPE_PERIODIC
} DRV_TMR_ALARM_TYPE;
```

Description

Timer Alarm Type Enumeration

This data type identifies the Timer alarm type.

Members

Members	Description
DRV_TMR_ALARM_TYPE_ONE_SHOT	One Shot Alarm
DRV_TMR_ALARM_TYPE_PERIODIC	Periodic Alarm

Remarks

Not all modes are available on all devices.

5.1.10.7.7.3 DRV_TMR_CALLBACK Type**C**

```
typedef void (* DRV_TMR_CALLBACK)(void);
```

Description

Timer Driver Callback Function Pointer

This data type defines a pointer to a Timer driver callback function.

Remarks

Useful only when timer alarm callback support is enabled by defining the [DRV_TMR_ALARM_ENABLE](#) configuration option.

5.1.10.7.7.4 DRV_TMR_CLIENT_STATUS Enumeration**C**

```
typedef enum {
    DRV_TMR_CLIENT_STATUS_INVALID,
    DRV_TMR_CLIENT_STATUS_ERROR,
    DRV_TMR_CLIENT_STATUS_BUSY,
    DRV_TMR_CLIENT_STATUS_READY,
    DRV_TMR_CLIENT_STATUS_STOPPED,
    DRV_TMR_CLIENT_STATUS_STARTED
} DRV_TMR_CLIENT_STATUS;
```

Description

Timer Driver Client Status

This enumeration identifies the client-specific status of the Timer driver.

Members

Members	Description
DRV_TMR_CLIENT_STATUS_INVALID	Client in an invalid (or unopened) state
DRV_TMR_CLIENT_STATUS_ERROR	Unspecified error condition
DRV_TMR_CLIENT_STATUS_BUSY	An operation is currently in progress
DRV_TMR_CLIENT_STATUS_READY	Up and running, no operations running
DRV_TMR_CLIENT_STATUS_STOPPED	Timer stopped (not running), but ready to accept commands
DRV_TMR_CLIENT_STATUS_STARTED	Timer started and running, processing transactions

Remarks

None.

5.1.10.7.7.5 DRV_TMR_INIT Structure

C

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    TMR_MODULE_ID tmrId;
    TMR_CLOCK_SOURCE clockSource;
    uint32_t timerPeriod;
    TMR_PRESCALE prescale;
    TMR_CLOCK_SOURCE_EDGE sourceEdge;
    uint8_t postscale;
    DRV_TMR_SYNC_MODE syncMode;
    INT_SOURCE interruptSource;
} DRV_TMR_INIT;
```

Description

Timer Driver Initialize Data

This data type defines data required to initialize or reinitialize the Timer driver.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
TMR_MODULE_ID tmrId;	Identifies timer hardware module (PLIB-level) ID
TMR_CLOCK_SOURCE clockSource;	Clock Source select enumeration
uint32_t timerPeriod;	Timer Period Value
TMR_PRESCALE prescale;	Prescaler Selection from the processor enumeration
TMR_CLOCK_SOURCE_EDGE sourceEdge;	TMR Source Edge Selection
uint8_t postscale;	Post Scale Selection
DRV_TMR_SYNC_MODE syncMode;	Timer Sync Mode
INT_SOURCE interruptSource;	Interrupt Source for TMR module

Remarks

Not all initialization features are available on all devices.

5.1.10.7.7.6 DRV_TMR_SYNC_MODE Enumeration

C

```
typedef enum {
    DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL,
    DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL_GATED,
    DRV_TMR_SYNC_MODE_SYNCHRONOUS_EXTERNAL_WITHOUT_CLKSYNC,
    DRV_TMR_SYNC_MODE_SYNCHRONOUS_EXTERNAL_WITH_CLKSYNC,
    DRV_TMR_SYNC_MODE_ASYNCHRONOUS_EXTERNAL_WITHOUT_CLKSYNC,
    DRV_TMR_SYNC_MODE_ASYNCHRONOUS_EXTERNAL_WITH_CLKSYNC,
    DRV_TMR_SYNC_MODE_IDLE
} DRV_TMR_SYNC_MODE;
```

Description

Timer Synchronous Modes

This data type identifies the synchronous modes for the Timer driver.

Members

Members	Description
DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL	Synchronous Internal Clock Counter
DRV_TMR_SYNC_MODE_SYNCHRONOUS_INTERNAL_GATED	Synchronous Internal Gated Counter
DRV_TMR_SYNC_MODE_SYNCHRONOUS_EXTERNAL_WITHOUT_CLKSYNC	Synchronous External Clock Counter
DRV_TMR_SYNC_MODE_SYNCHRONOUS_EXTERNAL_WITH_CLKSYNC	Synchronous External Clock Counter
DRV_TMR_SYNC_MODE_ASYNCHRONOUS_EXTERNAL_WITHOUT_CLKSYNC	Asynchronous External Clock Counter
DRV_TMR_SYNC_MODE_ASYNCHRONOUS_EXTERNAL_WITH_CLKSYNC	Asynchronous External Clock Counter
DRV_TMR_SYNC_MODE_IDLE	Timer Idle Mode

Remarks

Not all modes are available on all devices.

5.1.10.7.8 Macros**5.1.10.7.8.1 DRV_TMR_INTERRUPT_SOURCE Macro**

C

```
#define DRV_TMR_INTERRUPT_SOURCE INT_SOURCE_TIMER_1
```

Description

TMR Interrupt Source

This macro defines the interrupt source of the static driver.

Remarks

Refer to the Interrupt Peripheral Library documentation for more information on the [INT_SOURCE](#) enumeration.

5.1.10.7.8.2 DRV_TMR_POWER_STATE Macro

C

```
#define DRV_TMR_POWER_STATE SYS_MODULE_POWER_IDLE_STOP
```

Description

TMR power state configuration

This macro controls the power state of the Timer.

Remarks

This feature is not available in all modules/devices. Refer to the specific device data sheet for more information.

5.1.10.7.8.3 DRV_TMR_UNIFIED Macro

C

```
#define DRV_TMR_UNIFIED false
```

Description

TMR driver configuration

This definition sts up the driver functionality in either split or the unified mode.

Remarks

None.

5.1.10.8 Files

Files

Name	Description
drv_tmr.h	Timer device driver interface header.
drv_tmr_config_template.h	Timer driver configuration definitions for the template version.

Description

5.1.10.8.1 drv_tmr.h

Timer Device Driver Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Timer device driver.

Enumerations

	Name	Description
	DRV_TMR_ALARM_TYPE	Timer driver type enumeration.
	DRV_TMR_CLIENT_STATUS	Identifies the client-specific status of the Timer driver
	DRV_TMR_SYNC_MODE	Identifies the synchronous modes for the Timer driver.

Functions

	Name	Description
	DRV_TMR_AlarmCountClear	Clears the alarm elapse counter.
	DRV_TMR_AlarmCountGet	Provides the number of times the alarm has elapsed since the last time it was cleared.
	DRV_TMR_AlarmSet	Initializes an alarm.
	DRV_TMR_ClientStatus	Gets current client-specific status of the Timer driver.
	DRV_TMR_Close	Closes an opened instance of the Timer driver
	DRV_TMR_Counter16BitGet	Provides the 16-bit Timer's current counter value.
	DRV_TMR_Counter16BitSet	Updates the 16-bit Timer's initial count value.
	DRV_TMR_Counter32BitGet	Provides the 32-bit Timer's current counter value.
	DRV_TMR_Counter32BitSet	Updates the 32-bit Timer's initial count value.
	DRV_TMR_Counter8BitGet	Provides the 8-bit Timer's current counter value.
	DRV_TMR_Counter8BitSet	Updates the 8-bit Timer's initial count value
	DRV_TMR_Deinitialize	Deinitializes the specified instance of the Timer driver.
	DRV_TMR_ElapsedStatusGetAndClear	Provides and clears the Timer's elapse count.
	DRV_TMR_Initialize	Initializes the Timer driver .
	DRV_TMR_Open	Opens the specified Timer driver instance and returns a handle to it.
	DRV_TMR_OperatingFrequencyGet	Provides the Timer operating frequency.
	DRV_TMR_Period16BitGet	Provides the 16-bit Timer's period

	DRV_TMR_Period16BitSet	Updates the 16-bit Timer's period.
	DRV_TMR_Period32BitGet	Provides the 32-bit Timer's period
	DRV_TMR_Period32BitSet	Updates the 32-bit Timer's period.
	DRV_TMR_Period8BitGet	Provides the 8-bit Timer's period.
	DRV_TMR_Period8BitSet	Updates the 8-bit timer's period
	DRV_TMR_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings.
	DRV_TMR_Start	Starts the Timer counting.
	DRV_TMR_Status	Provides the current status of the Timer driver.
	DRV_TMR_Stop	Stops the Timer from counting.
	DRV_TMR_SyncModeGet	Provides the Timer's current synchronization mode.
	DRV_TMR_SyncModeSet	Changes the Timer's synchronization mode.
	DRV_TMR_Tasks	Maintains the driver's state machine and implements its ISR.
	DRV_TMR_TickFrequencyGet	Provides the Timer's counter "tick" frequency.
	DRV_TMR_VersionGet	Gets the Timer driver version in numerical format.
	DRV_TMR_VersionStrGet	Gets the Timer driver version in string format.

Macros

	Name	Description
	DRV_TMR_INDEX_0	Timer driver index definitions
	DRV_TMR_INDEX_1	This is macro DRV_TMR_INDEX_1.
	DRV_TMR_INDEX_2	This is macro DRV_TMR_INDEX_2.
	DRV_TMR_INDEX_3	This is macro DRV_TMR_INDEX_3.
	DRV_TMR_INDEX_4	This is macro DRV_TMR_INDEX_4.
	DRV_TMR_INDEX_5	This is macro DRV_TMR_INDEX_5.
	DRV_TMR_INDEX_COUNT	Number of valid Timer driver indices.

Structures

	Name	Description
	DRV_TMR_ALARM_CONFIG	Timer module alarm configuration.
	DRV_TMR_INIT	Defines the Timer driver initialization data.

Types

	Name	Description
	DRV_TMR_CALLBACK	Pointer to a Timer driver callback function data type.

File Name

drv_tmr.h

Company

Microchip Technology Inc.

5.1.10.8.2 drv_tmr_config_template.h

Timer Driver Configuration Definitions for the Template Version

These definitions set up the driver for the default mode of operation of the driver.

Macros

	Name	Description
	<u>DRV_TMR_ALARM_ENABLE</u>	Enables alarm functionality.
	<u>DRV_TMR_ALARM_PERIODIC</u>	Alarm type configuration.
	<u>DRV_TMR_ASYNC_WRITE_ENABLE</u>	Controls Asynchronous Write mode of the Timer.
	<u>DRV_TMR_CLIENTS_NUMBER</u>	Selects the maximum number of clients
	<u>DRV_TMR_CLOCK_SOURCE</u>	Timer module clock source selection.
	<u>DRV_TMR_COUNT_WIDTH</u>	Controls the mode of the Timer.
	<u>DRV_TMR_INDEX</u>	Timer static index selection.
	<u>DRV_TMR_INSTANCES_NUMBER</u>	Sets up the maximum number of hardware instances that can be supported by the dynamic driver.
	<u>DRV_TMR_INTERRUPT_MODE</u>	Controls operation of the driver in the interrupt or polled mode.
	<u>DRV_TMR_INTERRUPT_SOURCE</u>	Defines the interrupt source of the static driver.
	<u>DRV_TMR_PERIPHERAL_ID</u>	Timer peripheral ID selection.
	<u>DRV_TMR_POSTSCALE</u>	Macro controls post scale of the TMR
	<u>DRV_TMR_POWER_STATE</u>	Controls the power state of the Timer.
	<u>DRV_TMR_PRESCALE</u>	Timer module prescale value.
	<u>DRV_TMR_PRESCALER_ENABLE</u>	Controls prescaler assignment of the Timer.
	<u>DRV_TMR_SOURCE_EDGE</u>	Timer module source edge.
	<u>DRV_TMR_SYNCHRONIZATION_MODE</u>	Controls Synchronization mode of the Timer.
	<u>DRV_TMR_TIMER_PERIOD</u>	Timer period value for an overflow or a period match-based Timer.
	<u>DRV_TMR_UNIFIED</u>	Sets up the driver functionality in either split or the unified mode.

File Name

drv_tmr_config_template.h

Company

Microchip Technology Inc.

5.1.11 USART Driver Library

5.1.11.1 Introduction

USART Driver Library

for

Microchip Microcontrollers

This library provides an interface to manage the USART module on Microchip family of micro controllers in different modes of operation.

Overview of USART

The Universal Synchronous Asynchronous Receiver/Transmitter (USART) controller is the key component of the serial

communications subsystem of many embedded systems.

The USART driver can support following modes of operation. (Refer to the Device Specific data sheet to determine which modes are supported on the part being used.)

1. RS232

RS232 is an asynchronous full duplex serial communication protocol. It uses separate lines for transmitting and receiving data, point-to-point, between a Data Terminal Equipment (DTE) item and a Data Communication Equipment (DCE) item at a maximum speed of 20 kbps with a maximum cable length of 50 feet.

2. RS485

RS485 is a multi-point communications network. It provides bi-directional, half-duplex, data transmission that allows multiple receivers and drivers in "bus" configurations. The standard specifies up to 32 drivers and 32 receivers on a single (2-wire) bus. With the introduction of "automatic" repeaters and high-impedance drivers and receivers this "limitation" can be extended to hundreds (or even thousands) of nodes on a network.

3. IrDA

Some Microchip micro controllers provide an Infrared (IR) encoder and decoder and can also interface with external IR encoders and decoders.

4. Interdevice Communication

USART module can be used for communication(without any external transceiver) between two devices. The maximum communication speed in this case is

determined by the maximum Baud supported by the device.

5. Synchronous Master/Slave

Synchronous serial transmission requires that the sender and receiver share a clock with one another or that the sender provide a strobe or other timing signal so that the receiver knows when to latch the next bit of the data. It is typically used in systems with a single master and one or more slaves. The master device contains the necessary circuitry for baud rate generation and supplies the clock for all devices in the system.

Note: if a USART does not support a synchronous mode, it is called UART. However, UARTs are still controlled by the USART driver.

5.1.11.2 Release Notes

MPLAB Harmony Version: v0.70b **USART Driver Library Version :** 0.050b **Alpha Release Date:** 18Nov2013

New This Release:

The USART Driver supports the following modes of operation.

Modes of operation

- Polling Mode
- Interrupt Mode

Configuration modes

- Asynchronous Transmit & Receive(i.e. RS232, RS485)
- IrDA Mode
- Loop Back Mode

Handshake modes

- Flow Control Mode
- No Handshake Mode

Line Control modes

- 8 bit, No Parity, 1 stop bit Mode
- 9 bit, No Parity, 1 stop bit Mode
- 8 bit, Even Parity, 1 stop bit Mode
- 8 bit, Even Parity, 2 stop bits Mode
- 8 bit, Odd Parity, 1 stop bit Mode
- 8 bit, Odd Parity, 2 stop bits Mode

Interrupt Modes

- Wakeup On Start Enable
- Auto Baud enable

Following demo shows how to use USART Driver

- `usart_echo` (<install drive:>\apps\driver\usart\drv_usart_demo)

Known Issues:

The following are the known issues in this release.

- Although implementation exists, the driver doesn't work in the following line control modes:
 - 8 bit, Even Parity, 1 stop bit Mode
 - 8 bit, Even Parity, 2 stop bits Mode
 - 8 bit, Odd Parity, 1 stop bit Mode
 - 8 bit, Odd Parity, 2 stop bits Mode
- Deinitialize API of the driver doesn't work. This means that the application cannot initialize the driver after deinitializing it.
- When UART driver is configured for 115200 baud rate, there could be no data received or received data is corrupted
- More than one client is not supported
- The USART Driver does not support "Odd" and "Even" parity settings

5.1.11.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have

paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.11.4 Using the Library

This topic describes the basic architecture of the USART Driver Library and provides information and examples on how to use it.

Interface Header File: `drv_usart.h`

The interface to the USART library is defined in the "drv_usart.h" header file

Please refer to the [MPLAB Harmony Overview](#) for how the driver interacts with the framework.

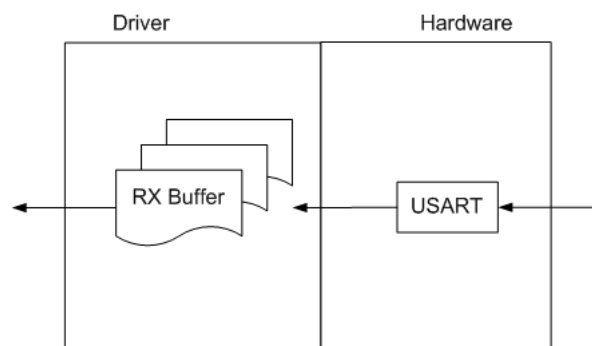
5.1.11.4.1 Abstraction Model

Different types of USARTs are available on Microchip microcontrollers. Some have a FIFO and some do not. The FIFO depth varies across part families. The USART driver abstracts out these differences and provides a unified model for data transfer across different types of USARTS available.

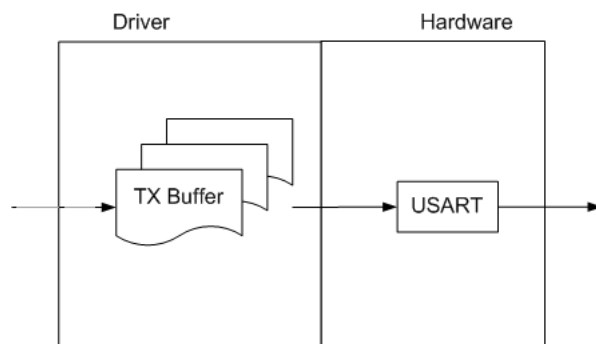
Both transmitter and receiver provide a buffer in the driver which transmits and receives data to/from the hardware. The USART driver provides a set of interfaces to perform the read and the write.

The diagrams below illustrates the model used by the USART driver for transmitter and receiver.

Receiver Abstraction Model



Transmitter Abstraction Model



5.1.11.4.2 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

The table below lists the interface section and its brief description.

Library Interface Section	Description
Data Types and Constants	Provides macros for configuring the system. It is required that the system configures the driver to build correctly by choosing appropriate configuration options as listed in this section.
System Initialization Functions	Provides system module interfaces, Device initialization, de-initialization, re-initialization and status functions.
Setup and Status Functions	Provides open, close and other setup functions
Data Transfer Functions	Provides data transfer functions available in the core configuration

5.1.11.4.3 How the Library Works

The library provides interfaces to support:

- System Interaction
- Client Core Functionality

5.1.11.4.3.1 System Initialization

System Initialization and Reinitialization

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or re-initialized. During system initialization each instance of the USART device will be initialized with the following configuration settings (either passed dynamically at run time using [DRV_USART_INIT](#) or by using Initialization Overrides) that are supported by the specific USART device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to the Data Types section in this guide
- The actual peripheral ID enumerated as the PLIB level module ID. e.g USART_ID_2.
- Operation Mode: Select the operation mode of the instance using one of the possible values from [DRV_USART_OPERATION_MODES](#). Make sure that the operation mode support is enabled in the driver configuration setup. Also for RS485 operation mode, select the address to which the module instance should respond to.
- Line Control Mode: Setup the line control mode using one of the possible values from

DRV_USART_LINE_CONTROL_MODES

- Baud Rate Setup: Setup the baud rate by either providing the baud rate value or enabling the auto baud setup.
- Configuration of the RS485 mode if enabled.
- Enabling the handshake signals.e.g. Simplex, FlowControl. For specifics please refer the Data Types section in this guide.
- Defining the respective interrupt sources for Tx, Rx, and Error Interrupt.

The [DRV_USART_Initialize](#) API returns an object handle of the type [SYS_MODULE_OBJ](#). There on, the object handle returned by the Initialize interface would be used by the other system interfaces like [DRV_USART_Reinitialize](#), [DRV_USART_Deinitialize](#), [DRV_USART_Status](#) and [DRV_USART_TasksTX](#), [DRV_USART_TasksRX](#).

Note:The system initialization and the re-initialization settings, only effect the instance of the peripheral that is being initialized or re-initialized.

Example:

```
DRV_USART_INIT          usartInitData;
//Alternatively NULL can be passed to Initialize which then
//makes sure that static overrides as selected by the user
//will be used as default values
SYS_MODULE_OBJ          objectHandle;

usartInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
usartInitData.moduleId              = USART_ID_2;
usartInitData.operationMode        = DRV_USART_OPERATION_MODE_RS232;
usartInitData.initFlags             = 0;
usartInitData.lineControlMode       = DRV_USART_LINE_CONTROL_MODE_8NONE1;
usartInitData.brgClock              = 4000000;
usartInitData.brgValue              = 9600;
usartInitData.operationModeInit     = 0;
usartInitData.handShakeMode         = DRV_USART_HANDSHAKE_MODE_FLOWCONTROL;
usartInitData.txInterruptSource     = INT_SOURCE_USART_2_TRANSMIT;
usartInitData.rxInterruptSource     = INT_SOURCE_USART_2_RECEIVE;
usartInitData.errorInterruptSource  = INT_SOURCE_USART_2_ERROR;
usartInitData.txQueueSize           = 3;
usartInitData.rxQueueSize           = 3;

objectHandle = DRV_USART_Initialize(DRV_USART_INDEX_1, (SYS_MODULE_INIT*)usartInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Tasks Routine

The system will either call [DRV_USART_TasksTX](#), [DRV_USART_TasksRX](#), from System Task Service (in a polled environment) or [DRV_USART_TasksTX](#), [DRV_USART_TasksRX](#) will be called from interrupt service routine of the USART.

5.1.11.4.3.2 Core Functionality

Core functionality provides a extremely basic interface for the driver operation.

Core Transmitter Functionality

Application using the USART core functionality, needs to perform the following:

1. The system should have completed necessary initialization and the [DRV_USART_TasksTX](#) should either be running in polled environment, or in an interrupt environment.
2. Open the driver using [DRV_USART_Open](#), timer driver supports exclusive access only.
3. Write a byte using [DRV_USART_WriteByte](#).
4. The client will be able to close itself using the [DRV_USART_Close](#) when required.

Example :

```

DRV_HANDLE      handle;    // Returned from DRV_USART_Open

bool            status;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.
    status = DRV_USART_WriteByte(handle, myBuffer[numBytes++]);

    // Do something else...
}

```

Core Receiver Functionality

Application using the USART core functionality, needs to perform the following:

1. The system should have completed necessary initialization and the [DRV_USART_TasksRX](#) should either be running in polled environment, or in an interrupt environment.
2. Open the driver using [DRV_USART_Open](#), timer driver supports exclusive access only.
3. Write a byte using [DRV_USART_ReadByte](#).
4. The client will be able to close itself using the [DRV_USART_Close](#) when required.

Example :

```

DRV_HANDLE      handle;    // Returned from DRV_USART_Open
bool            status;
char            byte;

bool = DRV_USART_ReadByte(handle, &byte);

// Do something else...

```

5.1.11.4.3.3 Client Access

For the application to start using an instance of the module, it must call the [DRV_USART_Open](#) function. This provides the configuration required to open the USART instance for operation. If the driver is de-initialized using the function [DRV_USART_Deinitialize](#), the application must call the [DRV_USART_Open](#) function again to setup the instance of the USART. The function [DRV_USART_Open](#) need not be called again if the system is re-initialized using the [DRV_USART_Reinitialize](#) function.

//This i have to mod

For the various options available for I/O INTENT please refer to the Data Types and Constants section.

Example:

```

DRV_HANDLE handle;

handle = DRV_USART_Open(DRV_USART_INDEX_1, DRV_IOINTENT_READWRITE); // Configure the instance
USART_ID_1 with the above configuration

if(handle == DRV_HANDLE_INVALID)
{
    // Client cannot open the instance.
}

```

5.1.11.4.3.4 Data Transfer

In the basic data transfer mode, the USART device driver allows the client to perform simple read or write operation on the USART. The client (who is this) has to provide a buffer where the transfer data is/will be stored.

Example : Read in the RS-232 mode

```

// USART is operating in RS-232 Mode
// Assume that DRV_USART_Open operation was called with DRV_IO_INTENT_NONBLOCKING parameter

char myBuffer[MY_BUFFER_SIZE];
size_t total;
size_t count;
DRV_HANDLE drvHandle; // device handle obtained by a call to DRV_USART_Open

// Start the transfer
total = 0;
do
{
    count = DRV_USART_Read(drvHandle, myBuffer, MY_BUFFER_SIZE - total);
    total += count;

    // Do something else...
} while(total < MY_BUFFER_SIZE);

```

Example : Write in the RS-485 mode

```

// USART is operating in RS-485 Mode
// Assume that DRV_USART_Open operation was called with DRV_IO_INTENT_NONBLOCKING parameter

char myBuffer[MY_BUFFER_SIZE];
size_t total;
size_t count;
DRV_HANDLE drvHandle; // device handle obtained by a call to DRV_USART_Open

// Start the transfer
total = 0;
do
{
    count = DRV_USART_WriteToAddress(drvHandle, 0x07, myBuffer, MY_BUFFER_SIZE - total); //
    // Write the string "writestring" to the communications channel

    total += count;

    // Do something else...
} while(total < MY_BUFFER_SIZE);

```

5.1.11.4.3.5 Advanced Data Transfer

USART device client transfer is composed of various buffers that are queued internally by the driver.

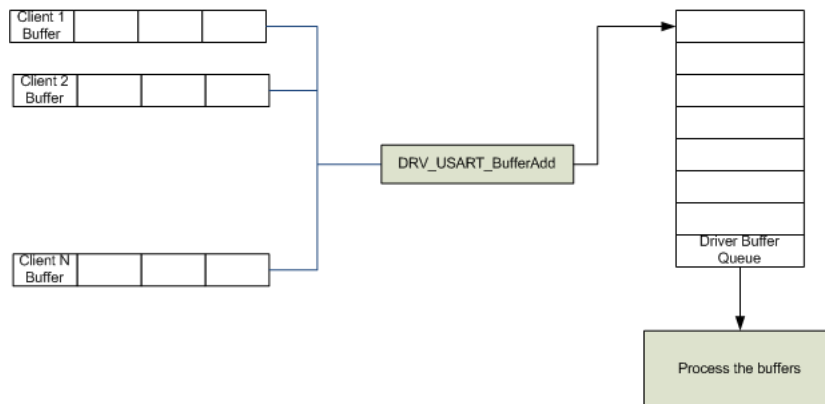
Each buffer is described by a buffer descriptor which contains the transfer buffer, size and an address

Each buffer has associated flags that specify the behavior that the device driver has to take once that particular segment reaches a particular stage of processing.

The client can add segments to the transfer by using [DRV_USART_BufferAdd](#) for adding the Read and write buffers.

The client can query the status of the buffer using the function [DRV_USART_BufferStatus](#) and also can query how many bytes of the buffer are read or written if the buffer is in progress using the function [DRV_USART_BufferTransferStatus](#).

The diagram below illustrates the buffer model used by the advanced data transfer.

**Example:**

```

void Callback(void);
uint8_t mybuffer[MY_BUFFER_SIZE];
uint8_t bytesWritten = 0;
DRV_BUFFER_HANDLE bufferHandle = NULL;
DRV_USART_IO_BUFFER buffer1;

void Callback(void)
{
    Nop();
}

buffer1.buffer      = writeBuffer2;
buffer1.bufferSize  = sizeof(writeBuffer2);
buffer1.xferCallback = Callback;
buffer1.flags       = DRV_USART_BUFFER_FLAG_WRITE;

// myUSARTHandle is the handle returned by the DRV_USART_Open function.

bufferHandle = DRV_USART_BufferAdd(myUSARTHandle, &buffer1);

bufferStatus = DRV_USART_BufferStatus ( uartHandle, uartBufferHandle );

if ( DRV_USART_BUFFER_COMPLETED == bufferStatus )
{
    // Handle Buffer Complete
}else
{
    // Handle other
}

```

Example:

```

void Callback(void);
uint8_t mybuffer[MY_BUFFER_SIZE];
uint8_t bytesWritten = 0;
DRV_BUFFER_HANDLE bufferHandle = NULL;
DRV_USART_IO_BUFFER buffer1;

void Callback(void)
{
    Nop();
}

buffer1.buffer      = readBuffer2;
buffer1.bufferSize  = sizeof(readBuffer2);
buffer1.xferCallback = Callback;
buffer1.flags       = DRV_USART_BUFFER_FLAG_READ;

```

```
// myUSARTHandle is the handle returned by the DRV_USART_Open function.

bufferHandle = DRV_USART_BufferAdd(myUSARTHandle, &buffer1);

bufferStatus = DRV_USART_BufferStatus ( uartHandle, uartBufferHandle );

if ( DRV_USART_BUFFER_COMPLETED == bufferStatus )
{
    // Handle Buffer Complete
}else
{
    // Handle other
}
```

5.1.11.5 Configuring the Library

The configuration of the USART device driver is based on the file sys_config.h

This header file contains the configuration selection for the USART device driver build. Based on the selections made here and the system setup the USART device driver will support or not selected features. These configuration settings will apply to all instances of the device driver.

This header can be placed anywhere in the application specific folders and the path of this header needs to be presented to the include search for a successful build. Refer to the [Applications Overview](#) section for more details.

Initialization overrides not supported in this version.

5.1.11.6 Building the Library

This section list the files that are available in the \src of the timer driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.1.11.7 Library Interface

Data Types and Constants

	Name	Description
	DRV_USART_AUTO_BAUD_ENABLE	Macro configures enabling the auto baud.
	DRV_USART_BAUD_RATE	Macro controls operation of the driver for Baud rate configuration
	DRV_USART_BYTE_Q_SIZE_RX	Macro controls operation of the driver for defining the size of the Rx buffer
	DRV_USART_BYTE_Q_SIZE_TX	Macro controls operation of the driver for defining the size of the Tx buffer
	DRV_USART_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_USART_HANDSHAKE_MODE	Configures the handshake mode
	DRV_USART_INDEX	USART Static Index selection
	DRV_USART_INDEX_0	USART driver index definitions
	DRV_USART_INDEX_1	This is macro DRV_USART_INDEX_1.
	DRV_USART_INDEX_2	This is macro DRV_USART_INDEX_2.
	DRV_USART_INDEX_3	This is macro DRV_USART_INDEX_3.
	DRV_USART_INDEX_4	This is macro DRV_USART_INDEX_4.

	<u>DRV_USART_INDEX_5</u>	This is macro DRV_USART_INDEX_5.
	<u>DRV_USART_INDEX_COUNT</u>	Number of valid USART driver indices
	<u>DRV_USART_INSTANCES_NUMBER</u>	Sets up the maximum number of hardware instances that can be supported
	<u>DRV_USART_INTERRUPT_MODE</u>	Macro controls operation of the driver in the interrupt mode
	<u>DRV_USART_INTERRUPT_SOURCE_ERROR</u>	Defines the interrupt source for the error interrupt
	<u>DRV_USART_INTERRUPT_SOURCE_RX</u>	Macro to define the Rx interrupt source in case of static driver
	<u>DRV_USART_INTERRUPT_SOURCE_TX</u>	Macro to define the Tx interrupt source in case of static driver
	<u>DRV_USART_LINE_CONTROL</u>	Macro defines the line control mode
	<u>DRV_USART_OPERATION_MODE</u>	Defines the mode to be used for the USART driver
	<u>DRV_USART_PERIPHERAL_ID</u>	Configures the PLIB id
	<u>DRV_USART_POLARITY_INVERTED_RX</u>	Defines the polarity of the RX pin.
	<u>DRV_USART_POLARITY_INVERTED_TX</u>	Defines the polarity of the TX pin.
	<u>DRV_USART_POWER_STATE</u>	Macro controls power state of the USART
	<u>DRV_USART_WAKE_ON_START_ENABLE</u>	Macro configures that the peripheral should wake up on start
	<u>DRV_USART_XFER_BUFFER_NUMBER</u>	Transfer buffer depth
	<u>DRV_USART_BUFFER_FLAGS</u>	Defines the IO types that can be serviced by the USART driver
	<u>DRV_USART_BUFFER_STATUS</u>	Defines the buffer status
	<u>DRV_USART_CALLBACK</u>	Pointer to a USART callback function
	<u>DRV_USART_CLIENT_STATUS</u>	Defines the client status
	<u>DRV_USART_HANDSHAKE_MODES</u>	Identifies the handshaking modes available on the USART
	<u>DRV_USART_INIT</u>	Defines the data required to initialize or reinitialize the USART driver
	<u>DRV_USART_INIT_FLAGS</u>	Defines specific initialization features
	<u>DRV_USART_IO_BUFFER</u>	IO buffer information structure
	<u>DRV_USART_LINE_CONTROL_MODES</u>	Identifies the line control modes available
	<u>DRV_USART_OPERATION_MODE_INIT</u>	Defines the initialization data required for different operation modes of USART
	<u>DRV_USART_OPERATION_MODES</u>	Identifies the modes of the operation of the USART module
	<u>DRV_USART_TRANSFER_STATUS</u>	Specifies the status of the receive or transmit
	<u>DRV_USART_XFER_HANDLE</u>	Handle for requested transfer..

Data Transfer Functions

	Name	Description
	<u>DRV_USART_BufferAdd</u>	Starts the writes to the communication channel.
	<u>DRV_USART_BufferStatus</u>	Provides the status of the given buffer
	<u>DRV_USART_Read</u>	Reads data from the USART
	<u>DRV_USART_ReadByte</u>	Reads a byte of data from the USART
	<u>DRV_USART_TransferStatus</u>	Returns the transmitter and receiver transfer status
	<u>DRV_USART_Write</u>	Writes data to the USART
	<u>DRV_USART_WriteByte</u>	Writes a byte of data to the USART

Setup and Status Functions

	Name	Description
	<u>DRV_USART_BufferTransferStatus</u>	Gets the number of bytes which are written or read
	<u>DRV_USART_ClientStatus</u>	Gets current client-specific status the USART driver
	<u>DRV_USART_Close</u>	Closes an opened-instance of the USART driver

	DRV_USART_Open	Opens the specified timer driver instance and returns a handle to it
	DRV_USART_Status	Gets the current status of the USART driver module.

System Initialization Functions

	Name	Description
	DRV_USART_Deinitialize	Deinitializes the specified instance of the USART driver module
	DRV_USART_Initialize	Initializes the USART instance for the specified driver index
	DRV_USART_Reinitialize	Reinitializes the USART driver and refreshes any associated hardware settings
	DRV_USART_TasksError	Maintains the driver's error-handling state machine and implements its ISR
	DRV_USART_TasksRX	Maintains the driver's receiver state machine and implements its ISR
	DRV_USART_TasksTX	Maintains the driver's transmitter state machine and implements its ISR

Description

This section describes the functions of the USART Driver Library.

Refer to each section for a detailed description.

5.1.11.7.1 System Initialization Functions

5.1.11.7.1.1 DRV_USART_Deinitialize Function

C

```
void DRV_USART_Deinitialize(
    SYS_MODULE_OBJ object
);
```

Description

Deinitializes the specified instance of the USART driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Preconditions

Function [DRV_USART_Initialize](#) should have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USART_Initialize routine

Returns

None.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This routine will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the [DRV_USART_Status](#) operation. The system has to use [DRV_USART_Status](#) to find out when the module is in the ready state.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_USART_Initialize
SYS_STATUS        status;
```

```

DRV_USART_Deinitialize(object);

status = DRV_USART_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}

```

5.1.11.7.1.2 DRV_USART_Initialize Function

C

```

SYS_MODULE_OBJ DRV_USART_Initialize(
    const SYS_MODULE_INDEX index,
    const SYS_MODULE_INIT * const init
);

```

Description

This routine initializes the USART driver instance for the specified driver index, making it ready for clients to open and use it.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver. This pointer may be null if no data is required because static overrides have been provided.

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#).

Remarks

This routine must be called before any other USART routine is called.

This routine should only be called once during system initialization unless [DRV_USART_Deinitialize](#) is called to de-initialize the driver instance.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to re-initialize, it will be reported by the [DRV_USART_Status](#) operation. The system must use [DRV_USART_Status](#) to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this routine.

Example

```

DRV_USART_INIT          usartInitData;
//Alternatively NULL can be passed to Initialize which then
//makes sure that static overrides as selected by the user
//will be used as default values
SYS_MODULE_OBJ          objectHandle;

usartInitData.moduleInit.value    = SYS_MODULE_POWER_RUN_FULL;
usartInitData.moduleId           = USART_ID_2;
usartInitData.operationMode      = DRV_USART_OPERATION_MODE_RS232;
usartInitData.initFlags          = 0;
usartInitData.lineControlMode    = DRV_USART_LINE_CONTROL_MODE_8NONE1;
usartInitData.brgClock           = 4000000;

```

```

usartInitData.brgValue           = 9600;
usartInitData.operationModeInit  = 0;
usartInitData.handShakeMode      = DRV_USART_HANDSHAKE_MODE_FLOWCONTROL;
usartInitData.txInterruptSource   = INT_SOURCE_USART_2_TRANSMIT;
usartInitData.rxInterruptSource   = INT_SOURCE_USART_2_RECEIVE;
usartInitData.errorInterruptSource = INT_SOURCE_USART_2_ERROR;
usartInitData.txQueueSize        = 3;
usartInitData.rxQueueSize        = 3;

objectHandle = DRV_USART_Initialize(DRV_USART_INDEX_1, (SYS_MODULE_INIT*)usartInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}

```

5.1.11.7.1.3 DRV_USART_Reinitialize Function

C

```

void DRV_USART_Reinitialize(
    SYS_MODULE_OBJ object,
    const SYS_MODULE_INIT * const init
);

```

Description

This routine reinitializes the driver and refreshes any associated hardware settings using the initialization data given, but it will not interrupt any ongoing operations.

Preconditions

Function [DRV_USART_Initialize](#) must have been called before calling this routine and a valid [SYS_MODULE_OBJ](#) must have been returned.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USART_Initialize routine
init	Pointer to the initialization data structure

Returns

None.

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to re-initialize, it will be reported by the [DRV_USART_Status](#) operation. The system must use [DRV_USART_Status](#) to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this routine.

Example

```

DRV_USART_INIT usartInitData;
//Alternatively NULL can be passed to Initialize which then
//makes sure that static overrides as selected by the user
//will be used as default values

SYS_STATUS usartStatus;

```

```

SYS_MODULE_OBJ  object;

usartInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
usartInitData.moduleId              = USART_ID_2;
usartInitData.operationMode         = DRV_USART_OPERATION_MODE_RS232;
usartInitData.initFlags              = 0;
usartInitData.lineControlMode        = DRV_USART_LINE_CONTROL_MODE_8NONE1;
usartInitData.brgClock               = 4000000;
usartInitData.brgValue               = 9600;
usartInitData.operationModeInit      = 0;
usartInitData.handShakeMode          = DRV_USART_HANDSHAKE_MODE_FLOWCONTROL;
usartInitData.txInterruptSource      = INT_SOURCE_USART_2_TRANSMIT;
usartInitData.rxInterruptSource      = INT_SOURCE_USART_2_RECEIVE;
usartInitData.errorInterruptSource   = INT_SOURCE_USART_2_ERROR;

DRV_USART_Reinitialize(object, (SYS_MODULE_INIT*)usartInitData);

usartStatus = DRV_USART_Status(object);
if (SYS_STATUS_BUSY == usartStatus)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= usartStatus)
{
    // Handle error
}

```

5.1.11.7.1.4 DRV_USART_TasksError Function

C

```

void DRV_USART_TasksError(
    SYS_MODULE_OBJ object
);

```

Description

This routine is used to maintain the driver's internal error-handling state machine and implement its ISR for interrupt-driven implementations.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USART_Initialize)

Returns

None.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Example

```

SYS_MODULE_OBJ  object;    // Returned from DRV_USART_Initialize

while (true)
{
    DRV_USART_TasksError (object);
}

```

```
    // Do other tasks  
}
```

5.1.11.7.1.5 DRV_USART_TasksRX Function

C

```
void DRV_USART_TasksRX(  
    SYS_MODULE_OBJ object  
);
```

Description

This routine is used to maintain the driver's internal receiver state machine and implement its ISR for interrupt-driven implementations.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USART_Initialize)

Returns

None.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_USART_Initialize  
  
while (true)  
{  
    DRV_USART_TasksRX (object);  
  
    // Do other tasks  
}
```

5.1.11.7.1.6 DRV_USART_TasksTX Function

C

```
void DRV_USART_TasksTX(  
    SYS_MODULE_OBJ object  
);
```

Description

This routine is used to maintain the driver's internal transmitter state machine and implement its ISR for interrupt-driven implementations.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USART_Initialize)

Returns

None.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)) or by the appropriate raw ISR.

This routine may excute in an ISR context and will never block or access any resources that may cause it to block.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_USART_Initialize

while (true)
{
    DRV_USART_TasksTX (object);

    // Do other tasks
}
```

5.1.11.7.2 Data Transfer Functions**5.1.11.7.2.1 DRV_USART_BufferAdd Function****C**

```
DRV_HANDLE DRV_USART_BufferAdd(
    const DRV_HANDLE handle,
    DRV_USART_IO_BUFFER * bufferObject
);
```

Description

Starts the writes to the communication channel.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART device instance and the [DRV_USART_Status](#) must have returned SYS_STATUS_READY.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_USART_Open](#) call.

Parameters

Parameters	Description
handle	Handle of the communication channel as return by the DRV_USART_Open function.
bufferObject	Buffer Object that needs to be specified for the buffer that needs to be read or written.

Returns

The buffer handle, which should be used to query how many bytes have been written or read using the function

DRV_USART_OperationStatus

Remarks

If the DRV_IO_INTENT_BLOCKING is requested and the driver was built appropriately to support blocking behavior in an OS environment the call will block until the operation is complete.

For DRV_IO_INTENT_NONBLOCKING request the driver client can call the [DRV_USART_ClientStatus](#) operation to find out when the module is in the ready state.

In any circumstance, the client supplied buffer has to be non-volatile and it is not allowed to go out of scope until the operation completes.

Affected by the following configuration options:

- DRV_USART_CONFIG_SUPPORT_CLIENT_ACCESS_BLOCKING
- DRV_USART_CONFIG_BUFFER_QUEUE_SLOTS_MAX
- DRV_USART_CONFIG_SUPPORT_BUFFERQ

Example

```
void Callback(void);
uint8_t mybuffer[MY_BUFFER_SIZE];
int8_t bytesWritten = 0;
DRV_BUFFER_HANDLE bufferHandle = NULL;
DRV_USART_IO_BUFFER buffer1;
void Callback(void)
{
    Nop();
}

buffer1.buffer      = writeBuffer2;
buffer1.bufferSize  = sizeof(writeBuffer2);
buffer1.xferCallback = Callback;
buffer1.flags       = DRV_USART_BUFFER_FLAG_WRITE;

// myUSARTHandle is the handle returned by the DRV_USART_Open function.

bufferHandle = DRV_USART_BufferAdd(myUSARTHandle, &buffer1);

while(bytesWritten < MY_BUFFER_SIZE)
{
    bytesWritten = DRV_USART_BufferTransferStatus(handle, bufferHandle);
}
```

5.1.11.7.2.2 DRV_USART_BufferStatus Function

C

```
DRV_USART_BUFFER_STATUS DRV_USART_BufferStatus(
    DRV_HANDLE handle,
    const DRV_HANDLE bufferHandle
);
```

Description

This function provides the current status of the buffer identified by the given buffer handle.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART device instance and the [DRV_USART_Status](#) must have returned SYS_STATUS_READY.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the

[DRV_USART_Open](#) call.

The "bufferHandle" parameter must be the return value from a call to the "[DRV_USART_BufferAdd](#)" function.

Parameters

Parameters	Description
handle	Handle of the communication channel as return by the DRV_USART_Open function.
bufferHandle	Handle to the buffer in question. Must be the return value from a call to the " DRV_USART_BufferAdd " function.

Returns

The current status of the given buffer.

Remarks

See the definition of "[DRV_USART_BUFFER_STATUS](#)" for a complete description of all possible status values.

Example

```
DRV_USART_BUFFER_STATUS bufferStatus;

bufferStatus = DRV_USART_BufferStatus ( uartHandle, uartBufferHandle );
if ( DRV_USART_BUFFER_COMPLETED == bufferStatus )
{
    // Handle Buffer Complete
} else
{
    // Handle other
}
```

5.1.11.7.2.3 DRV_USART_Read Function

C

```
int DRV_USART_Read(
    const DRV_HANDLE handle,
    uint8_t * buffer,
    const unsigned int numbytes
);
```

Description

This routine reads data from the USART.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART driver instance.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_USART_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer into which the data read from the USART instance will be placed.
numbytes	Total number of bytes that need to be read from the module instance (must be equal to or less than the size of the buffer)

Returns

Number of bytes actually copied into the caller's buffer or -1 if there is an error (call [DRV_USART_ClientStatus](#) to identify the error).

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_USART_Open](#) routine and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_USART_Open](#) routine or the driver was built to only support non-blocking behavior the call will return immediately, identifying how many bytes of data were actually copied into the client's buffer and the client must call DRV_USART_Read again with adjusted parameters as shown in the example.

Example

```
DRV_HANDLE    handle;    // Returned from DRV_USART_Open
char          myBuffer[MY_BUFFER_SIZE];
unsigned int  count;
unsigned int  total;

total = 0;
do
{
    count = DRV_USART_Read(myUSARTHandle, &myBuffer[total], MY_BUFFER_SIZE - total);
    total += count;

    // Do something else...
} while( total < MY_BUFFER_SIZE );
```

5.1.11.7.2.4 DRV_USART_ReadByte Function**C**

```
bool DRV_USART_ReadByte(
    const DRV_HANDLE handle,
    char * byte
);
```

Description

This routine reads a byte of data from the USART.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART driver instance.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_USART_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
byte	A variable in which data is read into

Returns

true - Read successful false - Read failed

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_USART_Open](#) routine and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_USART_Open](#) routine or the driver was built to only support non-blocking behavior the call will return immediately. If data is not available, a zero (0) value will be returned and an underrun error status will be captured. To ensure that valid data is returned, the caller must first check the return value to [DRV_USART_TransferStatus](#) as shown in the example.

Example

```

DRV_HANDLE    handle;    // Returned from DRV_USART_Open
bool          status;
char          byte;

bool = DRV_USART_ReadByte(handle, &byte);

// Do something else...

```

5.1.11.7.2.5 DRV_USART_TransferStatus Function**C**

```

DRV_USART_TRANSFER_STATUS DRV_USART_TransferStatus(
    const DRV_HANDLE handle
);

```

Description

This returns the transmitter and receiver transfer status.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART driver instance.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

A [DRV_USART_TRANSFER_STATUS](#) value describing the current status of the transfer.

Remarks

The returned status may contain a value with more than one of the bits specified in the [DRV_USART_TRANSFER_STATUS](#) enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Example

```

DRV_HANDLE handle;    // Returned from DRV_USART_Open

if (DRV_USART_TRANSFER_STATUS_TX_EMPTY & DRV_USART_TransferStatus(handle) )
{
    // All transmitter data has been sent.
}

```

5.1.11.7.2.6 DRV_USART_Write Function**C**

```

bool DRV_USART_Write(
    const DRV_HANDLE handle,
    const uint8_t * buffer,
    const uint32_t numbytes
);

```

Description

This routine writes data to the USART.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART driver instance.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_USART_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer containing the data write to the USART instance
numbytes	Total number of bytes that to write to the USART instance (must be equal to or less than the size of the buffer)

Returns

Number of bytes actually written to the USART or -1 if there is an error (call [DRV_USART_ClientStatus](#) to identify the error)

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_USART_Open](#) routine and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_USART_Open](#) routine or the driver was built to only support non-blocking behavior the call will return immediately, identifying how many bytes of data were actually written to the USART and the client must call DRV_USART_Write again with adjusted parameters as shown in the example.

Example

```

DRV_HANDLE    handle;    // Returned from DRV_USART_Open
char          myBuffer[MY_BUFFER_SIZE];
unsigned int   count;
unsigned int   total;

total = 0;
do
{
    count = DRV_USART_Write(myUSARTHandle, &myBuffer[total], MY_BUFFER_SIZE - total);
    total += count;

    // Do something else...
} while( total < MY_BUFFER_SIZE );

```

5.1.11.7.2.7 DRV_USART_WriteByte Function**C**

```

bool DRV_USART_WriteByte(
    const DRV_HANDLE handle,
    const uint8_t byte
);

```

Description

This routine writes a byte of data to the USART.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART driver instance.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the [DRV_USART_Open](#) call.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
byte	Data byte to write to the USART

Returns

true - Write successful. false - Write failed.

Remarks

If the DRV_IO_INTENT_BLOCKING flag was given in the call to the [DRV_USART_Open](#) routine and the driver was built to support blocking behavior the call will block until the operation is complete.

If the DRV_IO_INTENT_NONBLOCKING flag was given in the call to the [DRV_USART_Open](#) routine or the driver was built to only support non-blocking behavior the call will return immediately. If the driver was not able to [accept](#) the data, an overrun error status will be captured. To ensure that the driver is ready to [accept](#) data, the caller must first check the return value to [DRV_USART_TransferStatus](#) as shown in the example.

Example

```
DRV_HANDLE    handle;    // Returned from DRV_USART_Open

bool          status;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.
    status =  DRV_USART_WriteByte(handle, myBuffer[numBytes++]);
    // Do something else...
}
```

5.1.11.7.3 Setup and Status Functions

5.1.11.7.3.1 DRV_USART_BufferTransferStatus Function

C

```
int8_t DRV_USART_BufferTransferStatus(
    const DRV_HANDLE handle,
    const DRV_HANDLE bufferHandle
);
```

Description

Gets the number of bytes which are written or read for the transaction handle

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART device instance and the [DRV_USART_Status](#) must have returned SYS_STATUS_READY.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

The function [DRV_USART_BufferAdd](#) should be called before calling this function and should have returned a valid buffer handle.

Parameters

Parameters	Description
handle	Handle of the communication channel as return by the DRV_USART_Open function.

bufferHandle	Handle of the buffer returned by the DRV_USART_BufferAdd

Returns

The number of bytes read if the buffer id is DRV_IO_BUFFER_READ or number of bytes written when the buffer id is DRV_IO_BUFFER_WRITE. If -1 is returned

Remarks

None.

Example

```
void Callback(void);
uint8_t mybuffer[MY_BUFFER_SIZE];
int8_t bytesWritten = 0;
DRV_BUFFER_HANDLE bufferHandle = NULL;
DRV_USART_IO_BUFFER buffer1;
void Callback(void)
{
    Nop();
}

buffer1.buffer      = writeBuffer2;
buffer1.bufferSize  = sizeof(writeBuffer2);
buffer1.xferCallback = Callback;
buffer1.flags       = DRV_USART_BUFFER_FLAG_WRITE;

// myUSARTHandle is the handle returned by the DRV_USART_Open function.

bufferHandle = DRV_USART_BufferAdd(myUSARTHandle, &buffer1);

while(bytesWritten < MY_BUFFER_SIZE)
{
    bytesWritten = DRV_USART_BufferTransferStatus(handle, bufferHandle);
}
```

5.1.11.7.3.2 DRV_USART_ClientStatus Function**C**

```
DRV_USART_CLIENT_STATUS DRV_USART_ClientStatus(
    const DRV_HANDLE handle
);
```

Description

This routine gets the client-specific status of the USART driver associated with the given handle.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

A [DRV_USART_CLIENT_STATUS](#) value describing the current status of the driver

Remarks

This routine will not block for hardware access and will immediately return the current status.

Example

```

DRV_HANDLE      handle;           // Returned from DRV_USART_Open
DRV_USART_CLIENT_STATUS clientStatus;

clientStatus = DRV_USART_ClientStatus(handle);
if(DRV_USART_CLIENT_STATUS_ERROR >= clientStatus)
{
    // Handle the error
}

```

5.1.11.7.3.3 DRV_USART_Close Function**C**

```

void DRV_USART_Close(
    const DRV_HANDLE handle
);

```

Description

This routine closes an opened-instance of the USART driver, invalidating the handle.

Preconditions

The [DRV_USART_Initialize](#) routine must have been called for the specified USART driver instance.

[DRV_USART_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Returns

None

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling [DRV_USART_Open](#) before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the [DRV_USART_Status](#) operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Example

```

DRV_HANDLE handle; // Returned from DRV_USART_Open

DRV_USART_Close(handle);

```

5.1.11.7.3.4 DRV_USART_Open Function**C**

```

DRV_HANDLE DRV_USART_Open(
    const SYS_MODULE_INDEX index,
    const DRV_IO_INTENT ioIntent
);

```

Description

This routine opens the specified USART driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Preconditions

Function [DRV_USART_Initialize](#) must have been called before calling this function.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "or'd" together to indicate the intended use of the driver

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is [DRV_HANDLE_INVALID](#).

Remarks

The handle returned is valid until the [DRV_USART_Close](#) routine is called.

This routine will NEVER block waiting for hardware.

If the DRV_IO_INTENT_BLOCKING is requested and the driver was built appropriately to support blocking behavior, then other client-level operations may block waiting on hardware until they are complete.

If DRV_IO_INTENT_NON_BLOCKING is requested the driver client can call the [DRV_USART_ClientStatus](#) operation to find out when the module is in the ready state.

If the requested intent flags are not supported, the routine will return [DRV_HANDLE_INVALID](#).

Example

```
DRV_HANDLE handle;

handle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

5.1.11.7.3.5 DRV_USART_Status Function

C

```
SYS_STATUS DRV_USART_Status(
    SYS_MODULE_OBJ object
);
```

Description

This routine provides the current status of the USART driver module.

Preconditions

Function [DRV_USART_Initialize](#) should have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USART_Initialize routine

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Note Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to **accept** new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Remarks

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been de-initialized

This value is less than SYS_STATUS_ERROR

The this operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This routine will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_USART_Initialize
SYS_STATUS        usartStatus;

usartStatus = DRV_USART_Status(object);
else if (SYS_STATUS_ERROR >= usartStatus)
{
    // Handle error
}
```

5.1.11.7.4 Data Types and Constants**5.1.11.7.4.1 DRV_USART_AUTO_BAUD_ENABLE Macro****C**

```
#define DRV_USART_AUTO_BAUD_ENABLE true
```

Description

Auto Baud enable

This macro configures enabling the auto baud. This is a static override of initialization parameter. This is the static override of the dynamic initialization passed using **DRV_USART_INIT**. The possible values it can take:

- true - Enables Auto Baud
- false - Disable Auto Baud

5.1.11.7.4.2 DRV_USART_BAUD_RATE Macro

C

```
#define DRV_USART_BAUD_RATE
```

Description

USART Baud Rate configuration

This macro controls the operation of the driver for Baud rate configuration. This is the static override of the dynamic initialization passed using [DRV_USART_INIT](#).

5.1.11.7.4.3 DRV_USART_BYTE_Q_SIZE_RX Macro

C

```
#define DRV_USART_BYTE_Q_SIZE_RX 16
```

Description

USART Byte mode internal buffer Rx size configuration

This macro controls the operation of the driver for defining the size of the Rx buffer

5.1.11.7.4.4 DRV_USART_BYTE_Q_SIZE_TX Macro

C

```
#define DRV_USART_BYTE_Q_SIZE_TX 16
```

Description

USART Byte mode internal buffer Tx size configuration

This macro controls the operation of the driver for defining the size of the Tx buffer

5.1.11.7.4.5 DRV_USART_CLIENTS_NUMBER Macro

C

```
#define DRV_USART_CLIENTS_NUMBER 2
```

Description

USART Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance.

Remarks

None

5.1.11.7.4.6 DRV_USART_HANDSHAKE_MODE Macro

C

```
#define DRV_USART_HANDSHAKE_MODE
```

Description

Hand Shake Mode

This macro configures the handshake mode. This is the static override of the dynamic initialization passed using [DRV_USART_INIT](#). The possible values that it can take are

- DRV_USART_HANDSHAKE_MODE_FLOWCONTROL
- DRV_USART_HANDSHAKE_MODE_SIMPLEX
- DRV_USART_HANDSHAKE_MODE_NONE

5.1.11.7.4.7 DRV_USART_INDEX Macro

C

```
#define DRV_USART_INDEX DRV_USART_INDEX_2
```

Description

Index - Used for static drivers

USART Static Index selection for the driver object reference

Remarks

This index is required to make a reference to the driver object

5.1.11.7.4.8 DRV_USART_INDEX_0 Macro

C

```
#define DRV_USART_INDEX_0 0
```

Description

Driver USART Module Index reference

These constants provide timer driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the [DRV_USART_Initialize](#) and [DRV_USART_Open](#) routines to identify the driver instance in use.

5.1.11.7.4.9 DRV_USART_INDEX_1 Macro

C

```
#define DRV_USART_INDEX_1 1
```

Description

This is macro DRV_USART_INDEX_1.

5.1.11.7.4.10 DRV_USART_INDEX_2 Macro

C

```
#define DRV_USART_INDEX_2 2
```

Description

This is macro DRV_USART_INDEX_2.

5.1.11.7.4.11 DRV_USART_INDEX_3 Macro

C

```
#define DRV_USART_INDEX_3 3
```

Description

This is macro DRV_USART_INDEX_3.

5.1.11.7.4.12 DRV_USART_INDEX_4 Macro

C

```
#define DRV_USART_INDEX_4 4
```

Description

This is macro DRV_USART_INDEX_4.

5.1.11.7.4.13 DRV_USART_INDEX_5 Macro

C

```
#define DRV_USART_INDEX_5 5
```

Description

This is macro DRV_USART_INDEX_5.

5.1.11.7.4.14 DRV_USART_INDEX_COUNT Macro

C

```
#define DRV_USART_INDEX_COUNT USART_NUMBER_OF_MODULES
```

Description

USART Driver Module Index Count

This constant identifies USART driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is part-specific.

5.1.11.7.4.15 DRV_USART_INSTANCES_NUMBER Macro

C

```
#define DRV_USART_INSTANCES_NUMBER 2
```

Description

USART driver objects configuration

Sets up the maximum number of hardware instances that can be supported

Remarks

None

5.1.11.7.4.16 DRV_USART_INTERRUPT_MODE Macro

C

```
#define DRV_USART_INTERRUPT_MODE
```

Description

USART Interrupt Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values it can take are

- true - Enables the interrupt mode
- false - Enables the polling mode

5.1.11.7.4.17 DRV_USART_INTERRUPT_SOURCE_ERROR Macro

C

```
#define DRV_USART_INTERRUPT_SOURCE_ERROR INT_SOURCE_USART_2_ERROR
```

Description

Error Interrupt Source

This macro defines the interrupt source for the error interrupt. This is the static override of the dynamic initialization passed using [DRV_USART_INIT](#).

5.1.11.7.4.18 DRV_USART_INTERRUPT_SOURCE_RX Macro

C

```
#define DRV_USART_INTERRUPT_SOURCE_RX INT_SOURCE_USART_2_RECEIVE
```

Description

RX Interrupt Source

Macro to define the Rx interrupt source in case of static driver

Remarks

Refer to the INT PLIB document for more information on [INT_SOURCE](#) enumeration

5.1.11.7.4.19 DRV_USART_INTERRUPT_SOURCE_TX Macro

C

```
#define DRV_USART_INTERRUPT_SOURCE_TX INT_SOURCE_USART_2_TRANSMIT
```

Description

TX Interrupt Source

Macro to define the Tx interrupt source in case of static driver

Remarks

Refer to the INT PLIB document for more information on [PLIB_INT_SOURCE](#) enumeration

5.1.11.7.4.20 DRV_USART_LINE_CONTROL Macro

C

```
#define DRV_USART_LINE_CONTROL DRV_USART_LINE_CONTROL_8N1
```

Description

Line Control Mode

This macro defines the line control mode as 8 data bits, even parity and 1 stop bit. It provides a static override of the initialization configuration of the line control mode. The possible values it can take are:

- DRV_USART_LINE_CONTROL_8E1
- DRV_USART_LINE_CONTROL_8E2
- DRV_USART_LINE_CONTROL_8O1
- DRV_USART_LINE_CONTROL_8O2
- DRV_USART_LINE_CONTROL_8N1
- DRV_USART_LINE_CONTROL_8N2
- DRV_USART_LINE_CONTROL_9N1

5.1.11.7.4.21 DRV_USART_OPERATION_MODE Macro

C

```
#define DRV_USART_OPERATION_MODE DRV_USART_OPERATION_MODE_RS232
```

Description

USART mode

This macro controls support modes of the USART driver . The possible values it can take are:

- DRV_USART_OPERATION_MODE_IRDA
- DRV_USART_OPERATION_MODE_RS232
- DRV_USART_OPERATION_MODE_RS485
- DRV_USART_OPERATION_MODE_SYNC_MASTER
- DRV_USART_OPERATION_MODE_SYNC_SLAVE
- DRV_USART_OPERATION_MODE_LOOPBACK

Remarks

None

5.1.11.7.4.22 DRV_USART_PERIPHERAL_ID Macro

C

```
#define DRV_USART_PERIPHERAL_ID
```

Description

Plib id

This macro configures the PLIB Id for the driver. This is the static override of the dynamic initialization passed using [DRV_USART_INIT](#).

5.1.11.7.4.23 DRV_USART_POLARITY_INVERTED_RX Macro

C

```
#define DRV_USART_POLARITY_INVERTED_RX true
```

Description

USART RX Polarity Mode.

This macro defines the polarity of the RX pin. The possible values it can take are

- true - polarity is inverted
- false - polarity is not inverted

Remarks

None.

5.1.11.7.4.24 DRV_USART_POLARITY_INVERTED_TX Macro

C

```
#define DRV_USART_POLARITY_INVERTED_TX true
```

Description

USART TX Polarity Mode.

This macro defines the polarity of the TX pin. The possible values it can take are

- true - polarity is inverted
- false - polarity is not inverted

Remarks

None

5.1.11.7.4.25 DRV_USART_POWER_STATE Macro

C

```
#define DRV_USART_POWER_STATE SYS_MODULE_POWER_IDLE_STOP
```

Description

USART power state configuration

This macro controls the power state of the USART

Remarks

This feature may not be available in the device or the USART module selected

5.1.11.7.4.26 DRV_USART_WAKE_ON_START_ENABLE Macro

C

```
#define DRV_USART_WAKE_ON_START_ENABLE
```

Description

Wake on start enable

This macro configures that the peripheral should wake up on start.

5.1.11.7.4.27 DRV_USART_XFER_BUFFER_NUMBER Macro

C

```
#define DRV_USART_XFER_BUFFER_NUMBER 10
```

Description

USART transfer buffer configuration

This macro controls the transfer buffer depth

Remarks

This feature may not be available in the device or the USART module selected

5.1.11.7.4.28 DRV_USART_BUFFER_FLAGS Enumeration

C

```
typedef enum {
    DRV_USART_BUFFER_FLAG_WRITE,
    DRV_USART_BUFFER_FLAG_WRITE_AFTER_BREAK,
    DRV_USART_BUFFER_FLAG_WRITE_TO_ADDRESS,
    DRV_USART_BUFFER_FLAG_READ_FROM_ADDRESS,
    DRV_USART_BUFFER_FLAG_READ,
    DRV_USART_BUFFER_FLAG_STOP_ON_ERROR
} DRV_USART_BUFFER_FLAGS;
```

Description

USART IO Types

This enumeration defines the IO types that can be serviced by the USART driver.

Members

Members	Description
DRV_USART_BUFFER_FLAG_WRITE	Write Buffer
DRV_USART_BUFFER_FLAG_WRITE_AFTER_BREAK	Write Buffer After Sending Break
DRV_USART_BUFFER_FLAG_WRITE_TO_ADDRESS	Write Buffer to the address
DRV_USART_BUFFER_FLAG_READ_FROM_ADDRESS	Read Buffer from the address
DRV_USART_BUFFER_FLAG_READ	Read Buffer
DRV_USART_BUFFER_FLAG_STOP_ON_ERROR	Instructs the driver to stop processing if error is encountered

Remarks

None

5.1.11.7.4.29 DRV_USART_BUFFER_STATUS Enumeration

C

```
typedef enum {
    DRV_USART_BUFFER_COMPLETED,
    DRV_USART_BUFFER_QUEUED,
    DRV_USART_BUFFER_IN_PROGRESS,
    DRV_USART_BUFFER_ERROR_UNKNOWN,
    DRV_USART_BUFFER_ERROR_RX_OVERFLOW
} DRV_USART_BUFFER_STATUS;
```

Description

USART Buffer Status

Defines the states that an USART buffer can be in during its lifetime

Members

Members	Description
DRV_USART_BUFFER_COMPLETED	Done OK and ready
DRV_USART_BUFFER_QUEUED	Scheduled but not started
DRV_USART_BUFFER_IN_PROGRESS	Currently being in transfer
DRV_USART_BUFFER_ERROR_UNKNOWN	Unknown buffer
DRV_USART_BUFFER_ERROR_RX_OVERFLOW	Read data lost: rx process not fast enough

Remarks

As buffers may have a limited life span, so too have the associated handles and status info. Once a buffer transfer is completed (and possibly notified and acknowledged) it will be discarded. The status of a discarded buffer is no longer maintained by the driver.

5.1.11.7.4.30 DRV_USART_CALLBACK Type**C**

```
typedef void (* DRV_USART_CALLBACK)(void);
```

Description

USART Callback Function

This data type defines a pointer callback function.

Remarks

A USART callback function must have the following function signature:

```
void MyCallBack ( DRV_USART_BUFFER_HANDLE *bufferHandle );
```

Where "MyCallBack" can be any name desired as the routine is called through the pointer.

A pointer of this type is passed into the DRV_USART_RegisterCallbacks routine.

5.1.11.7.4.31 DRV_USART_CLIENT_STATUS Enumeration**C**

```
typedef enum {
    DRV_USART_CLIENT_STATUS_READY,
    DRV_USART_CLIENT_STATUS_BUSY,
    DRV_USART_OVERRUN_ERROR,
    DRV_USART_PARITY_ERROR,
    DRV_USART_FRAMING_ERROR,
    DRV_USART_STATUS_INVALID
} DRV_USART_CLIENT_STATUS;
```

Description

USART Client Status

Defines the various client status codes.

Members

Members	Description
DRV_USART_CLIENT_STATUS_READY	Up and running, ready to start new operations
DRV_USART_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_USART_OVERRUN_ERROR	Overrun Error Occured
DRV_USART_PARITY_ERROR	Parity Error
DRV_USART_FRAMING_ERROR	Framing Error
DRV_USART_STATUS_INVALID	client Invalid

Remarks

None

5.1.11.7.4.32 DRV_USART_HANDSHAKE_MODES Enumeration

C

```
typedef enum {
    DRV_USART_HANDSHAKE_MODE_FLOWCONTROL,
    DRV_USART_HANDSHAKE_MODE_SIMPLEX,
    DRV_USART_HANDSHAKE_MODE_NONE
} DRV_USART_HANDSHAKE_MODES;
```

Description

USART Handshake Modes

This data type identifies the handshaking modes available on the USART module.

Members

Members	Description
DRV_USART_HANDSHAKE_MODE_FLOWCONTROL	Handshaking occurs in Flow Control Mode
DRV_USART_HANDSHAKE_MODE_SIMPLEX	Handshaking occurs in Simplex Mode
DRV_USART_HANDSHAKE_MODE_NONE	No Handshaking

Remarks

Not all modes are available on all micro-controllers.

5.1.11.7.4.33 DRV_USART_INIT Structure

C

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    USART_MODULE_ID usartID;
    uint8_t operationMode;
    uint8_t initFlags;
    DRV_USART_LINE_CONTROL_MODES lineControlMode;
    uint32_t brgValue;
    DRV_USART_OPERATION_MODE_INIT operationModeInit;
    DRV_USART_HANDSHAKE_MODES handShakeMode;
    uint32_t txInterruptSource;
    uint32_t rxInterruptSource;
    uint32_t errorInterruptSource;
    uint32_t rxQueueSize;
    uint32_t txQueueSize;
} DRV_USART_INIT;
```

Description

USART Driver Initialization Data

This data type defines the data required to initialize or reinitialize the USART driver.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
USART_MODULE_ID usartID;	Identifies USART hardware module (PLIB-level) ID
uint8_t operationMode;	Operation Modes of the driver
uint8_t initFlags;	Flags for the usart initialization
DRV_USART_LINE_CONTROL_MODES lineControlMode;	Control the line control configuration

uint32_t brgValue;	Baud Rate value to be used, if not using auto baud
DRV_USART_OPERATION_MODE_INIT operationModeInit;	Operation mode initialization data
DRV_USART_HANDSHAKE_MODES handShakeMode;	Handshake Mode
uint32_t txInterruptSource;	Interrupt Source for TX Interrupt INT_SOURCE txInterruptSource;
uint32_t rxInterruptSource;	Interrupt Source for RX Interrupt INT_SOURCE rxInterruptSource;
uint32_t errorInterruptSource;	Interrupt Source for Error Interrupt INT_SOURCE errorInterruptSource;

Remarks

Not all the init features are available for all the microcontrollers

5.1.11.7.4.34 DRV_USART_INIT_FLAGS Enumeration**C**

```
typedef enum {
    DRV_USART_INIT_FLAG_WAKE_ON_START_ENABLE,
    DRV_USART_INIT_FLAG_AUTO_BAUD_ENABLE
} DRV_USART_INIT_FLAGS;
```

Description

USART Initialization flags

Defines the various flags which determine specific modes

Members

Members	Description
DRV_USART_INIT_FLAG_WAKE_ON_START_ENABLE	Flag to setup Wake on Start Operation
DRV_USART_INIT_FLAG_AUTO_BAUD_ENABLE	Flag to enable auto baud detection enable.

Remarks

None

5.1.11.7.4.35 DRV_USART_IO_BUFFER Structure**C**

```
typedef struct {
    DRV_USART_BUFFER_FLAGS flags;
    char * buffer;
    uint32_t bufferSize;
    uint8_t address;
    DRV_USART_CALLBACK xferCallback;
} DRV_USART_IO_BUFFER;
```

Description

USART Driver IO Buffer

This structure holds information that the client must pass the driver to enqueue a buffer to be transferred by a read or write operation.

Members

Members	Description
DRV_USART_BUFFER_FLAGS flags;	Defines the IO Type this buffer is associated with
char * buffer;	The buffer which needs to be read or written

uint32_t bufferSize;	The size of the buffer which needs to be read or written
uint8_t address;	The address which needs to be written

Remarks

A pointer to an instance of this structure must be passed to the [DRV_USART_BufferAdd](#) routine.

5.1.11.7.4.36 DRV_USART_LINE_CONTROL_MODES Enumeration**C**

```
typedef enum {
    DRV_USART_LINE_CONTROL_MODE_8NONE1,
    DRV_USART_LINE_CONTROL_MODE_9NONE1,
    DRV_USART_LINE_CONTROL_MODE_8EVEN1,
    DRV_USART_LINE_CONTROL_MODE_8EVEN2,
    DRV_USART_LINE_CONTROL_MODE_8ODD1,
    DRV_USART_LINE_CONTROL_MODE_8ODD2,
} DRV_USART_LINE_CONTROL_MODES;
```

Description

USART Line Control Modes

This data type identifies the line control modes available

Members

Members	Description
DRV_USART_LINE_CONTROL_MODE_8NONE1	8 Data Bits, No Parity, 1 Stop Bits
DRV_USART_LINE_CONTROL_MODE_9NONE1	9 Data Bits, No Parity, 1 Stop Bits
DRV_USART_LINE_CONTROL_MODE_8EVEN1	8 Data Bits, Even Parity, 1 Stop Bits
DRV_USART_LINE_CONTROL_MODE_8EVEN2	8 Data Bits, Even Parity, 2 Stop Bits
DRV_USART_LINE_CONTROL_MODE_8ODD1	8 Data Bits, Odd Parity, 1 Stop Bits
DRV_USART_LINE_CONTROL_MODE_8ODD2	8 Data Bits, Odd Parity, 2 Stop Bits

Remarks

None

5.1.11.7.4.37 DRV_USART_OPERATION_MODE_INIT Union**C**

```
typedef union {
    struct {
        uint8_t address;
    } RS485Init;
} DRV_USART_OPERATION_MODE_INIT;
```

Description

Operation Mode Initialization

This data type defines the initialization data required for different operation modes of USART.

Members

Members	Description
struct { uint8_t address; } RS485Init;	Initialization for RS485 mode

uint8_t address;	Address of the device.

Remarks

None

5.1.11.7.4.38 DRV_USART_OPERATION_MODES Enumeration**C**

```
typedef enum {
    DRV_USART_OPERATION_MODE_IRDA = (1 << 0),
    DRV_USART_OPERATION_MODE_RS232,
    DRV_USART_OPERATION_MODE_RS485,
    DRV_USART_OPERATION_MODE_SYNCHRONOUS_MASTER,
    DRV_USART_OPERATION_MODE_SYNCHRONOUS_SLAVE,
    DRV_USART_OPERATION_MODE_LOOPBACK
} DRV_USART_OPERATION_MODES;
```

Description

USART Modes of Operation

This data type identifies the modes of the operation of the USART module.

Members

Members	Description
DRV_USART_OPERATION_MODE_RS232	RS232 Mode (Asynchronous Mode of Operation)
DRV_USART_OPERATION_MODE_RS485	RS485 Mode (Asynchronous Mode of Operation)
DRV_USART_OPERATION_MODE_SYNCHRONOUS_MASTER	Synchronous Master Mode
DRV_USART_OPERATION_MODE_SYNCHRONOUS_SLAVE	Synchronous Slave Mode
DRV_USART_OPERATION_MODE_LOOPBACK	Loopback Mode

Remarks

Not all modes are available on all microcontrollers. Refer to the data sheet for the microcontroller in use.

5.1.11.7.4.39 DRV_USART_TRANSFER_STATUS Enumeration**C**

```
typedef enum {
    DRV_USART_TRANSFER_STATUS_RX_FULL,
    DRV_USART_TRANSFER_STATUS_RX_DATA_PRESENT,
    DRV_USART_TRANSFER_STATUS_RX_EMPTY,
    DRV_USART_TRANSFER_STATUS_TX_FULL,
    DRV_USART_TRANSFER_STATUS_TX_EMPTY
} DRV_USART_TRANSFER_STATUS;
```

Description

USART Driver Transfer Flags

This type specifies the status of the receive or transmit operation.

Members

Members	Description
DRV_USART_TRANSFER_STATUS_RX_FULL	Indicates that the core driver buffer is full
DRV_USART_TRANSFER_STATUS_RX_DATA_PRESENT	Indicates that at least one byte of Data has been received
DRV_USART_TRANSFER_STATUS_RX_EMPTY	Indicates that the core driver receiver buffer is empty

DRV_USART_TRANSFER_STATUS_TX_FULL	Indicates that the core driver transmitter buffer is full
DRV_USART_TRANSFER_STATUS_TX_EMPTY	Indicates that the core driver transmitter buffer is empty

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

5.1.11.7.4.40 DRV_USART_XFER_HANDLE Type**C**

```
typedef uintptr_t DRV_USART_XFER_HANDLE;
```

Description

USART Transfer Handle

Handle for requested transfer..

Remarks

None

5.1.11.8 Files**Files**

Name	Description
drv_usart.h	USART Driver Interface Definition
drv_usart_config_template.h	USART Driver Configuration Template.

Description**5.1.11.8.1 drv_usart.h**

USART Driver Interface Definition

The USART device driver provides a simple interface to manage the USART or UART modules on Microchip microcontrollers. This file defines the interface definition for the USART driver.

Enumerations

	Name	Description
	DRV_USART_BUFFER_FLAGS	Defines the IO types that can be serviced by the USART driver
	DRV_USART_BUFFER_STATUS	Defines the buffer status
	DRV_USART_CLIENT_STATUS	Defines the client status
	DRV_USART_HANDSHAKE_MODES	Identifies the handshaking modes available on the USART
	DRV_USART_INIT_FLAGS	Defines specific initialization features
	DRV_USART_LINE_CONTROL_MODES	Identifies the line control modes available
	DRV_USART_OPERATION_MODES	Identifies the modes of the operation of the USART module
	DRV_USART_TRANSFER_STATUS	Specifies the status of the receive or transmit

Functions

	Name	Description
	DRV_USART_BufferAdd	Starts the writes to the communication channel.
	DRV_USART_BufferStatus	Provides the status of the given buffer
	DRV_USART_BufferTransferStatus	Gets the number of bytes which are written or read
	DRV_USART_ClientStatus	Gets current client-specific status the USART driver
	DRV_USART_Close	Closes an opened-instance of the USART driver
	DRV_USART_Deinitialize	Deinitializes the specified instance of the USART driver module
	DRV_USART_Initialize	Initializes the USART instance for the specified driver index
	DRV_USART_Open	Opens the specified timer driver instance and returns a handle to it
	DRV_USART_Read	Reads data from the USART
	DRV_USART_ReadByte	Reads a byte of data from the USART
	DRV_USART_Reinitialize	Reinitializes the USART driver and refreshes any associated hardware settings
	DRV_USART_Status	Gets the current status of the USART driver module.
	DRV_USART_TasksError	Maintains the driver's error-handling state machine and implements its ISR
	DRV_USART_TasksRX	Maintains the driver's receiver state machine and implements its ISR
	DRV_USART_TasksTX	Maintains the driver's transmitter state machine and implements its ISR
	DRV_USART_TransferStatus	Returns the transmitter and receiver transfer status
	DRV_USART_Write	Writes data to the USART
	DRV_USART_WriteByte	Writes a byte of data to the USART

Macros

	Name	Description
	DRV_USART_INDEX_0	USART driver index definitions
	DRV_USART_INDEX_1	This is macro DRV_USART_INDEX_1.
	DRV_USART_INDEX_2	This is macro DRV_USART_INDEX_2.
	DRV_USART_INDEX_3	This is macro DRV_USART_INDEX_3.
	DRV_USART_INDEX_4	This is macro DRV_USART_INDEX_4.
	DRV_USART_INDEX_5	This is macro DRV_USART_INDEX_5.
	DRV_USART_INDEX_COUNT	Number of valid USART driver indices

Structures

	Name	Description
	DRV_USART_INIT	Defines the data required to initialize or reinitialize the USART driver
	DRV_USART_IO_BUFFER	IO buffer information structure

Types

	Name	Description
	DRV_USART_CALLBACK	Pointer to a USART callback function
	DRV_USART_XFER_HANDLE	Handle for requested transfer..

Unions

	Name	Description
	DRV_USART_OPERATION_MODE_INIT	Defines the initialization data required for different operation modes of USART

File Name

drv_usart.h

Company

Microchip Technology Incorporated

5.1.11.8.2 drv_usart_config_template.h

USART Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

Macros

	Name	Description
	<u>DRV_USART_AUTO_BAUD_ENABLE</u>	Macro configures enabling the auto baud.
	<u>DRV_USART_BAUD_RATE</u>	Macro controls operation of the driver for Baud rate configuration
	<u>DRV_USART_BYTE_Q_SIZE_RX</u>	Macro controls operation of the driver for defining the size of the Rx buffer
	<u>DRV_USART_BYTE_Q_SIZE_TX</u>	Macro controls operation of the driver for defining the size of the Tx buffer
	<u>DRV_USART_CLIENTS_NUMBER</u>	Sets up the maximum number of clients that can be connected to any hardware instance.
	<u>DRV_USART_HANDSHAKE_MODE</u>	Configures the handshake mode
	<u>DRV_USART_INDEX</u>	USART Static Index selection
	<u>DRV_USART_INSTANCES_NUMBER</u>	Sets up the maximum number of hardware instances that can be supported
	<u>DRV_USART_INTERRUPT_MODE</u>	Macro controls operation of the driver in the interrupt mode
	<u>DRV_USART_INTERRUPT_SOURCE_ERROR</u>	Defines the interrupt source for the error interrupt
	<u>DRV_USART_INTERRUPT_SOURCE_RX</u>	Macro to define the Rx interrupt source in case of static driver
	<u>DRV_USART_INTERRUPT_SOURCE_TX</u>	Macro to define the Tx interrupt source in case of static driver
	<u>DRV_USART_LINE_CONTROL</u>	Macro defines the line control mode
	<u>DRV_USART_OPERATION_MODE</u>	Defines the mode to be used for the USART driver
	<u>DRV_USART_PERIPHERAL_ID</u>	Configures the PLIB id
	<u>DRV_USART_POLARITY_INVERTED_RX</u>	Defines the polarity of the RX pin.
	<u>DRV_USART_POLARITY_INVERTED_TX</u>	Defines the polarity of the TX pin.
	<u>DRV_USART_POWER_STATE</u>	Macro controls power state of the USART
	<u>DRV_USART_WAKE_ON_START_ENABLE</u>	Macro configures that the peripheral should wake up on start
	<u>DRV_USART_XFER_BUFFER_NUMBER</u>	Transfer buffer depth

File Name

drv_usart_config_template.h

Company

Microchip Technology Incorporated

5.1.12 USB Driver Library

5.1.12.1 Introduction

USB Device Driver Library

for

Microchip Microcontrollers

This library provides an interface to manage the USB module on Microchip family of micro controllers during different modes of operation.

Description

Overview of the USB Module

The USB module implements the physical layer in a USB based system. In that, it allows the USB stack software to access and control the USB. The USB module on Microchip's micro controllers are compliant with the USB 2.0 specification. This help file only provides relevant information about the operation of the USB. The reader is encouraged to refer to the USB 2.0 specification available at www.usb.org for complete information about the USB.

Not all Microchip microcontrollers feature a USB module. Among the ones that do, the supported features may vary. Please refer to the device specific data sheet for details on the availability of the USB module and the supported features.

Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software.

5.1.12.2 Release Notes

MPLAB Harmony Version: v0.70b **USB Driver Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

- Supports all four (Control, Bulk, Interrupt and Isochronous) USB Transfers.
- Control transfers supported on endpoint 0 only.
- Endpoint 0 pipe is always bidirectional. All other endpoint pipes require Receive and Transmit pipes to be opened separately.
- Supports multiple clients. Device function drivers can communicate on the USB directly via the USB Driver.
- Setup packet reception is automatic and does not require client intervention or scheduling. Clients receive notification when a Setup packet is received.
- The static multi client version of the driver support one transfer object per Control, Bulk and Interrupt Pipe and supports 2 transfer objects per Isochronous Pipe.

Known Issues:

- Nothing to report in this release

5.1.12.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.12.4 Using the Library

This topic describes the basic architecture of the USB Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_usb.h](#)

The interface to the USB Driver library is defined in the "[drv_usb.h](#)" header file.

Please refer to the [MPLAB Harmony Overview](#) for how the Driver interacts with the framework.

5.1.12.4.1 Library Overview

Refer to the section Driver Overview for how the driver operates in a system. The library interface routines are divided into various subsections, each of the sub section addresses one of the blocks or the overall operation of the USB module.

Library Interface Section	Description
System Functions	These routines are typically invoked by the MPLAB Harmony system module.
Client Functions	These routines are used by the driver clients to access driver functionality
Pipe Functions	These routines allow clients to manage pipes
Transfer Functions	These routines allow clients to manage transfers
Other Functions	These routines represent the rest of the driver functionality
Data Types and Constants	These data types and constants are required while interacting and configuring the driver

5.1.12.4.2 Abstraction Model

Figure 1: USB Driver Device Mode Model

Figure 1 shows the functioning of the USB Driver in device mode. In this mode, the driver offers services required to operate the

USB module as a USB device. The USB Driver supports multiple clients. Each client can receive notifications about USB events via callback functions. Typical clients could be the Device Layer firmware and one or more function drivers. The Device layer firmware and the function drivers communicate on the USB via pipes. A pipe is an association between the device's endpoint and host stack software. An Endpoint 0 Pipe is always bi-directional and is a message pipe. Communication on this pipe is defined by the USB specification. All other endpoints support stream pipes. The format of the data on a stream pipe is not specified by the USB specification. Stream pipes are uni-directional. Therefore, two pipes are required to transmit and receive data on a non-zero endpoint.

Clients schedule USB transfers through a pipe. Transfers on non-zero endpoint pipes can be queued. The USB driver manages the transactions required to complete the transfer. This is transparent to the clients. Clients can either get an event notification or can poll to check the completion status of a transfer. Clients can register callback functions to receive USB events notification. A Transfer complete callback function can be registered while requesting a transfer. This function will be called when the transfer is terminated. While operating in device mode, the first client of the USB Driver should always be the USB Device Stack Device layer. Only the first client can schedule control transfers on endpoint 0 (via a pipe) and receive the Setup Token and Setup Handshake events (along with the other events). Other clients may or may not register an event call back function.

The USB Driver uses configuration macros that control among other things, functional aspects such as maximum number of clients and maximum pipes.

5.1.12.4.3 How the Library Works

Before the driver is ready for use, its should be configured. Refer to the Configuring the Library section for more details on how to configure the driver.

To use the USB Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation. The following is the sequence in which various routines should be called:

1. Call the `DRV_USB_Init()` function to initialize the USB Driver. Note that this may be performed by the MPLAB Harmony system module.
2. In USB device mode operation, the first client should always be the USB Device Stack Device Layer. The Device Layer first opens the driver via the `DRV_USB_Open()` function.
3. The Device layer registers an event call back function via the `DRV_USB_Device_EventCallBackSet()` function.
4. The Device layer assigns a setup packet buffer by using the `DRV_USB_Device_SetupPacketBufferAssign()` function. The driver stores the received setup packet in this buffer. Calling this function also enables the USB Device Mode and at this point the device will attach to the host and be ready to receive setup tokens from the host.
5. The Device Layer waits for the Reset Event. In this event, the device layer creates an endpoint 0 pipe by using the `DRV_USB_Device_PipeSetup()` function. The Device layer can optionally register a transfer complete call back function. This function will be called when a transfer on the endpoint zero is complete.
6. Based on the device enumeration process and the selected configuration, other function drivers may open the driver by using the `DRV_USB_Open()` function.

It is strongly recommended that the above sequence be followed for correct operation on the driver.

Note: Not all modes are available on all the devices, please refer to the specific device data sheet for the set of modes supported.

5.1.12.4.3.1 Driver Initialization

At the time of initialization, the `DRV_USB_Initialize()` function requires `DRV_MODULE_INIT` data structure that provides information required for the function of the driver. The following data is needed:

1. Maximum number of endpoints to be supported by the driver. In case endpoints are skipped, the largest endpoint number

should be used. For example, if the application uses endpoints 0,1 and 2, then maximum number of endpoints should 3. If the application uses endpoints 0,1 and 5, then maximum number of endpoints should 6. The latter configuration is not recommended.

2. Maximum size of the endpoint 0. This should match the maximum size that is reported to the host during enumeration.
3. Pointer to byte array whose size is defined `DRV_USB_WORKING_BUFFER_SIZE(nEndpoint, maxEP0Size)`, where `nEndpoint` is the maximum number of endpoint as described in point 1 above and `maxEP0Size` is the maximum endpoint 0 size as described in point 2 above. These values should match `usbInitData.ep0MaxSize` and `usbInitData.nEndpoints`.
4. The PLIB peripheral ID of the USB module to associated with the driver object being initialized.
5. The operation mode of the driver (host, device or OTG). Refer to the [Release Notes](#) for supported modes.
5. The ping pong mode to be used by internal endpoint buffers. Refer to the device specific data sheet for supported ping pong modes.
6. Behavior of the USB module when the microcontroller device enters IDLE mode.
7. Behavior of the USB module when the microcontroller device enters SLEEP mode.
8. Power state of the USB module.
9. USB module interrupts to be enabled. These include the general, error and OTG interrupts. Refer to the USB PLIB documentation for available module interrupts.
10. The interrupt source of the USB module to be used with the driver instance. Refer to the USB PLIB documentation for available module interrupts.

The following code snippet shows an example initialization of the driver.

```
/* The following code snippet shows an example initialization of the
 * driver. The USB module to be used in USB1. The maximum number of
 * endpoints to be used is 3 i.e endpoint 0,1 and 2. Maximum size of
 * endpoint 0 is 64. The module should not automatically suspend when the
 * micro-controller enters Sleep mode. The module should continue
 * operation when the module enters Idle mode. The ping pong setting
 * is set to full ping pong (both RX and TX endpoints have ping
 * pong buffers). The power state is set to run at full clock speeds.*/

DRV_USB_INIT moduleInit;

uint8_t internalBuffer[[DRV_USB_WORKING_BUFFER_SIZE(3,64)]
                        __attribute__((aligned(512)))];

usbInitData.usbID = USB_ID_1;
usbInitData.usbModuleSetup.OpMode = USB_OPMODE_DEVICE;
usbInitData.usbModuleSetup.ppMode = USB_PING_PONG__FULL_PING_PONG;

usbInitData.usbModuleSetup.StopInIdle      = false;
usbInitData.usbModuleSetup.SuspendInSleep = false;

usbInitData.usbModuleSetup.otgInterruptEnables = ~USB_OTG_INT_ALL ;
usbInitData.usbModuleSetup.generalInterruptEnables = USB_INT_ALL & ~USB_INT_SOF ;
usbInitData.usbModuleSetup.errorInterruptEnables = USB_ERR_INT_ALL ;

usbInitData.workingBuffer = internalBuffer;
usbInitData.ep0MaxSize = 64;
usbInitData.nEndpoints = 3;
usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

usbInitData.interruptSource = INT_SOURCE_USB_1;

/* This is how this data structure is passed to the initialize
 * function. */
```

```
DRV_USB_Initialize(DRV_USB_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);
```

5.1.12.4.3.2 Opening the Driver

Once the driver is initialized for device mode, it can be opened by clients. The client must specify which driver instance it wants to be a client to and IO mode in which the driver should be opened. The USB Device Driver while operating in USB device mode only supports non-blocking, read write and shared client access I/O intent mode. Trying to open the driver in any other mode will cause the open routine to fail.

The driver treats the first client that opens it, differently from the other clients. This first client must open a bi-directional pipe on endpoint zero. It must then register a Setup Packet Buffer with the driver. The driver uses this buffer to store the 8 byte Setup command details received during the Setup stage of a Control transfer. While all clients can register a USB event callback function, only the first client receives the USB_DEVICE_SETUP_TOKEN and the USB_DEVICE_SETUP_HANDSHAKE event in its callback routine.

Typically (and it is recommended that) the first client to open the driver should be the USB Device Stack Device Layer Firmware. The Device Layer firmware is responsible for enumerating the USB device and responding to host requests on the Endpoint 0 message pipe. The first client should register a USB event callback function (by using the DRV_USB_Device_EventCallbackSet() function) and Setup Packet Buffer (by using the DRV_USB_Device_SetupPacketBufferAssign() function) with the Driver. Both of these are essential for operation of the driver. Registering a Setup Packet Buffer via the DRV_USB_Device_SetupPacketBufferAssign() also enables the USB device (activates the USB module pull up resistors) for operation on the USB.

The following code snippet shows an example of how a USB Device Stack Device Layer should open the driver and perform other functions.

```
/* In this example, the USB Device Layer (myUSBDevice) opens
 * the USB Driver. It is assumed that the Driver is already
 * initialized */

myUSBDevice.usbDriverHandle = DRV_USB_Open(DRV_USB_INDEX_0,
                                           DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_SHARED );

if(DRV_HANDLE_INVALID == myUSBDevice.usbDriverHandle)
{
    /* The open function failed. We can find out why */
    if(USB_CLIENT_NOTAVAIL == DRV_USB_Device_DriverClientErrorGet(DRV_USB_INDEX_0))
    {
        /* This means that there is no space for another client.
         * The driver configuration needs to be modified to support
         * more clients. This can be done by increasing the value
         * of DRV_USB_CLIENTS_NUMBER */

        while(1);
    }
}

/* Set the client event callback for the Device Layer.
 * The USBDeviceLayerEventHandler() function is the event
 * handler. When this event handler is invoked by the
 * driver, the driver returns back the second argument
 * specified in the below function (which in this case
 * is the device layer data structure). This allows the
 * application firmware to identify, as an example, the
 * device layer object associated with this callback
 */

DRV_USB_Device_EventCallbackSet(myUSBDevice.usbDriverHandle, (void *)&myUSBDevice,
                               USBDeviceLayerEventHandler);

/* Device Layer register setup packet buffer. The Setup Packet Buffer
 * size is 8 bytes and is of the type uint8_t. Calling this function
```

```

    * enables the device and the device is ready to receive a Setup token
    * from the host. */

DRV_USB_Device_SetupPacketBufferAssign(myUSBDevice.usbDriverHandle, setupPacketBuffer);

/* Device Layer opens a endpoint 0 pipe. Refer to Setting
 * up a Pipe section for more details on how to set up the
 * pipe. This may be done in the reset event.*/

myUSBDevice.ep0Pipe = DRV_USB_Device_PipeSetup(myUSBDevice.usbDriverHandle, 0,
USB_EP_TX_RX,
                                     64, 1, USB_CONTROL_PIPE);

```

A function driver may open the driver for use based on the configuration set by the host. Function drivers may or may not require USB Event notification and so may not need to register a USB Event Callback function. The user application can also open the driver. This allows the user application to track USB events and also communicate directly on the bus. The following code snippet shows an example of how a function driver can open the USB Driver.

```

/* This code snippet shows an example of how the
 * a function driver, say CDC, could open the USB Driver.
 * Note that this should happen on a set configuration
 * event from the host. */

cdc.usbDriverHandle = DRV_USB_Open(DRV_USB_INDEX_0,
    DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE );

if(DRV_HANDLE_INVALID == cdc.usbDriverHandle)
{
    /* The open function failed. We can find out why */
    if(USB_CLIENT_NOTAVAIL == DRV_USB_Device_DriverClientErrorGet(DRV_USB_INDEX_0))
    {
        /* This means that there is no space for another client.
         * The driver configuration needs to be modified to support
         * more clients. This can be done by increasing the value
         * of DRV_USB_CLIENTS_NUMBER */

        while(1);
    }
}

/* Set the USB Event handler for CDC. Note that
 * this optional and should be done only if the
 * function driver requires USB Events notification. */

DRV_USB_Device_EventCallbackSet(cdc.usbDriverHandle, (void *)&cdc,
    CDCDeviceUSBEventHandler);

/* CDC Function driver opens a TX and RX bulk pipe on
 * endpoint 1. */

cdc.bulkRxPipe = DRV_USB_Device_PipeSetup(cdc.usbDriverHandle, 1,
    USB_EP_RX, 64, 3, USB_BULK_PIPE);

cdc.bulkTxPipe = DRV_USB_Device_PipeSetup(cdc.usbDriverHandle, 1,
    USB_EP_TX, 64, 3, USB_BULK_PIPE);

```

5.1.12.4.3.3 Endpoint Pipes

A client must open a pipe in order to schedule transfers on the USB. This is done by using the DRV_USB_Device_PipeSetup() function. From a USB perspective, an endpoint 0 pipe is a message pipe (or a control pipe). The data flowing through this pipe has a structure which defined by the USB specification. Pipes on non-zero endpoints are stream pipes and need not follow a specific format. A Pipe connects a device's endpoint to a Host Driver software. Endpoint 0 pipe is always needed and must be created for the USB device to enumerate and hence operate correctly. Non-zero endpoint pipes are created based on the configuration set by the Host.

5.1.12.4.3.3.1 Pipe Setup

The DRV_USB_Device_PipeSetup() function creates a pipe. The following points should be noted while using this function to create a pipe.

1. The pipeDirection argument specifies the pipe direction. An endpoint 0 pipe is always bi-directional. So this parameter is ignored when the nEndpoint parameter is 0. All other pipes must be unidirectional. Specifying USB_EP_TX as the pipe direction sets up the pipe to **send** data from device to host (host Read transfers). Specifying USB_EP_RX as the pipe direction sets up the pipe to receive data from host (host Write transfers).
2. The nBytesMaxSize parameter specifies the maximum size of the data packet in a transaction on that pipe. It should not exceed the maximum endpoint size communicated to the host during enumeration. Depending on the type of the USB transfer that the pipe will handle and the speed of the USB, the maximum size of the data packet may vary. If the size of the read transfer (device to host) scheduled on the pipe is greater than nBytesMaxSize, the driver will respond correctly multiple transactions that are issued by the host to complete the transfer.
3. The nXferQueueSize parameter defines the size of the transfers that can be queued up this pipe. The minimum queue size should be 1. The control pipe (endpoint 0 pipe) always has queue size of 1. Control transfers on the endpoint 0 pipe cannot be queued. Refer to the Transfer Request section for more information on how transfers are scheduled. Other non-zero endpoint pipes can have a queue size greater than or equal to 1.
4. An endpoint and direction of communication cannot be shared between two pipes. For example if a client opens a RX pipe on endpoint 1, another RX pipe on endpoint 1 cannot be opened.
5. The pipeType parameter specified the type of transfers that this pipe will handle.

The client can check the return value of the DRV_USB_Device_PipeSetup() for the result. If the pipe was not setup successfully, the function returns **DRV_HANDLE_INVALID**. The client can then use the DRV_USB_Device_ClientPipeErrorGet() function to find out the reason for the failure.

The following code snippet shows how endpoint 0 pipe and non-zero endpoint pipes are created.

```
/* This is an example of how an endpoint 0 pipe is created.
 * Note that this pipe is always bi-directional and the
 * queue size is always 1. The maximum data packet size in this
 * case is 64. The pipe type is set to handle control transfers */

myUSBDevice.ep0Pipe = DRV_USB_Device_PipeSetup(myUSBDevice.usbDriverHandle, 0,
USB_EP_TX_RX,
                                           64, 1, USB_CONTROL_PIPE);

if(DRV_HANDLE_INVALID == myUSBDevice.ep0Pipe)
{
    switch(DRV_USB_Device_ClientPipeErrorGet(myUSBDevice.usbDriverHandle))
    {
        case USB_PIPE_QUEUE_NOT_AVAILABLE:
            /* This means that queue size is small. Increase
             * the value of DRV_USB_XFERS_NUMBER. */

            break;

        case USB_PIPE_BAD_ENDPT:
            /* The specified endpoint is greater than the
             * the maximum end points that driver is configured
             * to handle. */

            break;

        case USB_PIPE_TAKEN:
```

```

        /* The specified endpoint and direction are already
        * in use with another pipe. A non-zero endpoint
        * can have only one pipe in a direction. Endpoint
        * 0 can have one bi-directional pipe.*/

        break;

    case USB_PIPE_NOTAVAIL:

        /* No pipes are available. Increase the number of pipes
        * by increasing the value of DRV_USB_PIPES_NUMBER. */

        break;

    case USB_PIPE_BAD_DIRECTION:

        /* Non zero endpoint cannot be bi-directional. */

        break;
}

/* This is an example of how a client creates a non-zero endpoint pipe.
* In this case a TX and a RX pipe are opened on endpoint 1. The
* transfer schedule queue size is 4 and maximum data packet size
* is 64. Note that the maximum data packet size should match endpoint
* size specified in the endpoint configuration. */

cdc.bulkRxPipe = DRV_USB_Device_PipeSetup(cdc.usbDriverHandle, 1, USB_EP_RX, 64, 3,
                                           USB_BULK_PIPE);

/* Optionally check for error here by using the
* DRV_USB_Device_ClientPipeErrorGet() function */

cdc.bulkTxPipe = DRV_USB_Device_PipeSetup(cdc.usbDriverHandle, 1, USB_EP_TX, 64, 3,
                                           USB_BULK_PIPE);

/* Optionally check for error here by using the
* DRV_USB_Device_ClientPipeErrorGet() function */

/* The following usage of DRV_USB_Device_PipeSetup() is INCORRECT and
* will not work. The client must create separate TX and RX pipes for
* non-zero endpoints. This function will return DRV_HANDLE_INVALID
* and the DRV_USB_Device_ClientPipeErrorGet() function will return
* USB_PIPE_BAD_DIRECTION. */

someDevice.bulkPipe = DRV_USB_Device_PipeSetup(someDevice.usbDriverHandle, 1,
                                                USB_EP_TX_RX, 64, 3, USB_BULK_PIPE); // Will not work!

```

5.1.12.4.3.3.2 Pipe Status

The client can use the DRV_USB_Device_PipeStatusGet() function to obtain the status of the pipe. The client can use this information to find if a pipe is ready for transfers or if the pipe can be or is stalled. The following code snippet shows an example of the DRV_USB_Device_PipeStatusGet() function along with all the possible return values.

```

/* This code example assumes the DRV_USB_DevicePipeSetup() function
* was used to open a pipe */

DRV_USB_PIPE_HANDLE pipeHandle;
DRV_USB_PIPE_STATUS pipeStatus;

if(DRV_HANDLE_INVALID != pipeHandle)
{
    pipeStatus = DRV_USB_Device_PipeStatusGet(pipeHandle);
    switch (pipeStatus)
    {
        case USB_PIPE_CLOSED:

```



```

        /* The pipe was open and has been closed.
        * This pipe should not be used for transferring
        * data */
        break;
    case USB_PIPE_XFER_IN_PROGRESS:
        /* There is a transfer in progress on the USB
        * through this pipe.
        */
        break;
    case USB_PIPE_READY:
        /* Pipe is ready for transferring data. Also
        * indicates that there is no transfer in progress
        * on the USB through this pipe
        */
        break;
    case USB_PIPE_STALLED:
        /* Pipe is stalled. No transfers will take place.
        */
        break;
}
}

```

5.1.12.4.3.3.3 Pipe Close

An open pipe can be closed by using the DRV_USB_Device_PipeRelease() function. Calling this function performs the following function

1. It aborts all transfers that are in queue. If a transfer is in progress on the USB, it is cancelled.
2. It de-allocates its transfer queue and returns it to the driver.
3. It de-allocates the pipe and returns its to the driver.

The DRV_USB_Device_PipeRelease() may be called when the device stack detects detach from the host, or when a USB reset event had been detected. Because this function aborts transfers that are in progress on the USB, it is possible that calling this function while a transfer is in progress can cause the USB protocol to be violated. To avoid this, the client can use the DRV_USB_Device_PipeStatusGet() function to get the status of the pipe before closing it. In case of a USB reset event, the client can close the pipe without checking pipe status as the driver itself will cancel all transfer that are in progress.

The following code snippet shows example usage of the DRV_USB_Device_PipeRelease() function.

```

/* This code snippet assumes that this is not a reset
 * event. In such a case, it is recommended that any
 * transfers through the pipe be allowed to complete.
 */

while(DRV_USB_Device_PipeStatusGet(pipeHandle) == USB_PIPE_XFER_IN_PROGRESS);

/* Close the pipe */

DRV_USB_Device_PipeRelease(pipeHandle);

```

5.1.12.4.3.3.4 Pipe Transfer Queue

A pipe provides a queue for transfer requests. Clients can use the queue to reduce wait times and hence increase data throughput through the pipe. If the pipe queue size is greater than one, the transfer requests on the pipe will be added to the queue if a transfer is already in progress on the pipe. The size of queue is specified via the nXferQueueSize parameter in the DRV_USB_Device_PipeSetup() function. The queue contains transfer objects that hold information about the requested transfers. Multiple driver instance share one aggregate queue. The size of aggregate queue is defined by the DRV_USB_XFERS_NUMBER macro in the driver configuration. Pipe specific queues are formed from this aggregate queue. Figure 2 shows how this works.

Figure 2: Pipe Transfer Queues.

Figure 2a shows the aggregate driver queue. In this case, all the transfer objects in the queue are available for use. In figure 2b, the client creates a pipe (pipe 0) with a queue size of 1. This results in the first transfer object on the queue forming a single element queue for pipe 0. In figure 2c, the client creates a pipe (pipe 1) with a queue size of 3. This results in the next three transfer objects on the queue forming a 3 element queue for pipe 1. In figure 2d, the client creates a pipe (pipe 2) with a queue size of 4. This results in the next four transfer objects on the queue forming a 4 element queue for pipe 2. At this point, if a client tries to create another pipe with queue size of say 4, the pipe will not be created as there isn't enough space available in the queue.

The size of the aggregate queue should be decided in such a way as to meet the application's dynamic requirements while taking the memory availability into consideration.

5.1.12.4.3.3.5 Pipe Stall

The client can stall a pipe in response to USB events. When a pipe is stalled, it returns a STALL PID on the USB. The USB device firmware may stall endpoint 0 pipe when control requests from the host fail or are not supported. The driver automatically clears an endpoint 0 pipe stall when the host sends a SETUP packet. A non-zero endpoint pipe may be stalled when the endpoint's Halt feature is set. This is a functional stall. In this case the endpoint cannot [send](#) or receive data. Endpoint 0 pipe may not need to support functional stalls (although they can but this is rarely done). A pipe is stalled using the `DRV_USB_Device_PipeStall()` function and a pipe stall is cleared using the `DRV_USB_Device_PipeStallClear()` function.

There are 3 execution threads where the driver client can stall a pipe.

1. A pipe can be stalled in the client event handler.

The client can stall the pipe after receiving the `USB_DEVICE_SETUP_TOKEN` event. This is relevant to control transfer on endpoint 0 pipe. The firmware can stall the pipe if it does not support the command received in the Setup packet. It is recommended that both direction of the endpoint 0 pipe be stalled (this is possible because endpoint 0 pipe is always bi-directional). Figure 3 shows the transaction related to this case.

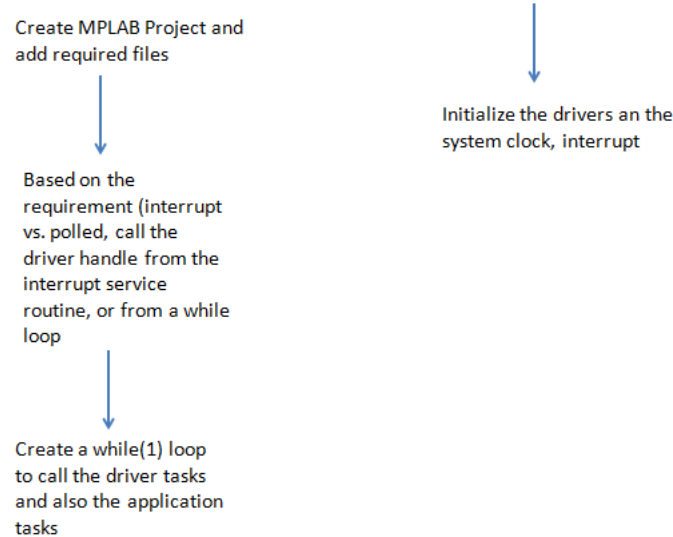


Figure 3: Stalling Endpoint 0 pipe on a Control Read Transfer. Pipe is stalled after `USB_DEVICE_SETUP_TOKEN` event.

The stall on the receive direction of the pipe will get cleared when the host issues a Setup token. The stall on the TX direction of the pipe needs to be cleared by using the `DRV_USB_Device_PipeStallClear()` function. For endpoint 0 pipe, it is recommended that the stall on both directions be cleared. Note that it is also possible to stall non-zero pipes in the `USB_DEVICE_SETUP_TOKEN` event. See code example below.

```
void USBEventHandler (void * referenceHandle,
```

```

        DRV_USB_EVENT  eventType, DRV_USB_EVENT_DATA * eventData )
{
    switch ( eventType )
    {
        /* There could be other events here as well. Only
         * the USB_DEVICE_SETUP_TOKEN is shown in this example
         * for the sake of clarity */

        case USB_DEVICE_SETUP_TOKEN:
            /* Setup token received from host */

            if(ProcessSetupPacket(eventData->setupEventData.pSetupPktBuffer) == false)
            {
                /* This means the command received in the setup packet is
                 * not supported. The endpoint should be stalled. While
                 * stalling endpoint 0, both directions should be stalled.
                 * This ensures that both data stage and handshake stage
                 * of the control transfer get stalled. */

                DRV_USB_Device_PipeStall(pipe0, USB_EP_TX_RX);
            }
            else
            {
                /* Setup packet command can be processed */
                if(DRV_USB_Device_PipeStatusGet(pipe0) == USB_PIPE_STALLED)
                {
                    /* Clear the pipe stall */
                    DRV_USB_Device_PipeStallClear(pipe0, USB_EP_TX_RX);
                }
            }
            break;
    }
}

```

2. A pipe can be stalled in the transfer complete event handler.

This is more relevant to Control Write transfers. In the case of Control Write transfers, the transfer complete event handler is called after the data stage of the control transfer. If the firmware does not support the contents of the data stage, it can stall the pipe. This will result in the handshake stage of the control write transfer getting stalled. Figure 4 shows the transactions related to this case.

Figure 4: Stalling Endpoint 0 pipe on a Control Write Transfer. Pipe is stalled in the Transfer Complete Callback Function.

The following code snippet shows an example of this:

```

void USBDeviceLayerXferCallback( void * referenceData,
                                DRV_USB_PIPE_HANDLE hPipe,
                                DRV_USB_XFER_HANDLE hTransfer,
                                unsigned short int transferByteCount ,
                                DRV_USB_DEVICE_XFER_STATUS statusTransfer )
{
    /* Note that the transfer call contains the pipe
     * handle. This pipe handle can be used to
     * stall the pipe */

    DRV_USB_Device_PipeStall(hPipe, USB_EP_TX_RX);
}

```

3. A pipe can be stalled asynchronous of driver events.

The client can stall a pipe based on device firmware logic. This is more applicable to non-zero endpoint pipes. Since these pipes are uni-directional, the client must specify the stall direction (USB_EP_TX or USB_EP_RX) while calling the DRV_USB_Device_PipeStall() function. Using the USB_EP_TX parameter will stall read transfers (IN tokens). Using the

USB_EP_RX parameter will stall write transfers (OUT tokens).

5.1.12.4.3.4 Event Handling

Any client that opens the driver can register to receive USB events. These USB events can be used by the clients to achieve the desired functionality of the USB device. The first client to open the driver (typically the USB Stack device layer firmware) will receive additional events. The client uses the DRV_USB_Device_EventCallBackSet() to set the client event call back function. The following code snippet shows a client event handler function.

```
/* This is the client event handler */
void ClientEventHandler (void * referenceHandle,
                        DRV_USB_EVENT eventType,
                        DRV_USB_EVENT_DATA * eventData )
{
    switch ( eventType )
    {
        case USB_ERROR:
            // Bus error occurred and was reported in error interrupts
            break;
        case USB_DEVICE_RESET:
            // Host has commanded a device reset
            break;
        case USB_DEVICE_RESUME:
            // Resume detected while USB in suspend mode
            break;
        case USB_IDLE_DETECT:
            // Idle detected
            break;
        case USB_STALL:
            // Stall handshake has occurred
            break;
        case USB_DEVICE_SETUP_TOKEN:
            // A Setup token was received
            break;
        case USB_DEVICE_SETUP_HANDSHAKE:
            // Handshake stage of the control
            // transfer is completed.
            break;
        case USB_DEVICE_GOT_SOF:
            // Device received a Start of Frame
            break;
        default:
            break;
    }
}

/* The client register the event handler
 * after it has opened the driver. referenceData
 * will be passed back to the event callback function
 * whenever it is invoked. */
```

DRV_USB_Device_EventCallBackSet(clientHandle, referenceData, ClientEventHandler);

The eventData argument in the event callback function should be interpreted according to the event. The following table shows all the possible events and the value of eventData for each event.

Event Type	Event Details	Interpretation of eventData
USB_ERROR	This event occurs when the USB module has detected an error on the bus	eventData.usbErrorData.errorType contains a bitmap of errors (defined by USB_ERROR_TYPE) that have occurred.
USB_DEVICE_RESET	This event occurs when the USB module has detected a USB Reset	NULL

USB_DEVICE_RESUME	This event occurs when the USB module has detected Resume Signalling	NULL
USB_IDLE_DETECT	This event occurs when the USB module has not detected any bus activity	NULL
USB_STALL	This event occurs when a stalled endpoint has received a token	eventData.stalledEndpoint.endpoint contains the stalled endpoint that received a token.
USB_DEVICE_SETUP_TOKEN	This event occurs when endpoint 0 has received a SETUP token	eventData.setupEventData.pSetupPktBuffer points to a 8 byte array that contains the setup command that was received.
USB_DEVICE_SETUP_HANDSHAKE	This event occurs when the handshake stage of a Control Write Transfer has completed.	NULL
USB_DEVICE_GOT_SOF	This event occurs when the USB module has received a start of frame.	NULL

The following code snippet shows the different USB errors that can occur

```
USB_ERROR_TYPE error = eventData.usbErrorData.errorType;
```

```
/* These are the possible errors that can occur.
 * Refer to the device specific data sheet for
 * more details on these errors. */

if (errorType & USB_ERR_INT_PID_CHECK_FAILURE)
{
    /* Invalid or unsupported PID received */
}
if ( errorType & USB_ERR_INT_BAD_CRC5)
{
    /* A CRC 5 error was encountered */
}
if ( errorType & USB_ERR_INT_BAD_CRC16)
{
    /* A CRC 16 error was encountered */
}
if ( errorType & USB_ERR_INT_BAD_DATA_FIELD_SIZE)
{
    /* Unsupported data field size was encountered */
}

if ( errorType & USB_ERR_INT_BUS_TURNAROUND_TIMEOUT)
{
    /* A bus turn around timeout was encountered */
}

if ( errorType & USB_ERR_INT_DMA_ERROR)
{
    /* A USB DMA error was encountered */
}

if ( errorType &= USB_ERR_INT_BUS_MATRIX_ERROR)
{
    /* A Bus matrix error was encountered */
}

if ( errorType & USB_ERR_INT_BIT_STUFF_ERROR)
{
    /* A bit stuffing error was encountered */
}
```

5.1.12.4.3.5 Transfer Requests

A USB Driver (operating in Device mode) client can receive and transmit data from the USB Host after creating a pipe. Data is transferred by using the `DRV_USB_Device_TransferRequest()` function. Four types of USB transfers are supported: Control, Bulk, Interrupt and Isochronous. The choice of transfer type depends on the nature of desired communication and the target functionality of the USB device. Requesting a transfer will cause the USB driver to schedule the transfer. The actual data communication takes place when the host has requested for the data. In cases where a transfer is already in progress and space is available on the pipe transfer queue, the transfer request is added to the queue and processed in turn. Transfers that are not in progress (pending in the queue) can be aborted by using the `DRV_USB_Device_TransferAbort()` function.

Depending on the size of the endpoint and the amount of data to be transferred, the host may complete the transfer through multiple transactions. This process is handled by the driver and is transparent to the client. The client can register a transfer complete event callback function for each transfer request. This function is called when the transfer is terminated, either because its completed or aborted. Optionally the client can call the `DRV_USB_Device_TransferStatusGet()` function to poll the status of the transfer. The transfer complete event callback function need not be registered in this case.

The `DRV_USB_Device_TransferRequest()` function also accepts flags that affect the processing of the transfer. The flag options are transfer type specific.

The following section provides more details on how the four different transfer types can be requested:

5.1.12.4.3.5.1 Control Transfers

The USB Driver supports Control Read and Write Transfers on a endpoint 0 pipe only. A control transfer (either read or write) occurs in stages:

1. The SETUP stage provides a setup command which among other things also defines if the transfer is a Control Read (data moves from device to host) or a Control Write transfer (data moves from host to device).
2. The DATA stage transports data related to the setup command in the SETUP stage. Depending on the size of the endpoint, the data stage could comprise of multiple IN or OUT transactions. A data stage may or may not be present in Control Write transfers.
3. The HANDSHAKE stage completes the Control transfer.

Control Transfers should only be requested in response to a setup packet from the host. The USB Driver notifies its first client (via the client event call back function) when a SETUP packet has been received. Only the first client receives this, `USB_DEVICE_SETUP_TOKEN`, event. The driver generates `USB_DEVICE_SETUP_HANDSHAKE` client event at the completion of the handshake stage of the control write transfer. Refer to Event Handling section on more details on handling events.

The client parses the 8-byte setup command available from the `USB_DEVICE_SETUP_TOKEN`, event. It then has the following options:

1. The client can request a control read transfer.
2. The client can request a control write transfer.
3. The client may not do anything and return from the event handler. In this case, the control transfer is treated as having no data stage and the transfer moves to the handshake stage. Note that a Control Read transfer must always have a data stage.
4. The client can stall the pipe.

These cases are described here in detail.

1. Requesting a Control Read Transfer:

The code snippet below shows an example of how a Control Read Transfer can be requested. In this transfer, data moves from the device to the host.

```

/* This is an example of a Control Read Transfer. A control read
 * or Write transfer should be requested in response to a
 * USB_DEVICE_SETUP_TOKEN event. The transfer request function is
 * called in the USB event handler. In this example, the amount of
 * data to be transferred is 18 bytes. The transfer is requested
 * over endpoint 0 pipe. The USBDeviceLayerXferCallback() function
 * will be called when the transfer is terminated. The myUSBDevice
 * object will be passed back to the USBDeviceLayerXferCallback()
 * function. The USB_XFER_REGULAR indicates to the driver that this
 * transfer does not need a data stage ZLP */

DRV_USB_XFER_HANDLE xferHandle;

void USBEventHandler (void * referenceHandle,
                     DRV_USB_EVENT eventType, DRV_USB_EVENT_DATA * eventData )
{
    switch ( eventType )
    {
        /* There could be other events here as well. Only
         * the USB_DEVICE_SETUP_TOKEN is shown in this example
         * for the sake of clarity */

        case USB_DEVICE_SETUP_TOKEN:
            // Setup token received from host

            ProcessSetupPacket(eventData->setupEventData.pSetupPktBuffer);
            xferHandle = DRV_USB_Device_TransferRequest(myUSBDevice.ep0Pipe,
                                                       USB_XFER_CONTROL_READ, deviceDescriptor,18,
                                                       USB_XFER_REGULAR, myUSBDevice,
                                                       USBDeviceLayerXferCallback);

            if(DRV_HANDLE_INVALID == xferHandle)
            {
                /* If the transfer request failed, the
                 * client can find out why. */

                switch(DRV_USB_Device_PipeXferErrorGet(myUSBDevice.ep0Pipe))
                {
                    case USB_XFER_WRONG_PIPE:
                        /* This means that either the transfer type or
                         * does not match the pipe type or direction */
                        break;
                    case USB_XFER_QUEUE_FULL:
                        /* This means that the transfer queue on this
                         * pipe is full. Either increase the queue size
                         * by increasing the value of DRV_USB_XFERS_NUMBER
                         * or wait for a slot on the queue to be available. */
                        break;
                    default:
                        break;
                }
            }
    }
}

```

In a case where the data to be transferred is more than the endpoint size reported to the host (also specified at the time when the control endpoint pipe is setup), the host will break up the data stage of the transfer into multiple transactions. This process is transparent to the client. If at any point a SETUP packet is received while the transfer was in progress, the transfer is aborted and client receives USB_DEVICE_SETUP_TOKEN event. If a transfer complete callback function was registered in the transfer request function, this callback function is called with transfer status as USB_XFER_ABORT. Figure 5 shows the generation of different events during the Control Read Transaction

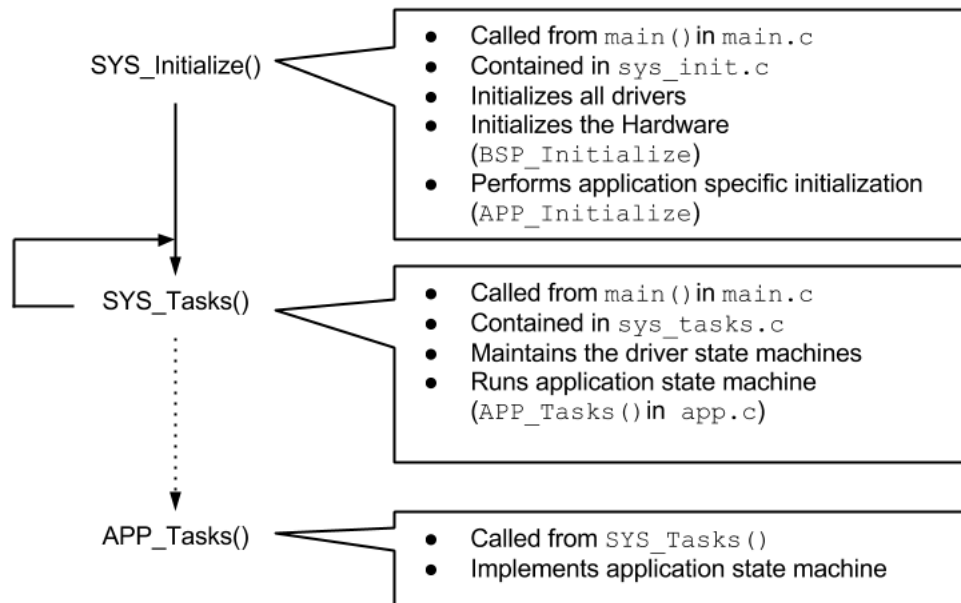


Figure 5: Control Read Transfer. Data size is 18 bytes. Maximum Endpoint size is 64 bytes (> 18 bytes).

A Control read transfer must have a data stage. In cases of control read transfers where the amount of data to be transferred is less than that requested by the host, the `USB_XFER_SEND_ZLP` flag must be specified. The driver will then check if the transfer size is an exact multiple of the endpoint size and then **send** a data stage zero length packet (ZLP). This indicates to the host that the device does not have any more data. There are two cases where specifying the `USB_XFER_SEND_ZLP` flag does not cause the driver to **send** a ZLP.

- When the size of the transfer is less than requested size but is not an exact multiple of the maximum endpoint size.
- The size of the transfer is less than maximum endpoint size.

The following code snippet shows an example of a transfer request which sends a ZLP to the host.

```

/* This is an example of a Control Read Transfer with the ZLP
 * option. Endpoint 0 maximum size is 8 bytes. The host has
 * requested 32 bytes but the device has only 24 bytes to send.
 * Specifying the USB_XFER_SEND_ZLP flag will cause the driver
 * to send a ZLP to the host in the fourth data transaction.
 */

DRV_USB_XFER_HANDLE xferHandle;

xferHandle = DRV_USB_Device_TransferRequest(myUSBDevice.ep0Pipe, USB_XFER_CONTROL_READ,
                                           deviceDescriptor, 24, USB_XFER_SEND_ZLP,
                                           myUSBDevice, USBDeviceLayerXferCallback);
  
```

Figure 6 shows the generation of different events and the additional ZLP sent in data stage in a multi transaction Control Read Transfer.

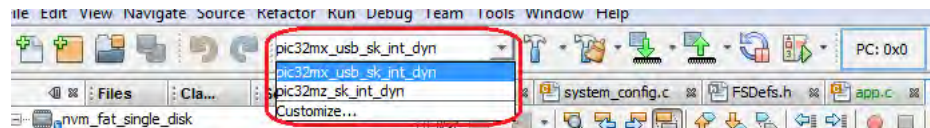


Figure 6: Multi Transaction Control Read Transfer with ZLP. Data size is 24 bytes and maximum endpoint size is 8 bytes.

2. Requesting a Control Write Transfer:

The code snippet below shows an example of how a Control Write Transfer can be requested. In this transfer, data moves from the host to device. The code example below shows an example of how a control write transfer is requested. Figure 7 shows the generation of different events related to the control write transfer.

```

/* This is an example of a Control Write Transfer. A control read
 * or Write transfer should be requested in response to a
 * USB_DEVICE_SETUP_TOKEN event. The transfer request function is
 * called in the USB event handler. In this example, the amount of
 * data to be transferred is 32 bytes. The transfer is requested
 * over endpoint 0 pipe. The USBDeviceLayerXferCallback() function
 * will be called when the transfer is terminated. The myUSBDevice
 * object will be passed back to the USBDeviceLayerXferCallback()
 * function. The Control Write Transfer only supports the
 * USB_XFER_REGULAR flag. */

DRV_USB_XFER_HANDLE xferHandle;

void USBDeviceLayerXferCallback( void * referenceData,
                                DRV_USB_PIPE_HANDLE hPipe,
                                DRV_USB_XFER_HANDLE hTransfer,
                                unsigned short int transferByteCount ,
                                DRV_USB_DEVICE_XFER_STATUS statusTransfer )
{
    /* This function will be called when the transfer is complete.
     * In this case, the referenceData argument will point to the
     * myUSBDevice data object that was specified in the Transfer
     * request function. */
}

void USBEventHandler (void * referenceHandle,
                     DRV_USB_EVENT eventType, DRV_USB_EVENT_DATA * eventData )
{
    switch ( eventType )
    {
        /* There could be other events here as well. Only
         * the USB_DEVICE_SETUP_TOKEN is shown in this example
         * for the sake of clarity */

        case USB_DEVICE_SETUP_TOKEN:
            // Setup token received from host

            ProcessSetupPacket(eventData->setupEventData.pSetupPktBuffer);
            xferHandle = DRV_USB_Device_TransferRequest(myUSBDevice.ep0Pipe,
                                                         USB_XFER_CONTROL_WRITE, buffer,32,
                                                         USB_XFER_REGULAR, myUSBDevice,
                                                         USBDeviceLayerXferCallback);

            if(DRV_HANDLE_INVALID == xferHandle)
            {
                /* The DRV_USB_Device_PipeXferErrorGet() can be used to
                 * find the error cause */
            }

        }
    }
}

```

In addition to the USB_DEVICE_SETUP_TOKEN event, the driver generates the USB_DEVICE_SETUP_HANDSHAKE event when the Control Write Handshake stage is complete. This event is generated even when the Control Write transfer does not have a data stage.

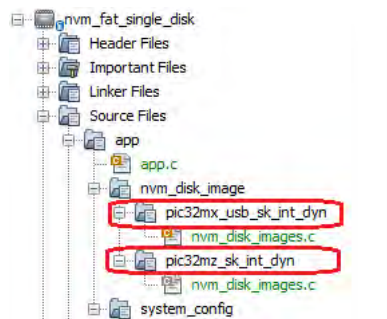


Figure 7: Example of Control Write Transfer. Data size is 18 bytes and maximum endpoint size is 64 bytes (> 18 bytes).

3. Control Write Transfers with Zero Data Stage.

A control write transfer can have zero data stages. One example of this is the SET ADDRESS command that the USB host **send** to the device during enumeration. In cases where no data stages are needed, the client should not request any transfer and return from the USB event handler. This will cause the driver to move the control transfer to the handshake stage. The USB_DEVICE_SETUP_HANDSHAKE can be used to track when the handshake stage of the transfer completes. In case of the SET ADDRESS command, the device layer firmware can change the device USB address in response to this event. The code example below shows an example of how this is done.

```
/* This is an example of zero data stage Control Write transfer.
 * This is done by not issuing a transfer request after the
 * USB_DEVICE_SETUP_TOKEN event is received. */

DRV_USB_XFER_HANDLE xferHandle;

void USBEventHandler (void * referenceHandle,
                     DRV_USB_EVENT eventType, DRV_USB_EVENT_DATA * eventData )
{
    switch ( eventType )
    {
        /* There could be other events here as well. Only
         * the USB_DEVICE_SETUP_TOKEN is shown in this example
         * for the sake of clarity */

        case USB_DEVICE_SETUP_TOKEN:
            /* Setup token received from host. In this case we assume
             * that this is the SET ADDRESS command. This command does
             * not have a data stage. So the client does not do a
             * transfer request*/

            ProcessSetupPacket(eventData->setupEventData.pSetupPktBuffer);
            break;

        case USB_DEVICE_SETUP_HANDSHAKE:
            /* The last control write transfer request is complete.
             * In this example, the client can change the device USB address
             * now. */
            DRV_USB_Device_AddressSet(deviceLayer, address);
            break;

        default:
            break;
    }
}
```

Figure 8 shows the generation of different events related to the transfer.

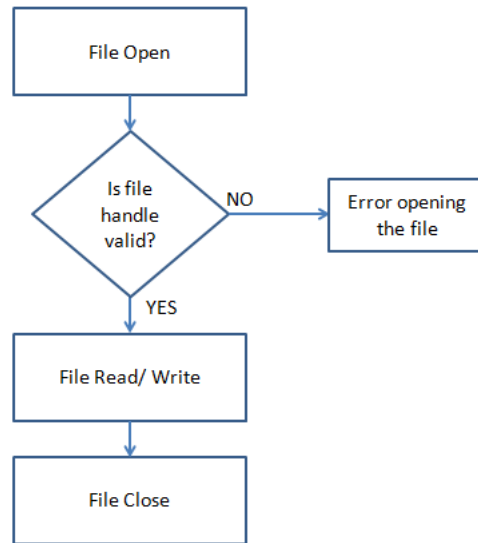


Figure 8: Example of Zero data stage Control Write Transfer.

4. Stalling the Pipe:

Refer to Pipe Stall section for more details on how a pipe can be stalled.

5.1.12.4.3.5.2 Bulk and Interrupt Transfers

The driver can process Bulk and Interrupt transfers. On the USB, both Bulk and Interrupt transfers have the same packet structure. The following discussion therefore applies to both Bulk and Interrupt transfers. A client should use the `DRV_USB_Device_TransferRequest()` function to schedule Bulk and Interrupt transfers. The following sections describe Bulk/Interrupt Read Transfers and Bulk/Interrupt Write Transfers.

1. Bulk/Interrupt Read Transfers:

In this transfer, data moves from device to host. The client can request a bulk/interrupt read transfer with a byte size greater than the maximum endpoint size. In this case the driver will complete the transfer and invoke the transfer complete callback function if provided. If the `USB_XFER_SEND_ZLP` flag is specified, the driver sends a ZLP after the last transaction if the transfer size was an exact multiple of the endpoint size.

The following code snippet shows an example of a Bulk/Interrupt read transfer can be requested

```

/* This code snippet shows an example of how a Bulk/Interrupt
 * read transfer can be requested */

DRV_USB_XFER_HANDLE transferHandle;

transferHandle = DRV_USB_Device_TransferRequest(pipe1, USB_XFER_BULK_READ,
    data, 120, USB_XFER_REGULAR, someReferenceData, XferCompleteCallback);

/* This transfer directs the driver to send a ZLP
 * after the last transaction */

transferHandle = DRV_USB_Device_TransferRequest(pipe1, USB_XFER_BULK_READ,
    data, 128, USB_XFER_SEND_ZLP, someReferenceData, XferCompleteCallback);
  
```

Figure 9 shows a regular bulk/interrupt read transfer. Figure 10 shows a bulk/interrupt read transfer with a ZLP.

Figure 9: Example of Bulk/Interrupt Read Transfer. Requested transfer size is 120 bytes. Maximum endpoint size 64

Figure 10: Example of Bulk/Interrupt Read Transfer with ZLP. Requested transfer size is 128 bytes. Maximum endpoint

size 64**2. Bulk/Interrupt Write Transfers:**

In this transfer, data moves from host to device. These transfers only support USB_XFER_REGULAR flag. The client can request a bulk/interrupt write transfer with a byte size greater than the maximum endpoint size. . In this case however, the driver will invoke the transfer complete callback function when a transaction completes. The callback thus gets invoked whenever the host sends any amount of data. Since the amount of data that the host sends in one transaction will not exceed the maximum endpoint size, specifying a transfer size greater than maximum endpoint size will not have any effect. The client must schedule another transfer to receive more data from the host. The following code snippet shows an example of bulk/interrupt write transfer.

```
/* This code snippet shows an example of how a Bulk/Interrupt
 * write transfer can be requested. Note the data size is
 * equal the size of the endpoint. The client has queued up
 * multiple transfers.*/

DRV_USB_XFER_HANDLE transferHandle1;
DRV_USB_XFER_HANDLE transferHandle1;

transferHandle1 = DRV_USB_Device_TransferRequest(pipe2, USB_XFER_BULK_WRITE,
    data,64,USB_XFER_REGULAR, someReferenceData ,XferCompleteCallback);
transferHandle1 = DRV_USB_Device_TransferRequest(pipe2, USB_XFER_BULK_WRITE,
    data,64,USB_XFER_REGULAR, someReferenceData ,XferCompleteCallback);
```

Figure 11 shows a bulk/interrupt write transfer with related events

Figure 11: Example of Bulk/Interrupt Write Transfer. Client has scheduled two transfers Maximum endpoint size 64

5.1.12.4.3.5.3 Isochronous Transfers

A client can request Isochronous transfers by using the DRV_USB_Device_TransferRequest() function. On the USB, Isochronous transfers are characterized by the absence of the handshake stage. The driver supports USB_XFER_REGULAR and USB_XFER_PERSIST flags for Isochronous transfers. While using the USB_XFER_REGULAR flag, the transfer request behavior is similar to Bulk and Interrupt Read/Write Transfer Requests. In that, a new transfer will require the client to call the DRV_USB_Device_TransferRequest() function.

By using the USB_XFER_PERSIST flag, the driver will not discard the transfer after the transfer is complete. Instead, after the transfer is complete, the transfer is rescheduled in the queue. This way the client does not have to schedule another transfer. If the client has scheduled two Isochronous transfers with the USB_XFER_PERSIST flag, the driver will ping pong between these transfers. The transfer complete callback function is called when the transfer is completed.

The code snippet below shows an example of setting up two Isochronous read transfers to create a ping pong type of transfer system.

```
DRV_USB_XFER_HANDLE pingTransferHandle;
DRV_USB_XFER_HANDLE pongTransferHandle;

/* Request two transfers. Note how the USB_XFER_PERSIST flag
 * is specified. Note also how the pingData and pongData buffer
 * pointers are specified as reference data. This will make
 * processing in the transfer complete call back easier.. */

pingTransferHandle = DRV_USB_Device_TransferRequest(pipe0, USB_XFER_CONTROL_READ,
    pingData,64,USB_XFER_PERSIST, pingData ,PingXferCompleteCallback);

pongTransferHandle = DRV_USB_Device_TransferRequest(pipe0, USB_XFER_CONTROL_READ,
    pongData,40,USB_XFER_PERSIST, pongData ,PongXferCompleteCallback);

/* This is the Ping transfer Complete callback */
void PingCompleteCallback( void * referenceData,
    DRV_USB_PIPE_HANDLE hPipe,
    DRV_USB_XFER_HANDLE hTransfer,
```

```

        unsigned short int transferByteCount ,
        DRV_USB_DEVICE_XFER_STATUS statusTransfer )
{
    /* Note that we dont schedule one more request here.
    * We just add data to the buffer and return. This
    * same transfer will be processed again. The referenceData
    * points to the ping data buffer (as set in the transfer request
    * function. */

    void * data = referenceData;
    UpdateBuffer(data);
}

/* This is the Ping transfer Complete callback */
void PongCompleteCallback( void * referenceData,
        DRV_USB_PIPE_HANDLE hPipe,
        DRV_USB_XFER_HANDLE hTransfer,
        unsigned short int transferByteCount ,
        DRV_USB_DEVICE_XFER_STATUS statusTransfer )
{
    /* Note that we dont schedule one more request here.
    * We just add data to the buffer and return. This
    * same transfer will be processed again. The referenceData
    * points to the pong data buffer (as set in the transfer request
    * function. */

    void * data = referenceData;
    UpdateBuffer(data);
}

```

5.1.12.4.3.5.4 Transfer Status

The driver offers to different interface to track the status of a transfer, a polling interface and a callback interface. Both these interfaces can be invoked by the application. The two interfaces are discussed here:

1. Polling interface:

The client can access the polling interface by using the DRV_USB_Device_XferStatusGet() function. This interface does not provide information when a transfer is aborted. The following code snippet shows how DRV_USB_Device_XferStatusGet() is used.

```

/* This code snippet shows the different
 * values that the DRV_USB_Device_XferStatusGet()
 * function. This function implements the polling
 * interface for the transfer status.
 */

DRV_USB_DEVICE_XFER_STATUS transferStatus;

transferStatus = DRV_USB_Device_XferStatusGet(transferHandle);

switch(transferStatus)
{
    case USB_XFER_PENDING:
        /* This means the transfer is still in the transfer
        * queue. This transfer can be aborted.*/
    case USB_XFER_IN_PROGRESS:
        /* This means the transfers is currently in progress
        * on the bus. This transfer cannot be aborted. */
    case USB_XFER_COMPLETED:
        /* This means the transfer is complete */
    default:
        break;
}

```

2. Callback interface:

The callback interface to track the status of the transfer is implemented by using Transfer Complete Callback function. The client should specify the Transfer Complete Callback function when calling DRV_USB_Device_TransferRequest() function. The driver call this function when a transfer has terminated, either due to the transfer completing or the transfer aborting. The Transfer Complete Callback function should be of the type DRV_USB_XFER_COMPLETE_CALLBACK. The driver returns the following information while calling the Transfer Complete Callback function.

1. Handle to the pipe that contained this transfer
2. Handle to the transfer
3. Number of bytes read or written
4. A reference to a data object that was specified in the DRV_USB_Device_TransferRequest() for this transfer.
5. Transfer termination status

The following code snippet shows an example of how a Transfer Complete Callback function can be used.

```
/* This code snippet shows an example of how to use the
 * transfer complete callback */

DRV_USB_XFER_HANDLE transferHandle;

/* Request a transfer. Note that the reference data pointer is set
 * to dataBuffer and the transfer complete callback function
 * XferCompleteCallback is specified. */

transferHandle = DRV_USB_Device_TransferRequest(pipe0, USB_XFER_CONTROL_READ,
        dataBuffer,18,false, dataBuffer ,XferCompleteCallback);

/* This is the transfer Complete callback */
void XferCompleteCallback( void * referenceData,
        DRV_USB_PIPE_HANDLE hPipe,
        DRV_USB_XFER_HANDLE hTransfer,
        unsigned short int transferByteCount ,
        DRV_USB_DEVICE_XFER_STATUS statusTransfer )
{
    /* The referenceData variable (the value of which
     * is specified in the transfer request function)
     * points to the data buffer associate with the
     * transfer. */

    void * dataBuffer = referenceData;

    if(transferStatus == USB_XFER_COMPLETED)
    {
        ProcessDataBuffer(dataBuffer, transferByteCount);
    }
    else if(transferStatus == USB_XFER_ABORTED)
    {
        /* This transfer was not successful. Take any
         * remedial action as needed. */
    }
}
```

5.1.12.4.3.5.5 Transfer Abort

A client can abort a USB transfer that it requested. This is achieved by using the DRV_USB_Device_TransferAbort() function. A Control read or write transfer can automatically get aborted when the host issues a SETUP token out of turn. The DRV_USB_Device_TransferAbort() will not abort a transfer if it is already in progress. Closing the pipe will however abort transfers that are in progress.

When a transfer is aborted, the Transfer Complete Callback function associated with the transfer is not invoked. If the Control Read or Write transfer was aborted due to a out of turn SETUP token from the host, Transfer Complete Callback function will be called and transfer status will be USB_XFER_ABORTED.

5.1.12.5 Configuring the Library

The configuration of the USB Driver is based on the file sys_config.h

This header file contains the configuration selection for the USB Driver. Based on the selections made, the USB Driver will support or not support selected features. Some configuration settings will apply to all instances of the USB Driver while some configurations settings are applicable only if the driver is built for a static mode.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.12.5.1 Initialization Overrides

This section lists out the macros that need to be configured when building the static version of the driver. These values should be defined for correction operation of the driver, when the driver is built statically. Commenting the DRV_USB_INSTANCES_NUMBER macro will cause the driver to be built in a static mode.

5.1.12.5.2 Others

These macros affect all instances and all build types of the driver.

5.1.12.6 Building the Library

The following table lists the USB Driver files to be included in the project for the available features

Features	File Path	Comments
Needed for all features	driver/usb/drv_usb.h	Must be included in every source code file that access the driver API.
Dynamic Build	driver/usb/src/dynamic/drv_usb.c	Must be included when using the dynamic version of the driver
Static Multi-Client Build	driver/usb/src/static_multi/drv_usb_static_multi.c	Must be included when using the static multi client version of the driver

5.1.12.7 Library Interface

Data Types and Constants

	Name	Description
	DRV_USB_ENDPOINT_TABLE_ENTRY_SIZE	
	DRV_USB_HOST_PIPE_HANDLE_INVALID	Defines the USB Driver Host Pipe Invalid handle
	DRV_USB_INDEX	This is macro DRV_USB_INDEX.
	DRV_USB_INDEX_0	USB driver index definitions
	DRV_USB_INTERRUPT_SOURCE	This is macro DRV_USB_INTERRUPT_SOURCE.

	DRV_USB_PERIPHERAL_ID	This is macro DRV_USB_PERIPHERAL_ID.
	DRV_USB_EVENT	Identifies the different events that the USB Driver provides.
	DRV_USB_EVENT_CALLBACK	Type of the USB event callback function
	DRV_USB_HOST_PIPE_HANDLE	Defines the USB Driver Host Pipe handle type
	DRV_USB_INIT	Contains all the data necessary to initialize the USB device

Client Functions

	Name	Description
	DRV_USB_Close	Closes an opened-instance of the USB driver
	DRV_USB_Open	Opens the specified USB driver instance and returns a handle to it

Other Functions

	Name	Description
	DRV_USB_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the a client of USB bus events.
	DRV_USB_DEVICE_AddressSet	This function will set the USB module address that is obtained from the host.
	DRV_USB_DEVICE_Attach	This function will attach the device to the USB.
	DRV_USB_DEVICE_CurrentSpeedGet	This function will return the USB speed at which device is operating.
	DRV_USB_DEVICE_Detach	This function will detach the device from the USB.
	DRV_USB_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USB_DEVICE_EndpointDisableAll	This function disables all provisioned non-zero endpoints.
	DRV_USB_DEVICE_EndpointEnable	This function enables a endpoint for the specified direction and endpoint size.
	DRV_USB_DEVICE_EndpointIsEnabled	This function returns the enable/ disable status of the specified endpoint and direction.
	DRV_USB_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USB_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	DRV_USB_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USB_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USB_DEVICE_IRPSubmit	This function submits a I/O Request Packet (IRP) for processing to the USB Driver.
	DRV_USB_HOST_BusResetControl	This is function DRV_USB_HOST_BusResetControl.
	DRV_USB_HOST_DeviceCurrentSpeedGet	This is function DRV_USB_HOST_DeviceCurrentSpeedGet.
	DRV_USB_HOST_IRPCancel	This is function DRV_USB_HOST_IRPCancel.
	DRV_USB_HOST_IRPSubmit	This is function DRV_USB_HOST_IRPSubmit.
	DRV_USB_HOST_OperationEnable	This is function DRV_USB_HOST_OperationEnable.
	DRV_USB_HOST_StartOfFrameControl	This is function DRV_USB_HOST_StartOfFrameControl.
	DRV_USB_HOST_TimerIsComplete	This is function DRV_USB_HOST_TimerIsComplete.
	DRV_USB_HOST_TimerReset	This is function DRV_USB_HOST_TimerReset.
	DRV_USB_HOST_TimerStart	This is function DRV_USB_HOST_TimerStart.
	DRV_USB_Tasks	Maintains the driver's state machine when the driver is configured for polled mode.

Pipe Functions

	Name	Description
	DRV_USB_HOST_PipeSetup	This is function DRV_USB_HOST_PipeSetup.
	DRV_USB_HOST_PipeClose	This is function DRV_USB_HOST_PipeClose.

System Functions

	Name	Description
	DRV_USB_Initialize	Initializes the USB driver
	DRV_USB_Status	Provides the current status of the USB driver module
	DRV_USB_Tasks_ISR	Maintains the driver's state machine and implements its ISR
	DRV_USB_HOST_OperationIsEnabled	This is function DRV_USB_HOST_OperationIsEnabled.

Description

This section describes the APIs of the USB Driver. Refer to each section for a detailed description.

5.1.12.7.1 System Functions

5.1.12.7.1.1 DRV_USB_Initialize Function

C

```
SYS_MODULE_OBJ DRV_USB_Initialize(
    const SYS_MODULE_INDEX drvIndex,
    const SYS_MODULE_INIT * const init
);
```

Description

This routine initializes the USB driver, making it ready for clients to open and use it.

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Ordinal number of driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver. This should be a DRV_USB_INIT structure reference type casted to SYS_MODULE_INIT reference.

Returns

If successful, returns a valid handle to a driver object. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#). The returned object must be passed as argument to [DRV_USB_Status](#) routines.

Remarks

This routine must be called before any other USB routine is called. This routine should only be called once during system initialization unless DRV_USB_Deinitialize is called to de-initialize the driver instance. If the driver is built for static mode, then the initialization data structure is ignored. The static overrides provided in system_config.h are considered instead.

Example

```
// The following code snippet shows an example initialization of the
// driver. The USB module to be used is USB1. The module should not
// automatically suspend when the microcontroller enters Sleep mode.
```



```

// The module should continue operation when the module enters Idle
// mode. The power state is set to run at full clock speeds. Device
// Mode operation should be at FULL speed.

DRV_USB_INIT moduleInit;

usbInitData.usbID           = USB_ID_1;
usbInitData.opMode          = DRV_USB_OPERATION_MODE_DEVICE;
usbInitData.pingpongMode    = USB_PING_PONG_FULL_PING_PONG;
usbInitData.stopInIdle      = false;
usbInitData.suspendInSleep  = false;
usbInitData.operationSpeed  = USB_SPEED_FULL;
usbInitData.interruptSource = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

DRV_USB_Initialize(DRV_USB_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);

```

5.1.12.7.1.2 DRV_USB_Status Function

C

```

SYS_STATUS DRV_USB_Status(
    SYS_MODULE_OBJ object
);

```

Description

This routine provides the current status of the USB driver module.

Preconditions

Function [DRV_USB_Initialize](#) must have been called before calling this

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USB_Initialize routine

Returns

- SYS_STATUS_READY - Indicates that the driver is ready
- SYS_STATUS_UNINITIALIZED - Indicates that the driver has never been initialized

Remarks

None.

Example

```

SYS_MODULE_OBJ    object;    // Returned from DRV_USB_Initialize
SYS_STATUS         status;
DRV_USB_INIT      moduleInit;

usbInitData.usbID           = USB_ID_1;
usbInitData.opMode          = DRV_USB_OPERATION_MODE_DEVICE;
usbInitData.stopInIdle      = false;
usbInitData.suspendInSleep  = false;
usbInitData.operationSpeed  = USB_SPEED_FULL;
usbInitData.interruptSource = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize

```

```
// function.  
object = DRV_USB_Initialize(DRV_USB_INDEX_0,  
                           (SYS_MODULE_INIT *) &usbInitData);  
  
status = DRV_USB_Status(object);  
if(SYS_STATUS_READY == status)  
{  
    // Driver is ready to be opened.  
}
```

5.1.12.7.1.3 DRV_USB_Tasks_ISR Function

C

```
void DRV_USB_Tasks_ISR(  
    SYS_MODULE_OBJ object  
);
```

Description

This routine is used to maintain the driver's internal state machine and implement its ISR for interrupt-driven implementations.

Preconditions

The [DRV_USB_Initialize](#) routine must have been called for the specified USB driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USB_Initialize)

Returns

None

Remarks

This routine should be called from the USB interrupt service routine. In case of multiple USB modules, it should be ensured that the correct USB driver system module object is passed to the this routine.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_USB_Initialize  
  
while (true)  
{  
    DRV_USB_Tasks_ISR (object);  
  
    // Do other tasks  
}
```

5.1.12.7.1.4 DRV_USB_HOST_OperationIsEnabled Function

C

```
bool DRV_USB_HOST_OperationIsEnabled(  
    DRV_HANDLE hClient  
);
```

Description

This is function DRV_USB_HOST_OperationIsEnabled.

5.1.12.7.2 Client Functions

5.1.12.7.2.1 DRV_USB_Close Function

C

```
void DRV_USB_Close(  
    DRV_HANDLE handle  
)  
;
```

Description

This routine closes an opened-instance of the USB driver, invalidating the handle.

Preconditions

The [DRV_USB_Initialize](#) routine must have been called for the specified USB driver instance. [DRV_USB_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)

Returns

None

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling [DRV_USB_Open](#) before the caller may use the driver again.

Example

```
DRV_HANDLE handle; // Returned from DRV_USB_Open  
  
DRV_USB_Close(handle);
```

5.1.12.7.2.2 DRV_USB_Open Function

C

```
DRV_HANDLE DRV_USB_Open(  
    const SYS_MODULE_INDEX drvIndex,  
    const DRV_IO_INTENT intent  
)  
;
```

Description

This routine opens the specified USB driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The intent flag should always be `DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NON_BLOCKING`. Anyother setting of the intent flag will return a invalid driver handle. A driver instance can only support one client. Trying to open a driver that has an existing client will result in an unsuccessful function call.

Preconditions

Function [DRV_USB_Initialize](#) must have been called before calling this function.

Parameters

Parameters	Description
drvIndex	Driver instance that client will use
intent	Should always be <code>DRV_IO_INTENT_EXCLUSIVE DRV_IO_INTENT_READWRITE DRV_IO_INTENT_NON_BLOCKING</code> .

Returns

If successful, valid driver handle for the client to use when interfacing with the driver. If an error occurs, the return value is [DRV_HANDLE_INVALID](#).

Remarks

The handle returned is valid until the [DRV_USB_Close](#) routine is called.

Example

```
// Shows an example of how to open the driver.
SYS_MODULE_OBJ    object;    // Returned from DRV_USB_Initialize
SYS_STATUS         status;
DRV_USB_INIT      moduleInit;
DRV_HANDLE        handle;

usbInitData.usbID          = USB_ID_1;
usbInitData.opMode         = DRV_USB_OPERATION_MODE_DEVICE;
usbInitData.stopInIdle    = false;
usbInitData.operationSpeed = USB_SPEED_FULL;
usbInitData.suspendInSleep = false;
usbInitData.interruptSource = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

object = DRV_USB_Initialize(DRV_USB_INDEX_0,
                           (SYS_MODULE_INIT *) &usbInitData);

status = DRV_USB_Status(object);
if(SYS_STATUS_READY == status)
{
    // Driver is ready to be opened.
    handle = DRV_USB_Open(DRV_USB_INDEX_0,
                          DRV_IO_INTENT_EXCLUSIVE|
                          DRV_IO_INTENT_READWRITE|
                          DRV_IO_INTENT_NON_BLOCKING);
    SYS_ASSERT(DRV_HANDLE_INVALID != handle, "Bad handle returned");
}
```

5.1.12.7.3 Pipe Functions

5.1.12.7.3.1 DRV_USB_HOST_PipeSetup Function

C

```
DRV_USB_HOST_PIPE_HANDLE DRV_USB_HOST_PipeSetup(
    DRV_HANDLE client,
    uint8_t deviceAddress,
    USB_ENDPOINT endpointAndDirection,
    uint8_t hubAddress,
    uint8_t hubPort,
    USB_TRANSFER_TYPE pipeType,
    uint8_t bInterval,
    uint16_t wMaxPacketSize,
    USB_SPEED speed
);
```

Description

This is function DRV_USB_HOST_PipeSetup.

5.1.12.7.3.2 DRV_USB_HOST_PipeClose Function

C

```
void DRV_USB_HOST_PipeClose(  
    DRV_USB_HOST_PIPE_HANDLE pipeHandle  
)  
;
```

Description

This is function DRV_USB_HOST_PipeClose.

5.1.12.7.4 Transfer Functions

5.1.12.7.5 Other Functions

5.1.12.7.5.1 DRV_USB_ClientEventCallbackSet Function

C

```
void DRV_USB_ClientEventCallbackSet(  
    DRV_HANDLE handle,  
    uintptr_t hReferenceData,  
    DRV_USB_EVENT_CALLBACK myEventCallback  
)  
;
```

Description

This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events. The call back is disabled by not calling this function after a USB_DRV_Open or by passing NULL for the myEventCallback argument. When the callback function is called, the hReferenceData argument is returned.

Preconditions

None.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
hReferenceData	object (could be a pointer) that is returned with the callback.
myEventCallback	Call back function for all USB events.

Returns

None.

Remarks

Typical usage of the USB Driver requires a client to register a callback.

Example

```
// Set the client event callback for the Device Layer.  
// The USBDeviceLayerEventHandler() function is the event  
// handler. When this event handler is invoked by the  
// driver, the driver returns back the second argument  
// specified in the below function (which in this case  
// is the device layer data structure). This allows the  
// application firmware to identify, as an example, the  
// device layer object associated with this callback  
//
```

```
DRV_USB_ClientEventCallbackSet(myUSBDevice.usbDriverHandle,  
                               (uintptr_t)&myUSBDevice,  
                               USBDeviceLayerEventHandler);
```

5.1.12.7.5.2 DRV_USB_DEVICE_AddressSet Function

C

```
void DRV_USB_DEVICE_AddressSet(  
    DRV_HANDLE handle,  
    uint8_t address  
);
```

Description

This function will set the USB module address that is obtained from the host in a setup transaction. The address is obtained from the SET_ADDRESS command issued by the host. The primary (first) client of the driver uses this function to set the module's USB address after decoding the setup transaction from the host.

Preconditions

None.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
address	The address of this module on the USB bus

Returns

None.

Remarks

None.

Example

```
// This function should be called by the first client of the driver,  
// which is typically the device layer. The address to set is obtained  
// from the host during enumeration.
```

```
DRV_USB_DEVICE_AddressSet(deviceLayer, 4);
```

5.1.12.7.5.3 DRV_USB_DEVICE_Attach Function

C

```
void DRV_USB_DEVICE_Attach(  
    DRV_HANDLE handle  
);
```

Description

This function will attach the device to the USB. It does this by enabling the pull up resistors on the D+ or D- lines. This function should be called when the driver client is ready to receive communication from the host (typically after all initialization is complete).

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)

Returns

None.

Remarks

None.

Example

```
// Open the device driver and attach the device to the USB.
SYS_MODULE_OBJ    object;    // Returned from DRV_USB_Initialize
SYS_STATUS         status;
DRV_USB_INIT      moduleInit;
DRV_HANDLE         handle;

usbInitData.usbID          = USB_ID_1;
usbInitData.opMode         = DRV_USB_OPERATION_MODE_DEVICE;
usbInitData.stopInIdle     = false;
usbInitData.suspendInSleep = false;
usbInitData.operationSpeed = USB_SPEED_FULL;
usbInitData.interruptSource = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

object = DRV_USB_Initialize(DRV_USB_INDEX_0,
                           (SYS_MODULE_INIT *) &usbInitData);

status = DRV_USB_Status(object);
if(SYS_STATUS_READY == status)
{
    // Driver is ready to be opened.
    handle = DRV_USB_Open(DRV_USB_INDEX_0,
                          DRV_IO_INTENT_EXCLUSIVE|
                          DRV_IO_INTENT_READWRITE|
                          DRV_IO_INTENT_NON_BLOCKING);
    SYS_ASSERT(DRV_HANDLE_INVALID != handle, "Bad handle returned");

    // Register a callback
    DRV_USB_ClientEventCallbackSet(handle, (uintptr_t)&myDeviceLayer,
                                    MyDeviceLayerEventCallback);

    // Perform any other initialization here and then attach the device
    // to the bus.
    DRV_USB_DEVICE_Attach(handle);
}
```

5.1.12.7.5.4 DRV_USB_DEVICE_CurrentSpeedGet Function**C**

```
USB_SPEED DRV_USB_DEVICE_CurrentSpeedGet(
    DRV_HANDLE handle
);
```

Description

This function will return the USB speed at which device is operating.

Preconditions

Only valid after device is attached to host.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)

Returns

Returns a member of the USB_SPEED type.

Remarks

None.

Example

```
// Get the current speed.
USB_SPEED deviceSpeed;

deviceSpeed = DRV_USB_DEVICE_CurrentSpeedGet(deviceLayer);

if(deviceLayer == USB_SPEED_HIGH)
{
    // Possibly adjust buffers for higher throughput.
}
```

5.1.12.7.5.5 DRV_USB_DEVICE_Detach Function**C**

```
void DRV_USB_DEVICE_Detach(
    DRV_HANDLE handle
);
```

Description

This function will detach the device from the USB. It does this by disabling the pull up resistors on the D+ or D- lines. This function should be called when the application wants to disconnect the device from the bus (typically to implement a soft detach or switch to host mode operation).

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)

Returns

None.

Remarks

None.

Example

```
// Detach the device from the USB
DRV_USB_DEVICE_Detach(handle);
```


5.1.12.7.5.6 DRV_USB_DEVICE_EndpointDisable Function

C

```
USB_ERROR DRV_USB_DEVICE_EndpointDisable(  
    DRV_HANDLE handle,  
    USB_ENDPOINT endpointAndDirection  
);
```

Description

This function disables an endpoint. If the endpoint type is control type then both directions are disabled. For non-control endpoints, the function disables one direction at a time.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
endpointAndDirection	Specifies the endpoint and direction.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

None.

Example

```
// This code snippet shows an example of how to disable  
// a control endpoint. Note that the direction parameter is ignored.  
// For a control endpoint, both the directions are disabled.  
  
USB_ENDPOINT ep;  
  
ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);  
  
DRV_USB_DEVICE_EndpointDisable(handle, ep );  
  
// This code snippet shows an example of how to disable a BULK IN  
// endpoint  
  
USB_ENDPOINT ep;  
  
ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);  
  
DRV_USB_DEVICE_EndpointDisable(handle, ep );
```

5.1.12.7.5.7 DRV_USB_DEVICE_EndpointDisableAll Function

C

```
USB_ERROR DRV_USB_DEVICE_EndpointDisableAll(  
    DRV_HANDLE handle  
);
```

Description

This function disables all provisioned endpoints in both directions.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)

Returns

None.

Remarks

This function is typically called by the USB Device Layer to disable all endpoints on detecting bus reset.

Example

```
// This code snippet shows an example of how to disable  
// all endpoints.
```

```
DRV_USB_DEVICE_EndpointDisableAll(handle);
```

5.1.12.7.5.8 DRV_USB_DEVICE_EndpointEnable Function**C**

```
USB_ERROR DRV_USB_DEVICE_EndpointEnable(  
    DRV_HANDLE handle,  
    USB_ENDPOINT endpointAndDirection,  
    USB_TRANSFER_TYPE transferType,  
    uint16_t endpointSize  
);
```

Description

This function enables a endpoint for the specified direction and endpoint size. The function will enable the endpoint for communication in one direction at a time. It must be called twice if the endpoint is required to communicate in both the directions, except for control endpoints. If the endpoint type is a control endpoint, the endpoint is always bi-directional and the function needs to be called only once. The size of the endpoint must match the wMaxPacketSize reported in the endpoint descriptor for this endpoint. The function does not check if the endpoint is already in use. It is the client's responsibility to make sure that a endpoint is not accidentally re-used.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
endpointAndDirection	Specifies the endpoint and direction.
transferType	Should be USB_TRANSFER_TYPE_CONTROL for control endpoint, USB_TRANSFER_TYPE_BULK for bulk endpoint, USB_TRANSFER_TYPE_INTERRUPT for interrupt endpoint and USB_TRANSFER_TYPE_ISOCHRONOUS for isochronous endpoint.
endpointSize	Maximum size (in bytes) of the endpoint as reported in the endpoint descriptor.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

None.

Example

```
// This code snippet shows an example of how to enable endpoint  
// 0 for control transfers. Note that for a control endpoint, the  
// direction parameter is ignored. A control endpoint is always  
// bi-directional. Endpoint size is 64 bytes.  
  
uint8_t ep;  
  
ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);  
  
DRV_USB_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_CONTROL, 64);  
  
// This code snippet shows an example of how to set up a endpoint  
// for BULK IN transfer. For an IN transfer, data moves from device  
// to host. In this example, endpoint 1 is enabled. The maximum  
// packet size is 64.  
  
uint8_t ep;  
  
ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);  
  
DRV_USB_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);  
  
// If endpoint 1 must also be set up for BULK OUT, then the enable  
// function must be called again, as shown here.  
  
ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);  
  
DRV_USB_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);
```

5.1.12.7.5.9 DRV_USB_DEVICE_EndpointIsEnabled Function**C**

```
bool DRV_USB_DEVICE_EndpointIsEnabled(  
    DRV_HANDLE client,  
    USB_ENDPOINT endpointAndDirection  
);
```

Description

This function returns the enable/ disable status of the specified endpoint and direction.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
endpointAndDirection	Specifies the endpoint and direction.

Returns

Returns true if the endpoint is enabled, false otherwise.

Remarks

None.

Example

```
// This code snippet shows an example of how the
// DRV_USB_DEVICE_EndpointIsEnabled() function can be used to obtain the
// status of the endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(DRV_USB_ENDPOINT_STATE_DISABLED ==
    DRV_USB_DEVICE_EndpointIsEnabled(handle, ep))
{
    // Endpoint is disabled. Enable endpoint.

    DRV_USB_DEVICE_EndpointEnable(handle, ep, USB_ENDPOINT_TYPE_BULK, 64);
}
```

5.1.12.7.5.10 DRV_USB_DEVICE_EndpointIsStalled Function**C**

```
bool DRV_USB_DEVICE_EndpointIsStalled(
    DRV_HANDLE client,
    USB_ENDPOINT endpoint
);
```

Description

This function returns the stall status of the specified endpoint and direction.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
endpointAndDirection	Specifies the endpoint and direction.

Returns

Returns true if endpoint is stalled, false otherwise.

Remarks

None.

Example

```
// This code snippet shows an example of how the
// DRV_USB_DEVICE_EndpointIsStalled() function can be used to obtain the
// stall status of the endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(true == DRV_USB_DEVICE_EndpointIsStalled (handle, ep))
{
    // Endpoint stall is enabled. Clear the stall.

    DRV_USB_DEVICE_EndpointStallClear(handle, ep);
}
```

5.1.12.7.5.11 DRV_USB_DEVICE_EndpointStall Function

C

```
USB_ERROR DRV_USB_DEVICE_EndpointStall(  
    DRV_HANDLE handle,  
    USB_ENDPOINT endpointAndDirection  
);
```

Description

This function stalls an endpoint in the specified direction.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
endpointAndDirection	Specifies the endpoint and direction.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

None.

Example

```
// This code snippet shows an example of how to stall an endpoint. In  
// this case , endpoint 1 IN direction is stalled.  
  
USB_ENDPOINT ep;  
  
ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);  
  
DRV_USB_DEVICE_EndpointStall(handle, ep);
```

5.1.12.7.5.12 DRV_USB_DEVICE_EndpointStallClear Function

C

```
USB_ERROR DRV_USB_DEVICE_EndpointStallClear(  
    DRV_HANDLE handle,  
    USB_ENDPOINT endpointAndDirection  
);
```

Description

This function clears the stall on an endpoint in the specified direction.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
endpointAndDirection	Specifies the endpoint and direction.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

None.

Example

```
// This code snippet shows an example of how to clear a stall. In this  
// example, the stall on endpoint 1 IN direction is cleared.
```

```
USB_ENDPOINT ep;  
  
ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);  
  
DRV_USB_DEVICE_EndpointStallClear(handle, ep);
```

5.1.12.7.5.13 DRV_USB_DEVICE_IRPCancelAll Function**C**

```
USB_ERROR DRV_USB_DEVICE_IRPCancelAll(  
    DRV_HANDLE client,  
    USB_ENDPOINT endpointAndDirection  
);
```

Description

This function cancels all IRPs that are queued and in progress at the specified endpoint.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
endpointAndDirection	Specifies the endpoint and direction.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

None.

Example

```
// This code snippet shows an example of how to cancel all IRPs.
```

```
void MyIRPCallback(USB_DEVICE_IRP * irp)  
{  
    // Check if this is setup command  
  
    if(irp->status == USB_DEVICE_IRP_STATUS_SETUP)  
    {  
        if(IsSetupCommandSupported(irp->data) == false)  
        {  
            // This means that this setup command is not  
            // supported. Stall the endpoint.  
  
            DRV_USB_DEVICE_IRPCancelAll(handle, ep);  
        }  
    }  
}
```

```

        DRV_USB_DEVICE_EndpointStall(handle, ep);
    }
}

```

5.1.12.7.5.14 DRV_USB_DEVICE_IRPSubmit Function

C

```

USB_ERROR DRV_USB_DEVICE_IRPSubmit(
    DRV_HANDLE client,
    USB_ENDPOINT endpointAndDirection,
    USB_DEVICE_IRP * irp
);

```

Description

This function submits a I/O Request Packet (IRP) for processing to the USB Driver. The IRP allows a client to **send** and receive data from the USB Host. The data will sent or received through the specified endpoint. The direction of the data transfer is indicated by the direction flag in the endpointAndDirection structure. Submitting an IRP arms the endpoint to either **send** data to or receive data from the host. If an IRP is under progress on the endpoint, then the subsequent IRP submit operation will result in the IRPs getting queued. The contents of the IRP should not be changed till the IRP has been processed.

Particular attention should be paid to the size parameter of IRP. The following should be noted:

- The size parameter while sending data to the host can be less than, greater than, equal to or be an exact multiple of maximum packet size for the endpoint. The maximum packet size for the endpoint determines the number of transactions required to process the IRP.
- If the size parameter while sending data to the host is less than maximum packet size, the transfer will complete in one transaction.
- If the size parameter while sending data to the host is greater than maximum packet size, the IRP will be processed in multiple transactions.
- If the size parameter while sending data to the host is equal to or an exact multiple of the maximum packet size, the client can optionally ask the driver to **send** a Zero Length packet by specifying the USB_DEVICE_IRP_FLAG_DATA_COMPLETE flag as the flag parameter.
- The size parameter while receiving data from the host must be an exact multiple of the maximum packet size of the endpoint. If this is not the case, the driver will return a USB_ERROR_IRP_SIZE_INVALID result. If while processing the IRP, the driver receives less than maximum packet size or a ZLP from the host, the driver considers the IRP as processed. The size parameter at this point contains the actual amount of data received from the host. The IRP status is returned as USB_DEVICE_IRP_STATUS_COMPLETED_SHORT.
- If a ZLP needs to be sent to host, the IRP size should be specified as 0 and flag should be set as USB_DEVICE_IRP_FLAG_DATA_COMPLETE.
- If the IRP size is an exact multiple of endpoint size, the client can request the driver to not **send** a ZLP, by specifying the USB_DEVICE_IRP_FLAG_DATA_PENDING. This flag indicates that there is more data pending in this transfer.
- Specifying a size less than endpoint size along with the USB_DEVICE_IRP_FLAG_DATA_PENDING flag will cause the driver to return a USB_ERROR_IRP_SIZE_INVALID.
- If the size is greater than endpoint size, and the flag is specified as USB_DEVICE_IRP_FLAG_DATA_PENDING, the driver will **send** multiple of endpoint size data.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USB_Open)
endpointAndDirection	Specifies the endpoint and direction.
irp	Pointer to the USB_DEVICE_IRP structure.

Returns

USB_ERROR_NONE - if the IRP was submitted successful. USB_ERROR_IRP_SIZE_INVALID - if the size parameter of the IRP is not correct. USB_ERROR_IRP_QUEUE_FULL - if the driver IRP queue is full.

Remarks

None.

Example

```
// The following code snippet shows an example of how to schedule a
// IRP to send data from device to host. Assume that max packet size
// is 64 and endpoint is 1. In this example, the transfer is
// processed as 3 transactions of 64, 64 and 2 bytes.
```

```
USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);
```

```
irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (DRV_USB_DEVICE_IRPSubmit(handle, ep, irp)
    != USB_ERROR_NONE)
{
    // This means there was an error.
}
else
{
    // The status of the IRP can be checked.
    while(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Wait or run a task routine.
    }
}
```

```
// The following code snippet shows how the client can request
// the driver to send a ZLP when the size is an exact multiple of
// end point size.
```

```
irp.data = myDataBufferToSend;
irp.size = 128;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;
```

```
// Note that while receiving data from the host, the size should
// be an exact multiple maximum packet size of the endpoint. In the
// example below, the DRV_USB_DEVICE_IRPSubmit() function will
// return a USB_DEVICE_IRP_SUBMIT_RESULT_INVALID_SIZE value.
```

```
ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);
```



```
irp.data = myDataBufferToSend;  
irp.size = 60; // THIS SIZE IS NOT CORRECT  
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;  
irp.callback = MyIRPCompletionCallback;  
irp.referenceData = (uintptr_t)&myDeviceLayerObj;
```

5.1.12.7.5.15 DRV_USB_HOST_BusResetControl Function

C

```
void DRV_USB_HOST_BusResetControl(  
    DRV_HANDLE client,  
    bool control  
);
```

Description

This is function DRV_USB_HOST_BusResetControl.

5.1.12.7.5.16 DRV_USB_HOST_DeviceCurrentSpeedGet Function

C

```
USB_SPEED DRV_USB_HOST_DeviceCurrentSpeedGet(  
    DRV_HANDLE client  
);
```

Description

This is function DRV_USB_HOST_DeviceCurrentSpeedGet.

5.1.12.7.5.17 DRV_USB_HOST_IRPCancel Function

C

```
void DRV_USB_HOST_IRPCancel(  
    USB_HOST_IRP * inputIRP  
);
```

Description

This is function DRV_USB_HOST_IRPCancel.

5.1.12.7.5.18 DRV_USB_HOST_IRPSubmit Function

C

```
USB_ERROR DRV_USB_HOST_IRPSubmit(  
    DRV_USB_HOST_PIPE_HANDLE hPipe,  
    USB_HOST_IRP * inputIRP  
);
```

Description

This is function DRV_USB_HOST_IRPSubmit.

5.1.12.7.5.19 DRV_USB_HOST_OperationEnable Function

C

```
void DRV_USB_HOST_OperationEnable(  
    DRV_HANDLE hClient,  
    bool enable  
);
```

Description

This is function DRV_USB_HOST_OperationEnable.

5.1.12.7.5.20 DRV_USB_HOST_StartOfFrameControl Function**C**

```
void DRV_USB_HOST_StartOfFrameControl(  
    DRV_HANDLE client,  
    bool control  
);
```

Description

This is function DRV_USB_HOST_StartOfFrameControl.

5.1.12.7.5.21 DRV_USB_HOST_TimerIsComplete Function**C**

```
bool DRV_USB_HOST_TimerIsComplete(  
    DRV_HANDLE client  
);
```

Description

This is function DRV_USB_HOST_TimerIsComplete.

5.1.12.7.5.22 DRV_USB_HOST_TimerReset Function**C**

```
void DRV_USB_HOST_TimerReset(  
    DRV_HANDLE client  
);
```

Description

This is function DRV_USB_HOST_TimerReset.

5.1.12.7.5.23 DRV_USB_HOST_TimerStart Function**C**

```
bool DRV_USB_HOST_TimerStart(  
    DRV_HANDLE client,  
    uint32_t periodInMilliseconds  
);
```

Description

This is function DRV_USB_HOST_TimerStart.

5.1.12.7.5.24 DRV_USB_Tasks Function**C**

```
void DRV_USB_Tasks(  
    SYS_MODULE_OBJ object  
);
```

Description

Maintains the driver's state machine when the driver is configured for polled mode. This function should be called from the

system tasks routine.

Preconditions

The [DRV_USB_Initialize](#) routine must have been called for the specified USB driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USB_Initialize)

Returns

None

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)). This function will never block.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_USB_Initialize

while (true)
{
    DRV_USB_Tasks(object);

    // Do other tasks
}
```

5.1.12.7.6 Data Types and Constants

5.1.12.7.6.1 DRV_USB_ENDPOINT_TABLE_ENTRY_SIZE Macro

C

```
#define DRV_USB_ENDPOINT_TABLE_ENTRY_SIZE
```

5.1.12.7.6.2 DRV_USB_HOST_PIPE_HANDLE_INVALID Macro

C

```
#define DRV_USB_HOST_PIPE_HANDLE_INVALID ((DRV_USB_HOST_PIPE_HANDLE)(-1))
```

Description

USB Driver Host Pipe Handle Invalid

Defines the USB Driver Host Pipe Invalid handle

Remarks

None.

5.1.12.7.6.3 DRV_USB_INDEX Macro

C

```
#define DRV_USB_INDEX 0
```

Description

This is macro DRV_USB_INDEX.

5.1.12.7.6.4 DRV_USB_INDEX_0 Macro

C

```
#define DRV_USB_INDEX_0 0
```

Description

USB Driver Module Index Numbers

These constants provide USB driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals and should be passed into the [DRV_USB_Initialize](#) and [DRV_USB_Open](#) routines to identify the driver instance in use. These are not indicative of the number of modules that are actually supported by the microcontroller.

5.1.12.7.6.5 DRV_USB_INTERRUPT_SOURCE Macro

C

```
#define DRV_USB_INTERRUPT_SOURCE INT_SOURCE_USB_1
```

Description

This is macro DRV_USB_INTERRUPT_SOURCE.

5.1.12.7.6.6 DRV_USB_PERIPHERAL_ID Macro

C

```
#define DRV_USB_PERIPHERAL_ID USB_ID_1
```

Description

This is macro DRV_USB_PERIPHERAL_ID.

5.1.12.7.6.7 DRV_USB_EVENT Enumeration

C

```
typedef enum {
    DRV_USB_EVENT_ERROR = 1,
    DRV_USB_EVENT_RESET_DETECT,
    DRV_USB_EVENT_RESUME_DETECT,
    DRV_USB_EVENT_IDLE_DETECT,
    DRV_USB_EVENT_STALL,
    DRV_USB_EVENT_SOF_DETECT,
    DRV_USB_EVENT_HOST_ATTACH,
    DRV_USB_EVENT_HOST_DETACH
} DRV_USB_EVENT;
```

Description

USB Driver Events Enumeration

Identifies the different events that the USB Driver provides.

Members

Members	Description
DRV_USB_EVENT_ERROR = 1	Bus error occurred and was reported
DRV_USB_EVENT_RESET_DETECT	Host has issued a device reset

DRV_USB_EVENT_RESUME_DETECT	Resume detected while USB in suspend mode
DRV_USB_EVENT_IDLE_DETECT	Idle detected
DRV_USB_EVENT_STALL	Stall handshake has occurred

5.1.12.7.6.8 DRV_USB_EVENT_CALLBACK Type

C

```
typedef void (* DRV_USB_EVENT_CALLBACK)(uintptr_t hClient, DRV_USB_EVENT eventType, void * eventData);
```

Description

Type of the USB Event Callback Function

Type of the USB event callback function. The client should register an event callback function of this type when it intends to receive events from the USB driver. The event callback function is registered using the [DRV_USB_ClientEventCallBackSet\(\)](#) function.

Parameters

Parameters	Description
hClient	handle to driver client that registered this callback function.
eventType	Event type
eventData	Event relevant data.

Returns

None.

Remarks

None.

5.1.12.7.6.9 DRV_USB_HOST_PIPE_HANDLE Type

C

```
typedef uintptr_t DRV_USB_HOST_PIPE_HANDLE;
```

Description

USB Driver Host Pipe Handle

Defines the USB Driver Host Pipe handle type

Remarks

None.

5.1.12.7.6.10 DRV_USB_INIT Structure

C

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    USB_MODULE_ID usbID;
    bool stopInIdle;
    bool suspendInSleep;
    INT_SOURCE interruptSource;
    USB_SPEED operationSpeed;
    USB_OPMODES operationMode;
    void * endpointTable;
} DRV_USB_INIT;
```

Description

USB Device Driver Initialization Data

This structure contains all the data necessary to initialize the USB driver.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
USB_MODULE_ID usbID;	Identifies peripheral (PLIB-level) ID
bool stopInIdle;	Boolean flag: true -> Stop USB module in Idle Mode
bool suspendInSleep;	Boolean flag: true -> Suspend USB in Sleep Mode
INT_SOURCE interruptSource;	Interrupt Source for USB module
USB_SPEED operationSpeed;	USB Operation speed
USB_OPMODES operationMode;	USB module operating mode
void * endpointTable;	Endpoint Descriptor Table buffer

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the [DRV_USB_Initialize](#) routine.

5.1.12.8 Files

Files

Name	Description
drv_usb.h	USB Device Driver Interface File
drv_usb_config_template.h	USB driver configuration definitions template

Description

5.1.12.8.1 drv_usb.h

USB Device Driver Interface

The USB device driver provides a simple interface to manage the "USB" peripheral. This file defines the interface definitions and prototypes for the USB driver.

Enumerations

	Name	Description
	DRV_USB_EVENT	Identifies the different events that the USB Driver provides.

Functions

	Name	Description
	DRV_USB_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the a client of USB bus events.
	DRV_USB_Close	Closes an opened-instance of the USB driver
	DRV_USB_DEVICE_AddressSet	This function will set the USB module address that is obtained from the host.

	DRV_USB_DEVICE_Attach	This function will attach the device to the USB.
	DRV_USB_DEVICE_CurrentSpeedGet	This function will return the USB speed at which device is operating.
	DRV_USB_DEVICE_Detach	This function will detach the device from the USB.
	DRV_USB_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USB_DEVICE_EndpointDisableAll	This function disables all provisioned non-zero endpoints.
	DRV_USB_DEVICE_EndpointEnable	This function enables a endpoint for the specified direction and endpoint size.
	DRV_USB_DEVICE_EndpointIsEnabled	This function returns the enable/ disable status of the specified endpoint and direction.
	DRV_USB_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USB_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	DRV_USB_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USB_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USB_DEVICE_IRPSubmit	This function submits a I/O Request Packet (IRP) for processing to the USB Driver.
	DRV_USB_HOST_BusResetControl	This is function DRV_USB_HOST_BusResetControl.
	DRV_USB_HOST_DeviceCurrentSpeedGet	This is function DRV_USB_HOST_DeviceCurrentSpeedGet.
	DRV_USB_HOST_IRPCancel	This is function DRV_USB_HOST_IRPCancel.
	DRV_USB_HOST_IRPSubmit	This is function DRV_USB_HOST_IRPSubmit.
	DRV_USB_HOST_OperationEnable	This is function DRV_USB_HOST_OperationEnable.
	DRV_USB_HOST_OperationIsEnabled	This is function DRV_USB_HOST_OperationIsEnabled.
	DRV_USB_HOST_PipeClose	This is function DRV_USB_HOST_PipeClose.
	DRV_USB_HOST_PipeSetup	This is function DRV_USB_HOST_PipeSetup.
	DRV_USB_HOST_StartOfFrameControl	This is function DRV_USB_HOST_StartOfFrameControl.
	DRV_USB_HOST_TimerIsComplete	This is function DRV_USB_HOST_TimerIsComplete.
	DRV_USB_HOST_TimerReset	This is function DRV_USB_HOST_TimerReset.
	DRV_USB_HOST_TimerStart	This is function DRV_USB_HOST_TimerStart.
	DRV_USB_Initialize	Initializes the USB driver
	DRV_USB_Open	Opens the specified USB driver instance and returns a handle to it
	DRV_USB_Status	Provides the current status of the USB driver module
	DRV_USB_Tasks	Maintains the driver's state machine when the driver is configured for polled mode.
	DRV_USB_Tasks_ISR	Maintains the driver's state machine and implements its ISR

Macros

	Name	Description
	DRV_USB_ENDPOINT_TABLE_ENTRY_SIZE	
	DRV_USB_HOST_PIPE_HANDLE_INVALID	Defines the USB Driver Host Pipe Invalid handle
	DRV_USB_INDEX_0	USB driver index definitions

Structures

	Name	Description
	DRV_USB_INIT	Contains all the data necessary to initialize the USB device

Types

	Name	Description
	DRV_USB_EVENT_CALLBACK	Type of the USB event callback function
	DRV_USB_HOST_PIPE_HANDLE	Defines the USB Driver Host Pipe handle type

Company

Microchip Technology Incorporated

FileName: drv_usb.h

5.1.12.8.2 drv_usb_config_template.h

USB Driver Configuration Definitions for the template version

These definitions statically define the driver's mode of operation.

Macros

	Name	Description
	DRV_USB_INDEX	This is macro DRV_USB_INDEX.
	DRV_USB_INTERRUPT_SOURCE	This is macro DRV_USB_INTERRUPT_SOURCE.
	DRV_USB_PERIPHERAL_ID	This is macro DRV_USB_PERIPHERAL_ID.

File Name

drv_usb_config_template.h

Company

Microchip Technology Incorporated

5.1.13 MRF24W Wi-Fi Driver Library

MRF24W Wi-Fi Driver Library

for

Microchip Microcontrollers

This library provides an interface to manage the Wi-Fi module on the MRF24W family of Microchip microcontrollers in different modes of operation.

Description

Content pending.

5.1.13.1 Introduction

MRF24W Wi-Fi Driver Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the MRF24W Wi-Fi Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.1.13.2 Release Notes

MPLAB Harmony Version: v0.70b **MRF24W Wi-Fi Driver Library Version :** 3.0.0 **Release Date:** 18Nov2013

New This Release:

Nothing to report in this release.

Known Issues:

- No W-iFi demonstrations are available for PIC32MZ
- Webserver demonstration:
 - Mpfupload of image requires system boot upon completion
- EasyConfig Demonstration (Explorer 16 Development Board and PIC32 Ethernet Starter Kit)
 - Mpfupload of image requires system boot upon completion
 - Ad-Hoc mode – cannot retrieve scan info/cannot redirect network
- Security (minor problem)
 - WEP Shared Key is not supported
- Stack
 - Heap memory leak
 - TCP network traffic (Max TX Packet error detect)
 - Heap allocation failure
- Services not verified
 - DNS Server
 - DNS Client
 - Dynamic DNS
 - Zero Conf
 - IPv6
 - SNMPv1, SNMP2, SNMP3
 - FTP
- Data throughput in open air (throughput is still lower than v5 stack)
 - IPerf TCP RX/TX (320/102 Kbps)
 - IPerf UDP RX/TX (570/468 Kbps)

5.1.13.2.1 v0.51.01b

Release notes for MRF24WG Wi-Fi Driver v0.51.01b

Description

Due to a special circumstance for this release the MRF24WG Wi-Fi driver will compile and work with legacy peripheral drivers (plib.h), and not with the MPLAB Harmony peripheral (peripheral.h). This MRF24WG Wi-Fi driver compiling issue will be fixed in a future release. Applications can still take advantage of the V6 stack features and services in this B1U dot release, along with Wi-Fi connectivity.

Known Issues:

- 1) Stack ASSERT condition while waiting for RAW_MOVE_TO_COMPLETE
- 2) Web Server Demo hang or irresponsive to HTTP query

Software Requirements:

- (1) MPLAB X IDE 1.80
- (2) XC32 compiler v1.21 and later

Applicable Product:

- (1) MRF24WG

Supported Application Demos:*(1) Web Server Demo*

- a. SNMPv3, multi-port connection (Ethernet + Wi-Fi)
- b. Network type supported: Infrastructure (Client STA)
- c. Security mode supported: Open , WEP 64/128 Open, WPS, WPA/WPA2 PSK TKIP/AES

Project Files:

To run with Explorer 16 Development Board (EX16)

Directory folder - ".\apps\tcpip\web_server\firmware\pic32_ex16.X"

To run with PIC32 IO Expansion Board with Ethernet Starter Kit (ESK)

Directory folder - ".\apps\tcpip\web_server\firmware\pic32_esk_mrf24w.X"

(2) Console Demo

- a. Network benchmark debug/test support such as iPerf, etc via system command line access.
- b. Network type supported: Infrastructure (Client STA), Ad-Hoc, Wi-Fi Direct

If testing Wifi-Direct, use RF module FW version 0x3107.

RF module FW version 0x3108 does NOT have support for Wi-Fi Direct.

Project File:

To run with Explorer 16 Development Board (EX16)

Directory

".\apps\tcpip\wifi_console\firmware\pic32_ex16_mrf24w.X"

(3) EasyConfig Demo

- a. Network provision allowing configurations of an embedded device on a wireless network. (SoftAP, Ad-Hoc)
- b. Network type supported: SoftAP, Ad-Hoc

Project File:

To run with Explorer 16 Development Board (EX16)

Directory

“.\apps\tcpip\wifi_easyconf\firmware\pic32_ex16.X”

Key Features:**Verified:**

- (1) SNMPv3
 - (2) DHCP Server/Client - server supports up to max 4 IPs
 - (3) SoftAP is able to support up to max 4 clients (STAs)
- Requires RF module FW version 0x3108.
- RF module FW version 0x3107 only supports 1 client for SoftAP.
- (4) Ad-Hoc network with support for up to max 4 STAs
 - (5) Multi-Ethernet ports (Ethernet and Wi-Fi)
 - (6) Serial console 115,200 baud

Supported:

- (1) ZeroConfig/Bonjour (with mDNS)
 - (2) SNMP v1, 2c, v3
 - (3) SMTP
 - (4) SSL (512, 1024, 2048-bit)
 - (5) HTTP2
 - (6) ICMP
 - (7) DNS
 - (8) NBNS
 - (9) Iperf
 - (10) TCP/UDP modules
 - (11) TCPIP Network Commands
 - (12) TCPIP Storage Service
 - (13) Announce
 - (14) ARP (Gratuitous)
 - (15) Berkeley API
 - (16) ZeroConfig/Bonjour (with mDNS)
 - (17) Power management / Power Save
- Active mode (Known as normal mode in v5 stack)

- Doze mode (Known as PS-Polling mode in v5 stack)

Supported But Not Verified:

- (1) Telnet
- (2) TFTP
- (3) SNTP
- (4) Reboot Server
- (5) IPV6 : This is currently not supported due to a defect of software multicast filter of MRF24WG 3107 or the older FW. The problem will be fixed with upcoming 3108 FW, and thereby the IPv6 will get supported for Wi-Fi.
- (6) FTP

To test ESK board:

- Ethernet Starter Kit (ESK) is valid for Web server demo
- ESK + MRF24WG
 - Please note that in order to use the MRF24WG module with the PIC32ESK you need to use an I/O Expansion board (DM320002) and insert the MRF24WG module in one of the PICTail Plus slots (SPI1 position is the default).
 - You need to [connect](#) the J10/PMPA4/56 and J10/INT1/37 pins on the I/O Expansion board so that the MRF24WG interrupt signal reaches the PIC32 processor. For this configuration the MRF24WG driver uses a CN (change notice) interrupt.

5.1.13.2.2 v0.51b

Release notes for MRF24WG Wi-Fi driver v0.51b.

Description

V6 MPLAB Harmony (v0.51b) Beta Release is limited release to demonstrate V6 stack capabilities, and was verified on one Development Board:

- (1) Explorer16 Development Board, with PIC32MX795F512L PIM, and MRF24WG PICTail (FW 0x3108 only)

There are two Application Demos included to demonstrate some Stack services:

- (1) Web Server Demo (Infrastructure mode)
- (2) EasyConfig Demo (SoftAP and Ad-Hoc modes)

a. Over-air system configuration

For development work and running the above Demos, will require:

- (1) MPLAB X IDE v1.85
- (2) XC32 compiler v1.21 or later
- (3) MPLAB ICD 3
- (4) Windows XP SP3 or 7 PC with serial port, USB port.
- (5) Development boards mentioned above.
- (6) And MLA Stack V6 Harmony (MPLAB Harmony v0_51b) Beta Release package.

Specific V6 Beta Stack features supported and demonstration included: (verified)

- (1) SoftAP with support for 4 STA Clients.
- (2) DHCP Server/Client. DHCP Server supports up to 4 clients.

(3) Ad-Hoc network with support for up to 4 STAs (joiners).

(4) Wi-Fi over-air system configuration

(5) MRF24G supporting 802.11b/g mixed

(6) 115,200 baud SPI interface

(7) MPLAB X IDE and X32C compiler

(8) ZeroConfig/Bonjour (with mDNS)

Other V6 Stack services: (included, verified)

(1) IPV4

(2) IPV6 (with Ping only)

(3) SMTP

(4) SSL (512, 1024, 2048-bit)

(5) HTTP

(6) ICMP

(7) NBNS

(8) IPerf

(9) ARP (Gratuitous)

(10) TCP/UDP modules

(11) TCPIP Network Commands

(12) TCPIP Storage Service

Other V6 Stack services: (included, NOT VERIFIED)

(1) SNMP v1, 2c

(2) SNMPv3

(3) SNTP

(4) DNS Client/Server

(5) Telnet

(6) FTP

(7) TFTP

(8) Announce

(9) AutoIP

(10) Berkeley API

(11) Reboot Server

Specific Stack/Wi-Fi Connection Agent supports:

(1) Wi-Fi Specific Features:

a. ENABLE_SOFTWARE_MULTICAST_FILTER

- b. RETRIEVE_BINARY_KEY (not support in Beta)
 - c. SAVE_WPS_CONFIDENTIALS (not support in Beta)
 - d. DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRUCTURE
 - e. DERIVE_KEY_FROM_PASSPHRASE_IN_HOST
 - f. MY_DEFAULT_MAC_ADDRESS
- (2) Network Type:
- a. 802.11 b/g mixed
 - b. 802.11 b-only
 - c. 802.11 legacy
- (3) Network Mode:
- a. Infrastructure (Open , WEP 64/128 Open/Shared Key), WPS, WPA/WPA2 PSK TKIP/AES)
 - b. Ad-Hoc (Open, WEP 64/128 Open/Shared Key)
 - c. SoftAP (Open, WEP 64/128 Open Key)
- (4) Power Save
- a. Normal
 - b. PS-Poll

MLA STACK V6 MPLAB Harmony (v0.51b) Beta Release PACKAGE

DEMO APPLICATION & MICROCHIP DEVELOPMENT BOARD

Single Port (Wi-Fi only) - WEB Server and EasyConfig Demos

Development Board Configuration:

Explorer16 Development Board

MRF24WG PICtail (FW 0x3108 only)

PIC32MX795F512L PIM

WEB Server Demo project:

"...\software\isp_root\apps\tcpip\tcpip_web_server_demo_app\firmware\pic32_ex16.X"

EasyConfig Demo project:

"...\software\isp_root\apps\tcpip\tcpip_wifi_easyconfig_app\firmware\pic32_ex16.X"

DEMO CONFIGURATION

Refer to WF_Config.h and TCPIP_Config.h files in respective MPLAB X IDE demo project for network configuration and setup information.

5.1.13.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its

licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.1.13.4 Using the Library

This topic describes the basic architecture of the MRF24W Wi-Fi Driver Library and provides information and examples on how to use it.

Interface Header File: [drv_wifi.h](#)

The interface to the ADC Driver Library is defined in the "[drv_wifi.h](#)" header file.

Please refer to the [MPLAB Harmony Overview](#) for how the driver interacts with the framework.

5.1.13.4.1 Abstraction Model

This library provides a low-level abstraction of the Wi-Fi module on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Content pending.

5.1.13.4.2 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each sub-section addresses one of the blocks or the overall operation of the Wi-Fi module.

Library Interface Section	Description
System Functions and System Configuration Functions	Provides system module interfaces, device initialization, deinitialization, re-initialization, tasks, and status functions.
Miscellaneous	Provides driver miscellaneous functions, version identification functions etc.

5.1.13.4.3 How the Library Works

This section describes how the Wi-Fi Driver Library operates.

Description

Before the driver is ready for use, its should be configured (compile time configuration). Refer to the [Configuring the Library](#) section for more details on how to configure the driver.

There are few run-time configuration items that are done during initialization of the driver instance, and a few that are client-specific and are done using dedicated functions.

To use the Wi-Fi Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation.

The following is the sequence in which various routines should be called:

Content pending.

5.1.13.4.3.1 System Initialization

This section describes initialization and reinitialization features.

Description

Content pending.

5.1.13.4.3.2 Client Functionality

This section describes core operation.

Description

Content pending.

5.1.13.4.3.3 Other Functionality

This section describes other functionality.

Description

Content pending.

5.1.13.5 Configuring the Library

The configuration of the MRF24W Wi-Fi driver is based on the file `drv_wifi_config.h`

This header file contains the configuration selection for the Wi-Fi Driver. Based on the selections made, the Wi-Fi Driver will support or not support selected features. These configuration settings will apply to all instances of the Wi-Fi Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.1.13.5.1 Initialization Overrides

[Drag and Drop Initialization overrides](#)

5.1.13.5.2 Files

5.1.13.5.3 Others

[Drag and Drop other configuration options](#)

5.1.13.5.4 Examples - Sample Functionality

[Drag the example configuration header file.](#)

5.1.13.6 Building the Library

This section list the files that are available in the \src of the MRF24W Wi-Fi driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.1.13.7 Library Interface

Data Types and Constants

	Name	Description
	DRV_WIFI_BSSID_LENGTH	This is macro DRV_WIFI_BSSID_LENGTH.
	DRV_WIFI_DEAUTH_REASONCODE_MASK	This is macro DRV_WIFI_DEAUTH_REASONCODE_MASK.
	DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD	ms
	DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID	Default values for WiFi AdHoc settings
	DRV_WIFI_DEFAULT_ADHOC_MODE	This is macro DRV_WIFI_DEFAULT_ADHOC_MODE.
	DRV_WIFI_DEFAULT_PS_DTIM_ENABLED	DTIM wake-up enabled (normally the case)
	DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL	number of beacon periods
	DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL	100ms multiplier, e.g. 1 * 100ms = 100ms
	DRV_WIFI_DEFAULT_SCAN_COUNT	Default values for WiFi scan context
	DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME	ms
	DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME	ms
	DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY	us
	DRV_WIFI_DEFAULT_WEP_KEY_TYPE	see DRV_WIFI_SecurityWepSet() and DRV_WIFI_WEP_CONTEXT
	DRV_WIFI_DISABLED	Do not make this an enumerated type!
	DRV_WIFI_DISASSOC_REASONCODE_MASK	This is macro DRV_WIFI_DISASSOC_REASONCODE_MASK.
	DRV_WIFI_ENABLED	This is macro DRV_WIFI_ENABLED.
	DRV_WIFI_MAX_CHANNEL_LIST_LENGTH	This is macro DRV_WIFI_MAX_CHANNEL_LIST_LENGTH.
	DRV_WIFI_MAX_NUM_RATES	This is macro DRV_WIFI_MAX_NUM_RATES.
	DRV_WIFI_MAX_SECURITY_KEY_LENGTH	This is macro DRV_WIFI_MAX_SECURITY_KEY_LENGTH.
	DRV_WIFI_MAX_SSID_LENGTH	This is macro DRV_WIFI_MAX_SSID_LENGTH.
	DRV_WIFI_MAX_WEP_KEY_LENGTH	This is macro DRV_WIFI_MAX_WEP_KEY_LENGTH.

	DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH	must include string terminator
	DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH	This is macro DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH.
	DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH	Key size defines
	DRV_WIFI_NETWORK_TYPE_ADHOC	This is macro DRV_WIFI_NETWORK_TYPE_ADHOC.
	DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE	Selection of different WiFi network types
	DRV_WIFI_NETWORK_TYPE_P2P	not supported
	DRV_WIFI_NETWORK_TYPE_SOFT_AP	This is macro DRV_WIFI_NETWORK_TYPE_SOFT_AP.
	DRV_WIFI_NO_ADDITIONAL_INFO	eventInfo define for DRV_WIFI_ProcessEvent() when no additional info is supplied
	DRV_WIFI_RETRY_ADHOC	This is macro DRV_WIFI_RETRY_ADHOC.
	DRV_WIFI_RETRY_FOREVER	This is macro DRV_WIFI_RETRY_FOREVER.
	DRV_WIFI_RTS_THRESHOLD_MAX	maximum RTS threshold size in bytes
	DRV_WIFI_SECURITY_EAP	This is macro DRV_WIFI_SECURITY_EAP.
	DRV_WIFI_SECURITY_OPEN	Selection of different WiFi security types
	DRV_WIFI_SECURITY_WEP_104	This is macro DRV_WIFI_SECURITY_WEP_104.
	DRV_WIFI_SECURITY_WEP_40	This is macro DRV_WIFI_SECURITY_WEP_40.
	DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY	This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY.
	DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE.
	DRV_WIFI_SECURITY_WPA_WITH_KEY	This is macro DRV_WIFI_SECURITY_WPA_WITH_KEY.
	DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE	This is macro DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE.
	DRV_WIFI_SECURITY_WPA2_WITH_KEY	This is macro DRV_WIFI_SECURITY_WPA2_WITH_KEY.
	DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE	This is macro DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE.
	DRV_WIFI_SECURITY_WPS_PIN	This is macro DRV_WIFI_SECURITY_WPS_PIN.
	DRV_WIFI_SECURITY_WPS_PUSH_BUTTON	This is macro DRV_WIFI_SECURITY_WPS_PUSH_BUTTON.
	DRV_WIFI_VERSION_NUMBER	WiFi Driver Version Number
	DRV_WIFI_WEP104_KEY_LENGTH	4 keys of 13 bytes each
	DRV_WIFI_WEP40_KEY_LENGTH	4 keys of 5 bytes each
	DRV_WIFI_WPA_KEY_LENGTH	This is macro DRV_WIFI_WPA_KEY_LENGTH.
	DRV_WIFI_WPA2_KEY_LENGTH	This is macro DRV_WIFI_WPA2_KEY_LENGTH.
	DRV_WIFI_WPS_PIN_LENGTH	7 digits + checksum byte
	DRV_WIFI_ADHOC_MODES	Selection of different AdHoc connection modes
	DRV_WIFI_ADHOC_NETWORK_CONTEXT	Contains data pertaining to WiFi AdHoc context
	DRV_WIFI_CONNECTION_CONTEXT	Contains data pertaining to MRF24WG connection context
	DRV_WIFI_CONNECTION_STATES	Wifi Connection states

	DRV_WIFI_DEVICE_INFO	Contains data pertaining to MRF24WG device type and version number
	DRV_WIFI_DEVICE_TYPES	Codes for WiFi device types
	DRV_WIFI_DOMAIN_CODES	Regional domain codes.
	DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES	Selection of different codes when WiFi connection is temporarily lost.
	DRV_WIFI_EVENT_INFO	Selection of different EventInfo types
	DRV_WIFI_EVENTS	Selections for events that can occur.
	DRV_WIFI_GENERAL_ERRORS	This is type DRV_WIFI_GENERAL_ERRORS.
	DRV_WIFI_HIBERNATE_STATE	This is type DRV_WIFI_HIBERNATE_STATE.
	DRV_WIFI_HIBERNATE_STATES	Wifi Hibernate states
	DRV_WIFI_MAC_STATS	Wifi MIB states
	DRV_WIFI_MGMT_ERRORS	Error codes returned when a mgmt message is sent to the MRF24W
	DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT	Contains data pertaining to WiFi Soft AP event
	DRV_WIFI_MULTICAST_FILTER_IDS	Selections for software Multicast filter ID's
	DRV_WIFI_MULTICAST_FILTERS	Selections for Software Multicast Filters.
	DRV_WIFI_POWER_SAVE_STATES	Wifi Power Save states
	DRV_WIFI_PS_POLL_CONTEXT	Contains data pertaining to WiFi PS-Poll context
	DRV_WIFI_REASON_CODES	Selection of different codes when a deauth or disassociation event has occurred.
	DRV_WIFI_RECONNECT_MODES	Selection of different Reconnection modes
	DRV_WIFI_SCAN_CONTEXT	Contains data pertaining to WiFi scan context
	DRV_WIFI_SCAN_RESULT	Contains data pertaining to WiFi scan results
	DRV_WIFI_SCAN_TYPES	Selection of different WiFi scan types
	DRV_WIFI_SECURITY_CONTEXT	Contains data pertaining to WiFi security.
	DRV_WIFI_SOFT_AP_EVENT_REASON_CODES	This is type DRV_WIFI_SOFT_AP_EVENT_REASON_CODES.
	DRV_WIFI_SOFT_AP_STATES	This is type DRV_WIFI_SOFT_AP_STATES.
	DRV_WIFI_STATUS_CODES	Selection of different codes when WiFi connection fails due to association or authentication failure.
	DRV_WIFI_SWMULTICAST_CONFIG	Contains data pertaining to WiFi software multicast filter configuration
	DRV_WIFI_TX_MODES	Selections for WiFi Tx mode
	DRV_WIFI_WEP_CONTEXT	Contains data pertaining to WiFi WEP context
	DRV_WIFI_WEP_KEY_TYPE	Selections for WEP key type when using WEP security.
	DRV_WIFI_WPA_CONTEXT	Contains data pertaining to WiFi WPA.
	DRV_WIFI_WPA_KEY_INFO	Contains data pertaining to WiFi WPA Key
	DRV_WIFI_WPS_AUTH_TYPES	This is type DRV_WIFI_WPS_AUTH_TYPES.
	DRV_WIFI_WPS_CONTEXT	Contains data pertaining to WiFi WPS security.
	DRV_WIFI_WPS_CREDENTIAL	Contains data pertaining to WiFi WPS Credentials
	DRV_WIFI_WPS_ENCODE_TYPES	This is type DRV_WIFI_WPS_ENCODE_TYPES.
	adhocMode	Selection of different AdHoc connection modes
	ENABLE_P2P_PRINTS	not supported
	ENABLE_WPS_PRINTS	This is macro ENABLE_WPS_PRINTS.
	PA6_IO	debug, remove later

	PA6_TRISTATE	This is macro PA6_TRISTATE.
	SET_PA6_AS_OUTPUT	This is macro SET_PA6_AS_OUTPUT.
	WF_WPS_PIN_LENGTH	WPS PIN Length

Miscellaneous Functions

	Name	Description
	DRV_WIFI_ConfigDataErase	Erases configuration data from the board EEPROM.
	DRV_WIFI_ConfigDataLoad	Loads configuration data from the board EEPROM.
	DRV_WIFI_ConfigDataPrint	Outputs to console the configuration data from the board EEPROM.
	DRV_WIFI_ConfigDataSave	Save configuration data to the board EEPROM.
	DRV_WIFI_ProcessEvent	Processes WiFi event
	DRV_WIFI_ScanGetResult	Read selected scan results back from MRF24W.
	DRV_WIFI_SetLinkDownThreshold	Sets number of consecutive WiFi Tx failures before link is considered down.
	DRV_WIFI_SetPSK	Sets the binary WPA PSK code in WPS.
	DRV_WIFI_SWMultiCastFilterEnable	Forces the MRF24WG to use software multicast filters instead of hardware multicast filters.
	DRV_WIFI_YieldPassphraseToHost	Allows host to get WPS WPA-PSK passphrase and convert to binary key.

System Configuration Functions

	Name	Description
	DRV_WIFI_Connect	Directs the MRF24WG to connect to a WiFi network.
	DRV_WIFI_Deinitialize	Initializes the MRF24WG WiFi driver.
	DRV_WIFI_Disconnect	Directs the MRF24WG to disconnect from a WiFi network.
	DRV_WIFI_Initialize	Initializes the MRF24WG WiFi driver.
	DRV_WIFI_Scan	Commands the MRF24W to start a scan operation. This will generate the WF_EVENT_SCAN_RESULTS_READY event.

System Functions

	Name	Description
	DRV_WIFI_BssidGet	Sets the the BSSID set in DRV_WIFI_BssidSet() .
	DRV_WIFI_BssidSet	Sets the the Basic Service Set Identifier (BSSID).
	DRV_WIFI_ChannelListGet	Gets the the channel list.
	DRV_WIFI_ChannelListSet	Sets the the channel list.
	DRV_WIFI_ConnectContextGet	Gets the current WiFi connection context
	DRV_WIFI_ConnectionStateGet	Gets the current WiFi connection state
	DRV_WIFI_DeviceInfoGet	Retrieves MRF24WG device information
	DRV_WIFI_HWMulticastFilterGet	Gets a multicast address filter from one of the two multicast filters.
	DRV_WIFI_HWMulticastFilterSet	Sets a multicast address filter using one of the two hardware multicast filters.
	DRV_WIFI_MacAddressGet	Retrieves the MRF24WG MAC address
	DRV_WIFI_MacAddressSet	Uses a different MAC address for the MRF24W
	DRV_WIFI_MacStatsGet	Gets MAC statistics.
	DRV_WIFI_NetworkTypeGet	Gets the WiFi network type
	DRV_WIFI_NetworkTypeSet	Sets the WiFi network type
	DRV_WIFI_PowerSaveStateGet	Gets the current power-save state.
	DRV_WIFI_ReconnectModeGet	Gets the WiFi reconnection mode.
	DRV_WIFI_ReconnectModeSet	Sets the WiFi reconnection mode.
	DRV_WIFI_RegionalDomainGet	Retrieves the MRF24WG Regional domain

◆	DRV_WIFI_RtsThresholdGet	Gets the RTS Threshold
◆	DRV_WIFI_RtsThresholdSet	Sets the RTS Threshold.
◆	DRV_WIFI_ScanContextGet	Gets the WiFi scan context.
◆	DRV_WIFI_ScanContextSet	Sets the WiFi scan context.
◆	DRV_WIFI_SecurityGet	Gets the current WiFi security setting
◆	DRV_WIFI_SecurityOpenSet	Sets WiFi security to open (no security)
◆	DRV_WIFI_SecurityWepSet	Sets WiFi security to use WEP.
◆	DRV_WIFI_SecurityWpaSet	Sets WiFi security to use WPA or WPA2
◆	DRV_WIFI_SecurityWpsSet	Sets WiFi security to use WPS
◆	DRV_WIFI_SoftApEventInfoGet	Gets the stored Soft AP event info
◆	DRV_WIFI_SsidGet	Sets the SSID
◆	DRV_WIFI_SsidSet	Sets the SSID
◆	DRV_WIFI_SWMulticastFilterSet	Sets a multicast address filter using one of the software multicast filters.
◆	DRV_WIFI_TxModeGet	Gets 802.11 Tx mode
◆	DRV_WIFI_TxModeSet	Configures 802.11 Tx mode
◆	DRV_WIFI_TxPowerFactoryMaxGet	Retrieves the factory-set max Tx power from the MRF24W.
◆	DRV_WIFI_TxPowerMaxGet	Gets the Tx max power on the MRF24WG0M.
◆	DRV_WIFI_TxPowerMaxSet	Sets the Tx max power on the MRF24WG0M.
◆	DRV_WIFI_WepKeyTypeGet	Gets the Wep Key type
◆	DRV_WIFI_WPSCredentialsGet	Gets the WPS credentials
◆	DRV_WIFI_AdhocContextSet	Sets the AdHoc context.
◆	DRV_WIFI_RssiGet	Gets RSSI value set in DRV_WIFI_RssiSet()
◆	DRV_WIFI_RssiSet	Sets RSSI restrictions when connecting
◆	DRV_WIFI_GratuitousArpStart	Starts a periodic gratuitous ARP response
◆	DRV_WIFI_GratuitousArpStop	Stops a periodic gratuitous ARP.
◆	DRV_WIFI_HibernateEnable	Puts the MRF24WG into hibernate mode.
◆	DRV_WIFI_isHibernateEnable	Checks if MRF24W is in hibernate mode
◆	DRV_WIFI_PsPollDisable	Disables PS-Poll mode.
◆	DRV_WIFI_PsPollEnable	Enables PS Poll mode.

Description

This section describes the Application Programming Interface (API) functions of the MRF24W Wi-Fi Driver.

Refer to each section for a detailed description.

5.1.13.7.1 System Functions

5.1.13.7.1.1 DRV_WIFI_BssidGet Function

C

```
void DRV_WIFI_BssidGet(
    uint8_t * p_bssid
);
```

Description

Retrieves the BSSID set in the previous call to [DRV_WIFI_BssidSet\(\)](#).

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_context	pointer to where BSSID will be written.

Returns

None

Remarks

None

Example

```
uint8_t bssid[6];  
DRV_WIFI_BssidGet(bssid);
```

5.1.13.7.1.2 DRV_WIFI_BssidSet Function**C**

```
void DRV_WIFI_BssidSet(  
    uint8_t * p_bssid  
);
```

Description

This sets 6 byte (48-bit) MAC address of the Access Point that is being scanned for. It is optional to use this. Where it is useful is if there are two AP's with the same ID; the BSSID is used to [connect](#) to the specified AP. This setting can be used in lieu of the SSID. Set each byte to 0xFF (default) if the BSSID is not being used. Not typically needed.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_context	pointer to BSSID

Returns

None

Remarks

None

Example

```
uint8_t bssid[6];  
  
bssid[0] = 0x00;  
bssid[1] = 0xe8;  
bssid[2] = 0xc0;  
bssid[3] = 0x11;  
bssid[4] = 0x22;  
bssid[5] = 0x33;  
  
DRV_WIFI_BssidSet(bssid);
```

5.1.13.7.1.3 DRV_WIFI_ChannelListGet Function

C

```
void DRV_WIFI_ChannelListGet(  
    uint8_t * p_channelList,  
    uint8_t * p_numChannels  
);
```

Description

This function gets the current channel list.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_channelList	pointer to where channel list will be written
numChannels	pointer to where number of channels in the list will be written.

Returns

None

Remarks

None

Example

```
uint8_t channelList[DRV_WIFI_MAX_CHANNEL_LIST_LENGTH];  
uint8_t numChannels;  
  
DRV_WIFI_ChannelListGet(channelList, &numChannels);
```

5.1.13.7.1.4 DRV_WIFI_ChannelListSet Function

C

```
void DRV_WIFI_ChannelListSet(  
    uint8_t * p_channelList,  
    uint8_t numChannels  
);
```

Description

This function sets the channel list that the MRF24WG will use when scanning or connecting.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_channelList	list of channels
numChannels	number of channels in list; if set to 0, then MRF24WG will set its channel list to all valid channels in its regional domain.

Returns

None

Remarks

None

Example

```
uint8_t channelList[1,6,11];  
  
DRV_WIFI_ChannelListSet(channelList, sizeof(channelList));
```

5.1.13.7.1.5 DRV_WIFI_ConnectContextGet Function

C

```
void DRV_WIFI_ConnectContextGet(  
    DRV_WIFI_CONNECTION_CONTEXT * p_ctx  
);
```

Description

This function gets the current WiFi connection context.

Preconditions

WiFi initialization must be complete

Parameters

Parameters	Description
p_ctx	pointer to where connection context is written. See DRV_WIFI_CONNECTION_CONTEXT structure.

Returns

None

Remarks

None

Example

```
DRV_WIFI_CONNECTION_CONTEXT ctx;  
  
DRV_WIFI_ConnectContextGet(&ctx);
```

5.1.13.7.1.6 DRV_WIFI_ConnectionStateGet Function

C

```
void DRV_WIFI_ConnectionStateGet(  
    uint8_t * p_state  
);
```

Description

This function gets the current WiFi connection state.

*p_state	Description
DRV_WIFI_CSTATE_NOT_CONNECTED	No WiFi connection exists
DRV_WIFI_CSTATE_CONNECTION_IN_PROGRESS	WiFi connection in progress
DRV_WIFI_CSTATE_CONNECTED_INFRASTRUCTURE	WiFi connected in infrastructure mode
DRV_WIFI_CSTATE_CONNECTED_ADHOC	WiFi connected in adhoc mode

DRV_WIFI_CSTATE_RECONNECTION_IN_PROGRESS	WiFi in process of reconnecting
DRV_WIFI_CSTATE_CONNECTION_PERMANENTLY_LOST	WiFi connection permanently lost

Preconditions

WiFi initialization must be complete

Parameters

Parameters	Description
p_state	pointer to where state is written (see description)

Returns

None

Remarks

None

Example

```
uint8_t state;

DRV_WIFI_ConnectionStateGet(&state);
```

5.1.13.7.1.7 DRV_WIFI_DeviceInfoGet Function**C**

```
void DRV_WIFI_DeviceInfoGet(
    DRV_WIFI_DEVICE_INFO * p_deviceInfo
);
```

Description

This function retrieves MRF24WG device information. See [DRV_WIFI_DEVICE_INFO](#) structure.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_deviceInfo	Pointer where device info will be written

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_DEVICE_INFO    info;

DRV_WIFI_DeviceInfoGet(&info);
```

5.1.13.7.1.8 DRV_WIFI_HWMulticastFilterGet Function**C**

```
void DRV_WIFI_HWMulticastFilterGet(
    uint8_t multicastFilterId,
```

```
    uint8_t multicastAddress[6]
);
```

Description

Gets the current state of the specified Multicast Filter.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
multicastFilterId	DRV_WIFI_MULTICAST_FILTER_1 or DRV_WIFI_MULTICAST_FILTER_2
multicastAddress	pointer to where 6-byte multicast filter is written.

Returns

None.

Remarks

None.

Example

```
uint8_t filterAddress[6];

DRV_WIFI_HWMulticastFilterGet(DRV_WIFI_MULTICAST_FILTER_1, filterAddress);
```

5.1.13.7.1.9 DRV_WIFI_HWMulticastFilterSet Function**C**

```
void DRV_WIFI_HWMulticastFilterSet(
    uint8_t multicastFilterId,
    uint8_t multicastAddress[6]
);
```

Description

This function allows the application to configure up to two hardware Multicast Address Filters on the MRF24W. If two active multicast filters are set up they are OR'd together – the MRF24W will receive and pass to the Host CPU received packets from either multicast address. The allowable values for the multicast filter are:

- DRV_WIFI_MULTICAST_FILTER_1
- DRV_WIFI_MULTICAST_FILTER_2

By default, both Multicast Filters are inactive.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
multicastFilterId	DRV_WIFI_MULTICAST_FILTER_1 or DRV_WIFI_MULTICAST_FILTER_2
multicastAddress	6-byte address (all 0xFF will inactivate the filter)

Returns

None.

Remarks

Cannot mix hardware and software multicast filters.

Example

```
uint8_t multicastFilterId = DRV_WIFI_MULTICAST_FILTER_1;
uint8_t filterAddress[6] = {0x00, 0x01, 0x5e, 0x11, 0x22, 0x33};

DRV_WIFI_HWMulticastFilterSet(multicastFilterId, filterAddress);
```

5.1.13.7.1.10 DRV_WIFI_MacAddressGet Function**C**

```
void DRV_WIFI_MacAddressGet(
    uint8_t * p_mac
);
```

Description

This function retrieves the MRF24WG MAC address.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_mac	Pointer where mac address will be written (must point to a 6-byte buffer)

Returns

None.

Remarks

None.

Example

```
uint8_t mac[6];

DRV_WIFI_MacAddressGet(mac);
```

5.1.13.7.1.11 DRV_WIFI_MacAddressSet Function**C**

```
void DRV_WIFI_MacAddressSet(
    uint8_t * p_mac
);
```

Description

Directs the MRF24W to use the input MAC address instead of its factory-default MAC address. This function does not overwrite the factory default, which is in FLASH memory.

Preconditions

WiFi initialization must be complete. Cannot be called when the MRF24W is in a connected state.

Parameters

Parameters	Description
p_mac	Pointer to 6-byte MAC that will be sent to MRF24WG

Returns

None.

Remarks

None.

Example

```
uint8_t mac[6] = {0x00, 0x1e, 0xc0, 0x11, 0x22, 0x33};

DRV_WIFI_MacAddressSet(mac);
```

5.1.13.7.1.12 DRV_WIFI_MacStatsGet Function

C

```
void DRV_WIFI_MacStatsGet(
    DRV_WIFI_MAC_STATS * p_macStats
);
```

Description

This function gets the various MAC layer stats as maintained by the MRF24WG.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_macStats	Pointer to where MAC statistics are written. See DRV_WIFI_MAC_STATS structure.

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_MAC_STATS macStats;

DRV_WIFI_MacStatsGet(&macStats);
```

5.1.13.7.1.13 DRV_WIFI_NetworkTypeGet Function

C

```
void DRV_WIFI_NetworkTypeGet(
    uint8_t * p_networkType
);
```

Description

This function gets the WiFi network type.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_networkType	Pointer to where the network type will be written.
One of the following	DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE DRV_WIFI_NETWORK_TYPE_ADHOC

DRV_WIFI_NETWORK_TYPE_P2P	not supported DRV_WIFI_NETWORK_TYPE_SOFT_AP

Returns

None

Remarks

None

Example

```
uint8_t networkType;

DRV_WIFI_NetworkTypeGet(&networkType);
```

5.1.13.7.1.14 DRV_WIFI_NetworkTypeSet Function

C

```
void DRV_WIFI_NetworkTypeSet(
    uint8_t networkType
);
```

Description

This function selects the WiFi network type.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
networkType	One of the following: DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE DRV_WIFI_NETWORK_TYPE_ADHOC
DRV_WIFI_NETWORK_TYPE_P2P	not supported DRV_WIFI_NETWORK_TYPE_SOFT_AP

Returns

None

Remarks

None

Example

```
DRV_WIFI_NetworkTypeSet(DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE);
```

5.1.13.7.1.15 DRV_WIFI_PowerSaveStateGet Function

C

```
void DRV_WIFI_PowerSaveStateGet(
    uint8_t * p_powerSaveState
);
```

Description

This function gets the current MRF24WG power save state.

powerSaveState	Definition
DRV_WIFI_PS_HIBERNATE	MRF24W in hibernate state

DRV_WIFI_PS_PS_POLL_DTIM_ENABLED	MRF24W in PS-Poll mode with DTIM enabled
DRV_WIFI_PS_PS_POLL_DTIM_DISABLED	MRF24W in PS-Poll mode with DTIM disabled
DRV_WIFI_PS_POLL_OFF	MRF24W is not in any power-save state

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_powerSaveState	Pointer to where power state is written (see description)

Returns

None.

Remarks

None.

Example

```
uint8_t state;

DRV_WIFI_PowerSaveStateGet(&state);
```

5.1.13.7.1.16 DRV_WIFI_ReconnectModeGet Function**C**

```
void DRV_WIFI_ReconnectModeGet(
    uint8_t * p_retryCount,
    uint8_t * p_deauthAction,
    uint8_t * p_beaconTimeout,
    uint8_t * p_beaconTimeoutAction
);
```

Description

This function gets the reconnection mode parameters set in DRV_WIFI_ReconnectModeGet().

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_retryCount	Pointer where retry count is written
p_deauthAction	Pointer where deauth action is written
p_beaconTimeout	Pointer where beacon timeout is written
p_beaconTimeoutAction	Pointer where beacon timeout action is written.

Returns

None

Remarks

None

Example

```
uint8_t retryCount, deauthAction, beaconTimeout, beaconTimeoutAction;
```

```
DRV_WIFI_ReconnectModeGet(&retryCount,  
                           &deauthAction,  
                           &beaconTimeout,  
                           &beaconTimeoutAction);
```

5.1.13.7.1.17 DRV_WIFI_ReconnectModeSet Function

C

```
void DRV_WIFI_ReconnectModeSet(  
    uint8_t retryCount,  
    uint8_t deauthAction,  
    uint8_t beaconTimeout,  
    uint8_t beaconTimeoutAction  
);
```

Description

This function controls how the MRF24WG behaves when an existing WiFi connection is lost. The MRF24WG can lose an existing connection in one of two ways: 1) Beacon timeout 2) Deauth received from AP

There are two options with respect to regaining a lost WiFi connection: 1) MRF24WG informs the host that the connection was temporarily lost and then the MRF24WG retries N times (or forever) to regain the connection. 2) MRF24WG simply informs the host application that the connection is lost, and it is up to the host to regain the connection via the API.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
retryCount	Number of times the MRF24WG should try to regain the connection (see description)
deauthAction	In the event of a deauth from the AP, the action the MRF24WG should take (see description)
beaconTimeout	Number of missed beacons before the MRF24WG designates the connection as lost (see description)
beaconTimeoutAction	In the event of a beacon timeout, the action the MRF24WG should take (see description)

Returns

None

Remarks

The retryCount parameter also applies when initially connecting. That is, the retryCount tells the MRF24WG how many time to try to **connect** to a WiFi network before giving up and generating the DRV_WIFI_EVENT_CONNECTION_FAILED event.

'retryCount' field	Description
0	Do not try to regain a connection (simply report event to host)
1:254	Number of times MRF24WG should try to regain the connection
255	MRF24WG will retry forever (do not use for AdHoc connections)

'deathAction' field	Description
DRV_WIFI_DO_NOT_ATTEMPT_TO_RECONNECT	Do not attempt to reconnect after a death
DRV_WIFI_ATTEMPT_TO_RECONNECT	Attempt to reconnect after a death

'beaconTimeout' field	Description
0	MRF24WG will not monitor the beacon timeout condition
1:255	Number of missed beacons before designating the connection as lost.

'beaconTimeoutAction' field	Description
DRV_WIFI_DO_NOT_ATTEMPT_TO_RECONNECT	Do not attempt to reconnect after a beacon timeout
DRV_WIFI_ATTEMPT_TO_RECONNECT	Attempt to reconnect after a beacon timeout

None

Example

```
// Example 1: MRF24WG should retry forever if either a death or beacon
// timeout occurs (beacon timeout is 3 beacon periods).
DRV_WIFI_ReconnectModeSet(WF_RETRY_FOREVER,
                          WF_ATTEMPT_TO_RECONNECT,
                          3,
                          WF_ATTEMPT_TO_RECONNECT);

// Example 2: MRF24WG should not do any connection retries and only report
// death events to the host.
DRV_WIFI_ReconnectModeSet(0,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT,
                          0,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT);

// Example 3: MRF24WG should not do any connection retries, but report death
// and beacon timeout events to host. Beacon timeout should be 5 beacon periods.
DRV_WIFI_ReconnectModeSet(0,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT,
                          5,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT);

// Example 4: MRF24WG should ignore beacon timeouts, but attempt to
// reconnect 3 times if a death occurs.
DRV_WIFI_ReconnectModeSet(3,
                          WF_ATTEMPT_TO_RECONNECT,
                          0,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT);
```

5.1.13.7.1.18 DRV_WIFI_RegionalDomainGet Function

C

```
void DRV_WIFI_RegionalDomainGet(
    uint8_t * p_regionalDomain
);
```

Description

Gets the regional domain on the MRF24W. Values are:

- DRV_WIFI_DOMAIN_FCC
- DRV_WIFI_DOMAIN_ETSI
- DRV_WIFI_DOMAIN_JAPAN
- DRV_WIFI_DOMAIN_OTHER

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_regionalDomain	Pointer where the regional domain value will be written

Returns

None.

Remarks

None.

Example

```
uint8_t domain;  
DRV_WIFI_RegionalDomainGet(&domain);
```

5.1.13.7.1.19 DRV_WIFI_RtsThresholdGet Function**C**

```
void DRV_WIFI_RtsThresholdGet(  
    uint16_t * p_rtsThreshold  
);
```

Description

Gets the RTS/CTS packet size threshold.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_rtsThreshold	Pointer to where RTS threshold is written

Returns

None.

Remarks

None.

Example

```
uint16_t threshold;  
DRV_WIFI_RtsThresholdGet(&threshold);
```

5.1.13.7.1.20 DRV_WIFI_RtsThresholdSet Function**C**

```
void DRV_WIFI_RtsThresholdSet(  
    uint16_t rtsThreshold  
);
```

```
uint16_t rtsThreshold  
);
```

Description

Sets the RTS/CTS packet size threshold for when RTS/CTS frame will be sent. The default is 2347 bytes – the maximum for 802.11. It is recommended that the user leave the default at 2347 until they understand the performance and power ramifications of setting it smaller. Valid values are from 0 to [DRV_WIFI_RTS_THRESHOLD_MAX](#) (2347).

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
rtsThreshold	Value of the packet size threshold

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_RtsThresholdSet(DRV_WIFI_RTS_THRESHOLD_MAX);
```

5.1.13.7.1.21 DRV_WIFI_ScanContextGet Function**C**

```
void DRV_WIFI_ScanContextGet(  
    DRV_WIFI_SCAN_CONTEXT * p_context  
);
```

Description

This function gets the WiFi scan context.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_context	Pointer to where scan context will be written. See DRV_WIFI_SCAN_CONTEXT structure.

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_SCAN_CONTEXT context;  
DRV_WIFI_ScanContextSet(&context);
```

5.1.13.7.1.22 DRV_WIFI_ScanContextSet Function

C

```
void DRV_WIFI_ScanContextSet(
    DRV_WIFI_SCAN_CONTEXT * p_context
);
```

Description

This function sets the WiFi scan context. The MRF24WG defaults are fine for most applications, but they can be changed by this function.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_context	Pointer to scan context. See DRV_WIFI_SCAN_CONTEXT structure.

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_SCAN_CONTEXT context;

context.scantype = DRV_WIFI_ACTIVE_SCAN;
context.scanCount = 1;
context.minChannelTime = 200;    // ms
context.maxChannelTime = 400;    // ms
context.probeDelay = 20;         // uS

DRV_WIFI_ScanContextSet(&context);
```

5.1.13.7.1.23 DRV_WIFI_SecurityGet Function

C

```
void DRV_WIFI_SecurityGet(
    uint8_t * p_securityType,
    uint8_t * p_securityKey,
    uint8_t * p_securityKeyLength
);
```

Description

This function gets the current WiFi security setting.

'securityType' Field	Key	Length
DRV_WIFI_SECURITY_OPEN	N/A	N/A
DRV_WIFI_SECURITY_WEP_40	binary	4 keys, 5 bytes each (total of 20 bytes)
DRV_WIFI_SECURITY_WEP_104	binary	4 keys, 13 bytes each (total of 52 bytes)
DRV_WIFI_SECURITY_WPA_WITH_KEY	binary	32 bytes
DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE	ascii	8-63 ascii characters

DRV_WIFI_SECURITY_WPA2_WITH_KEY	binary	32 bytes
DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE	ascii	8-63 ascii characters
DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY	binary	32 bytes
DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	ascii	8-63 ascii characters

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_securityType	Value corresponding to the security type desired (see description)
p_securityKey	Binary key or passphrase (not used if security is DRV_WIFI_SECURITY_OPEN)
p_securityKeyLength	Number of bytes in p_securityKey (not used if security is DRV_WIFI_SECURITY_OPEN)

Returns

None.

Remarks

If security was initially set with a passphrase that the MRF24WG used to generate a binary key, this function returns the binary key, not the passphrase.

Example

```
uint8_t securityType;
uint8_t securityKey[DRV_WIFI_MAX_SECURITY_KEY_LENGTH];
uint8_t keyLength;

DRV_WIFI_SecurityGet(&securityType, securityKey, &keyLength);
```

5.1.13.7.1.24 DRV_WIFI_SecurityOpenSet Function

C

```
void DRV_WIFI_SecurityOpenSet();
```

Description

This function sets the WiFi security to open. One can only [connect](#) to an AP that is running in open mode.

Preconditions

WiFi initialization must be complete. Must be in an unconnected state.

Returns

None

Remarks

None

Example

```
DRV_WIFI_SecurityOpenSet();
```

5.1.13.7.1.25 DRV_WIFI_SecurityWepSet Function

C

```
void DRV_WIFI_SecurityWepSet(
```

```
    DRV_WIFI_WEP_CONTEXT* p_context  
);
```

Description

This function sets the WiFi security to WEP. One can only [connect](#) to an AP that is running the same WEP mode.

Preconditions

WiFi initialization must be complete. Must be in an unconnected state.

Parameters

Parameters	Description
p_context	desired WEP context. See DRV_WIFI_WEP_CONTEXT structure.

Returns

None

Remarks

None

Example

```
DRV_WIFI_WEP_CONTEXT context;  
  
context.wepSecurityType = DRV_WIFI_SECURITY_WEP_40;  
context.wepKey[] = {0x5a, 0xfb, 0x6c, 0x8e, 0x77,  
                   0xc1, 0x04, 0x49, 0xfd, 0x4e,  
                   0x43, 0x18, 0x2b, 0x33, 0x88,  
                   0xb0, 0x73, 0x69, 0xf4, 0x78};  
  
context.wepKeyLength = 20;  
context.wepKeyType = DRV_WIFI_SECURITY_WEP_OPENKEY;  
DRV_WIFI_SecurityOpenSet(&context);
```

5.1.13.7.1.26 DRV_WIFI_SecurityWpaSet Function

C

```
void DRV_WIFI_SecurityWpaSet(  
    DRV_WIFI_WPA_CONTEXT* p_context  
);
```

Description

This function sets the WiFi security to WPA or WPA2. One can only [connect](#) to an AP that is running the same WPA mode.

Preconditions

WiFi initialization must be complete. Must be in an unconnected state.

Parameters

Parameters	Description
p_context	desired WPA context. See DRV_WIFI_WPA_CONTEXT structure.

Returns

None

Remarks

None

Example

```
DRV_WIFI_WPA_CONTEXT context;
```

```
context.wpaSecurityType = DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE;
context.keyInfo.key[] = "MySecretWPA2PassPhrase";
context.keyInfo.keyLenth = strlen(context.keyInfo.key);
DRV_WIFI_SecurityWpaSet(&context);
```

5.1.13.7.1.27 DRV_WIFI_SecurityWpsSet Function

C

```
void DRV_WIFI_SecurityWpsSet(
    DRV_WIFI_WPS_CONTEXT * p_context
);
```

Description

This function sets the WiFi security to WPS. One can only [connect](#) to an AP that supports WPS.

Preconditions

WiFi initialization must be complete. Must be in an unconnected state.

Parameters

Parameters	Description
p_context	desired WPA context. See DRV_WIFI_WPS_CONTEXT structure.

Returns

None

Remarks

None

Example

```
DRV_WIFI_WPS_CONTEXT context;
uint8_t wpsPin[8] = {1, 2, 3, 9, 0, 2, 1, 2};

context.wpsSecurityType = DRV_WIFI_SECURITY_WPS_PUSH_BUTTON;
memcpy(context.wpsPin, wpsPin, sizeof(wpsPin));
context.wpsPinLength = 8;
DRV_WIFI_SecurityWpsSet(&context);
```

5.1.13.7.1.28 DRV_WIFI_SoftApEventInfoGet Function

C

```
DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT * DRV_WIFI_SoftApEventInfoGet();
```

Description

This function retrieves the additional event info after a Soft AP event has occurred.

Preconditions

Soft AP event must have occurred

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT info;;
```

```
DRV_WIFI_WPSCredentialsGet(&info);
```

5.1.13.7.1.29 DRV_WIFI_SsidGet Function

C

```
void DRV_WIFI_SsidGet(  
    uint8_t * p_ssid,  
    uint8_t * p_ssidLength  
);
```

Description

Gets the SSID and SSID Length.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_ssid	Pointer to buffer where SSID will be written
ssidLength	number of bytes in SSID

Returns

None

Remarks

None

Example

```
uint8_t ssid[DRV_WIFI_MAX_SSID_LENGTH];  
uint8_t ssidLength;  
  
DRV_WIFI_SsidGet(ssid, &ssidLength);
```

5.1.13.7.1.30 DRV_WIFI_SsidSet Function

C

```
void DRV_WIFI_SsidSet(  
    uint8_t * p_ssid,  
    uint8_t ssidLength  
);
```

Description

Sets the SSID and SSID Length. Note that an Access Point can have either a visible or hidden SSID. If an Access Point uses a hidden SSID then an active scan must be used.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_ssid	Pointer to SSID buffer
ssidLength	number of bytes in SSID

Returns

None

Remarks

Do not include a string terminator in the SSID length. SSID's are case- sensitive. SSID length must be less than or equal to [DRV_WIFI_MAX_SSID_LENGTH](#).

Example

```
uint8_t ssid[] = "MySSIDName";
uint8_t ssidLength = strlen(ssid);

DRV_WIFI_SsidSet(ssid, ssidLength);
```

5.1.13.7.1.31 DRV_WIFI_SWMulticastFilterSet Function**C**

```
void DRV_WIFI_SWMulticastFilterSet(
    DRV_WIFI_SWMULTICAST_CONFIG * p_config
);
```

Description

This function allows the application to configure up to two Multicast Address Filters on the MRF24W. If two active multicast filters are set up they are OR'd together – the MRF24W will receive and pass to the Host CPU received packets from either multicast address. The allowable values in p_config are:

filterId -- DRV_WIFI_MULTICAST_FILTER_1 thru DRV_WIFI_MULTICAST_FILTER_16

action -- DRV_WIFI_MULTICAST_DISABLE_ALL (default) The Multicast Filter discards all received multicast messages – they will not be forwarded to the Host PIC. The remaining fields in this structure are ignored.

DRV_WIFI_MULTICAST_ENABLE_ALL The Multicast Filter forwards all received multicast messages to the Host PIC. The remaining fields in this structure are ignored.

DRV_WIFI_MULTICAST_USE_FILTERS The MAC filter will be used and the remaining fields in this structure configure which Multicast messages are forwarded to the Host PIC.

macBytes -- Array containing the MAC address to filter on (using the destination address of each incoming 802.11 frame). Specific bytes with the MAC address can be designated as 'don't care' bytes. See macBitMask. This field is only used if action = DRV_WIFI_MULTICAST_USE_FILTERS.

macBitMask -- A byte where bits 5:0 correspond to macBytes[5:0]. If the bit is zero then the corresponding MAC byte must be an exact match for the frame to be forwarded to the Host PIC. If the bit is one then the corresponding MAC byte is a 'don't care' and not used in the Multicast filtering process. This field is only used if action = DRV_WIFI_MULTICAST_USE_FILTERS.

Preconditions

WiFi initialization must be complete. [DRV_WIFI_SWMultiCastFilterEnable\(\)](#) must have been called previously.

Returns

None.

Remarks

Cannot mix hardware and software multicast filters..

Example

```
DRV_WIFI_SWMULTICAST_CONFIG config;
uint8_t macMask[] = {01, 00, 5e, ff, ff, ff}; // (0xff are the don't care bytes)

// configure software multicast filter 1 to filter multicast addresses that
// start with 01:00:5e
config.action = DRV_WIFI_MULTICAST_USE_FILTERS;
config->filterId = DRV_WIFI_MULTICAST_FILTER_1;
```



```

memcpy(config->macBytes, macMask, 6);
config->macBitMask = 0x38;           // bits 5:3 = 1 (don't care on bytes 3,4,5)
                                     // bits 2:0 = 0 (exact match required on bytes
0,1,2)

```

5.1.13.7.1.32 DRV_WIFI_TxModeGet Function

C

```

void DRV_WIFI_TxModeGet(
    uint8_t * p_mode
);

```

Description

This function gets the MRF24WG Tx mode.

Preconditions

WiFi initialization must be complete

Returns

None.

Remarks

None.

Example

```

uint8_t mode;

DRV_WIFI_TxModeGet(&mode);

```

5.1.13.7.1.33 DRV_WIFI_TxModeSet Function

C

```

void DRV_WIFI_TxModeSet(
    uint8_t mode
);

```

Description

This function sets the MRF24WG Tx mode.

mode	Description
DRV_WIFI_TXMODE_G_RATES	Use all 802.11g rates (default)
DRV_WIFI_TXMODE_B_RATES	Use only 802.11b rates
DRV_WIFI_TXMODE_LEGACY_RATES	Use only 1 and 2 mbps rates

Preconditions

WiFi initialization must be complete

Parameters

Parameters	Description
mode	Tx mode value (see description)

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_TxModeSet(DRV_WIFI_TXMODE_G_RATES);
```

5.1.13.7.1.34 DRV_WIFI_TxPowerFactoryMaxGet Function**C**

```
void DRV_WIFI_TxPowerFactoryMaxGet(  
    int8_t * p_factoryMaxTxPower  
);
```

Description

This function retrieves the factory-set max tx power from the MRF24WG.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_factoryMaxTxPower	pointer to where factory max power is written (dbM)

Returns

None.

Remarks

None.

Example

```
int8_t maxPower;  
  
DRV_WIFI_TxPowerFactoryMaxGet(&maxPower);
```

5.1.13.7.1.35 DRV_WIFI_TxPowerMaxGet Function**C**

```
void DRV_WIFI_TxPowerMaxGet(  
    int8_t * p_maxTxPower  
);
```

Description

Gets the Tx max power setting from the MRF24WG.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_maxTxPower	pointer to where max power setting is written (dBm).

Returns

None.

Remarks

None

Example

```
int8_t maxPower;  
  
DRV_WIFI_TxPowerMaxGet(&maxPower);
```

5.1.13.7.1.36 DRV_WIFI_TxPowerMaxSet Function**C**

```
void DRV_WIFI_TxPowerMaxSet(  
    int8_t maxTxPower  
);
```

Description

After initialization the MRF24WG0M max Tx power is determined by a factory-set value. This function can set a different maximum Tx power levels. However, this function can never set a maximum Tx power greater than the factory-set value, which can be read via [DRV_WIFI_TxPowerFactoryMaxGet\(\)](#).

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
maxTxPower	valid range (0 to 17 dBm)

Returns

None.

Remarks

No conversion of units needed, input to MRF24WG0M is in dBm.

Example

```
DRV_WIFI_TxPowerMaxSet(8); // set max Tx power to 8dBm
```

5.1.13.7.1.37 DRV_WIFI_WepKeyTypeGet Function**C**

```
void DRV_WIFI_WepKeyTypeGet(  
    uint8_t * p_wepKeyType  
);
```

Description

This function gets the WEP key type:

- DRV_WIFI_SECURITY_WEP_SHAREDKEY
- DRV_WIFI_SECURITY_WEP_OPENKEY

Preconditions

WiFi initialization must be complete.

Returns

None.

Remarks

None.

Example

```
uint8_t wepKeyType;

DRV_WIFI_WepKeyTypeGet (&wepKeyType);
```

5.1.13.7.1.38 DRV_WIFI_WPSCredentialsGet Function

C

```
void DRV_WIFI_WPSCredentialsGet(
    DRV_WIFI_WPS_CREDENTIAL * p_cred
);
```

Description

This function gets the WPS credentials from the MRF24WG

Preconditions

WiFi initialization must be complete.

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_WPS_CREDENTIAL cred;;

DRV_WIFI_WPSCredentialsGet(&cred);
```

5.1.13.7.1.39 DRV_WIFI_AdhocContextSet Function

C

```
void DRV_WIFI_AdhocContextSet(
    DRV_WIFI_ADHOC_NETWORK_CONTEXT * p_context
);
```

Description

This function sets the AdHoc context. It is only applicable when the [DRV_WIFI_NETWORK_TYPE_ADHOC](#) has been selected in [DRV_WIFI_NetworkTypeSet\(\)](#).

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_context	pointer to AdHoc context data; see definition for the DRV_WIFI_ADHOC_NETWORK_CONTEXT structure.

Returns

None

Remarks

None

Example

```
DRV_WIFI_ADHOC_NETWORK_CONTEXT adHocContext;

adHocContext.mode           = DRV_WIFI_ADHOC_CONNECT_THEN_START;
adHocContext.hiddenSsid     = false;
adHocContext.beaconPeriod  = DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD;

DRV_WIFI_AdhocContextSet(&adHocContext);
```

5.1.13.7.1.40 DRV_WIFI_RssiGet Function

C

```
void DRV_WIFI_RssiGet(
    uint8_t * p_rssi
);
```

Description

This function retrieves the value set in Gets RSSI value set in [DRV_WIFI_RssiSet\(\)](#). It does not retrieve the current connection RSSI value. The scan result will yield the current RSSI.

Preconditions

WiFi initialization must be complete

Parameters

Parameters	Description
p_rssi	pointer where rssi value is written

Returns

None

Remarks

None

Example

```
uint8_t rssi;

DRV_WIFI_RssiGet(&rssi);
```

5.1.13.7.1.41 DRV_WIFI_RssiSet Function

C

```
void DRV_WIFI_RssiSet(
    uint8_t rssi
);
```

Description

This setting is only used if: 1) Neither an SSID or BSSID has been configured or 2) An SSID is defined and multiple AP's are discovered with the same SSID

rssi	Description
0	Connect to first network found
1-254	Only connect to network if the RSSI is greater or equal to this value
255	Connect to the highest RSSI found (default)

Preconditions

WiFi initialization must be complete

Parameters

Parameters	Description
rss	See description

Returns

None

Remarks

Rarely needed

Example

```
DRV_WIFI_RssiSet(255);
```

5.1.13.7.1.42 DRV_WIFI_GratuitousArpStart Function**C**

```
void DRV_WIFI_GratuitousArpStart(  
    uint8_t period  
);
```

Description

This function starts a gratuitous ARP response to be periodically transmitted.

Preconditions

WiFi initialization must be complete. Connection process must be complete.

Parameters

Parameters	Description
period	period between gratuitous ARP, in seconds

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_GratuitousArpStart(10); // begin sending gratuitous ARP's every  
                                // 10 seconds.
```

5.1.13.7.1.43 DRV_WIFI_GratuitousArpStop Function**C**

```
void DRV_WIFI_GratuitousArpStop();
```

Description

This function stops a gratuitous ARP.

Preconditions

WiFi initialization must be complete.

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_GratuitousArpStop();
```

5.1.13.7.1.44 DRV_WIFI_HibernateEnable Function**C**

```
void DRV_WIFI_HibernateEnable();
```

Description

Enables Hibernate mode on the MRF24W, which effectively turns off the device for maximum power savings.

MRF24W state is not maintained when it transitions to hibernate mode.

Preconditions

WiFi initialization must be complete.

Returns

None.

Remarks

None

Example

```
DRV_WIFI_HibernateEnable();
```

5.1.13.7.1.45 DRV_WIFI_isHibernateEnable Function**C**

```
bool DRV_WIFI_isHibernateEnable();
```

Description

Checks if MRF24W is in hibernate mode

Preconditions

WiFi initialization must be complete.

Returns

true or false

Remarks

None

Example

```
bool flag;  
  
flag = DRV_WIFI_isHibernateEnable();
```

5.1.13.7.1.46 DRV_WIFI_PsPollDisable Function

C

```
void DRV_WIFI_PsPollDisable();
```

Description

Disables PS Poll mode. The MRF24W will stay active and not go sleep.

Preconditions

WiFi initialization must be complete.

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_PsPollDisable(&context);
```

5.1.13.7.1.47 DRV_WIFI_PsPollEnable Function

C

```
void DRV_WIFI_PsPollEnable(  
    DRV_WIFI_PS_POLL_CONTEXT * p_context  
);
```

Description

Enables PS Poll mode. PS-Poll (Power-Save Poll) is a mode allowing for longer battery life. The MRF24W coordinates with the Access Point to go to sleep and wake up at periodic intervals to check for data messages, which the Access Point will buffer. The listenInterval in the Connection Algorithm defines the sleep interval. By default, PS-Poll mode is disabled.

When PS Poll is enabled, the WF Host Driver will automatically force the MRF24W to wake up each time the Host sends Tx data or a control message to the MRF24W. When the Host message transaction is complete the MRF24W driver will automatically re-enable PS Poll mode.

When the application is likely to experience a high volume of data traffic then PS-Poll mode should be disabled for two reasons:

- 1. No power savings will be realized in the presence of heavy data traffic.
- 2. Performance will be impacted adversely as the WiFi Host Driver continually activates and deactivates PS-Poll mode via SPI messages.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_context	Pointer to ps poll context. See DRV_WIFI_PS_POLL_CONTEXT structure.

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_PS_POLL_CONTEXT context;  
  
context.listenInterval = DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL;  
context.dtimInterval  = DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL;  
context.useDtim       = DRV_WIFI_DEFAULT_PS_DTIM_ENABLED;  
  
DRV_WIFI_PsPollEnable(&context);
```

5.1.13.7.2 System Configuration Functions

5.1.13.7.2.1 DRV_WIFI_Connect Function

C

```
void DRV_WIFI_Connect();
```

Description

This function causes the MRF24WG to [connect](#) to a WiFi network. Upon connection, or a failure to [connect](#), an event will be generated.

Preconditions

WiFi initialization must be complete and relevant connection parameters must have been set.

Returns

None

Remarks

None

Example

```
DRV_WIFI_Connect();
```

5.1.13.7.2.2 DRV_WIFI_Deinitialize Function

C

```
bool DRV_WIFI_Deinitialize();
```

Description

This function deinitializes the MRF24WG driver. It also saves the WiFi parameters in non-volatile storage.

Preconditions

None.

Returns

If successful returns true, else false.

Remarks

None

5.1.13.7.2.3 DRV_WIFI_Disconnect Function

C

```
uint16_t DRV_WIFI_Disconnect();
```

Description

This function causes the MRF24WG to disconnect from a WiFi network. No event is generated when a connection is terminated via the function call.

Preconditions

WiFi initialization must be complete and a connection must be in progress.

Returns

DRV_WIFI_SUCCESS or DRV_WIFI_ERROR_DISCONNECT_FAILED

Remarks

None

Example

```
DRV_WIFI_Disconnect();
```

5.1.13.7.2.4 DRV_WIFI_Initialize Function

C

```
bool DRV_WIFI_Initialize(
    void* pNetIf
);
```

Description

Initialization Functions

This function initializes the MRF24WG driver, making it ready for clients to use.

Preconditions

None.

Parameters

Parameters	Description
pNetIf	Pointer to network interface

Returns

If successful returns true, else false.

Remarks

This function must be called before any other WiFi routine is called. Currently, this function performs no work, but that may change in the future. The WiFi initialization takes place in a state machine called by MRF24W_MACInit().

5.1.13.7.2.5 DRV_WIFI_Scan Function

C

```
uint16_t DRV_WIFI_Scan(
    bool scanAll
);
```

Description

Directs the MRF24W to initiate a scan operation. The Host Application will be notified that the scan results are ready when it

receives the `WF_EVENT_SCAN_RESULTS_READY` event. The `eventInfo` field for this event will contain the number of scan results. Once the scan results are ready they can be retrieved with [DRV_WIFI_ScanGetResult\(\)](#).

Scan results are retained on the MRF24W until:

1. Calling `DRV_WIFI_Scan()` again (after scan results returned from previous call).
2. MRF24W reset.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description

5.1.13.7.3 Miscellaneous Functions

5.1.13.7.3.1 DRV_WIFI_ConfigDataErase Function

C

```
bool DRV_WIFI_ConfigDataErase();
```

Description

This function erases configuration data from the board EEPROM.

Preconditions

TCPIP stack should be initialized.

Returns

None

Remarks

None.

5.1.13.7.3.2 DRV_WIFI_ConfigDataLoad Function

C

```
bool DRV_WIFI_ConfigDataLoad();
```

Description

This function loads configuration data from the board EEPROM. If not present or corrupted then default values will be used.

Preconditions

TCPIP stack should be initialized.

Returns

None

Remarks

None.

5.1.13.7.3.3 DRV_WIFI_ConfigDataPrint Function

C

```
void DRV_WIFI_ConfigDataPrint();
```

Description

This function outputs configuration data from the board EEPROM.

Preconditions

TCPIP stack should be initialized.

Returns

None

Remarks

None.

5.1.13.7.3.4 DRV_WIFI_ConfigDataSave Function

C

```
bool DRV_WIFI_ConfigDataSave();
```

Description

This function saves configuration data to the board EEPROM.

Preconditions

TCPIP stack should be initialized.

Returns

None

Remarks

None.

5.1.13.7.3.5 DRV_WIFI_ProcessEvent Function

C

```
void DRV_WIFI_ProcessEvent(  
    uint16_t event,  
    uint16_t eventInfo  
);
```

Description

This function is called to process a WiFi event,

Preconditions

TCPIP stack should be initialized.

Parameters

Parameters	Description
event	event code
eventInfo	additional information about the event. Not all events have associated info, in which case this value will be set to DRV_WIFI_NO_ADDITIONAL_INFO (0xff)

Returns

None

Remarks

None.

5.1.13.7.3.6 DRV_WIFI_ScanGetResult Function

C

```
void DRV_WIFI_ScanGetResult(
    uint8_t listIndex,
    DRV_WIFI_SCAN_RESULT * p_scanResult
);
```

Description

After a scan has completed this function is used to read one scan result at a time from the MRF24WG.

Preconditions

WiFi initialization must be complete. WF_EVENT_SCAN_RESULTS_READY event must have already occurred.

Parameters

Parameters	Description
listIndex	Index (0-based list) of the scan entry to retrieve.
p_scanResult	Pointer to where scan result is written. See DRV_WIFI_SCAN_RESULT structure.

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_SCAN_RESULT scanResult;

DRV_WIFI_ScanGetResult(0, &scanResult); // get first scan result in list
```

5.1.13.7.3.7 DRV_WIFI_SetLinkDownThreshold Function

C

```
void DRV_WIFI_SetLinkDownThreshold(
    uint8_t threshold
);
```

Description

This function allows the application to set the number of MRF24W consecutive Tx failures before the connection failure event (DRV_WIFI_LINK_LOST) is reported to the host application.

Preconditions

WiFi initialization must be complete

Parameters

Parameters	Description
threshold -- 0	disabled (default)
1-255	number of consecutive Tx failures before connection failure event is reported

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_SetLinkDownThreshold(0); // disable link down threshold
```

5.1.13.7.3.8 DRV_WIFI_SetPSK Function**C**

```
void DRV_WIFI_SetPSK(  
    uint8_t * p_psk  
);
```

Description

This function is used in conjunction with [DRV_WIFI_YieldPassphraseToHost\(\)](#). It sends the binary key to the MRF24WG after the host has converted an ASCII passphrase to a binary key.

Preconditions

WiFi initialization must be complete.

Parameters

Parameters	Description
p_psk	pointer to the binary key

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_YieldPassphraseToHost(&info);
```

5.1.13.7.3.9 DRV_WIFI_SWMultiCastFilterEnable Function**C**

```
void DRV_WIFI_SWMultiCastFilterEnable();
```

Description

This function allows the application to configure up to max 16 software multicast address Filters on the MRF24WG0MA/B.

Preconditions

WiFi initialization must be complete.

Returns

None.

Remarks

Cannot mix hardware and software multicast filters..

Example

```
DRV_WIFI_SWMultiCastFilterEnable();
```

5.1.13.7.3.10 DRV_WIFI_YieldPassphraseToHost Function

C

```
void DRV_WIFI_YieldPassphraseToHost();
```

Description

During WPS the MRF24WG will receive a passphrase. That passphrase must be converted to a binary key. The MRF24WG can do this, but it will take 20-30 seconds. The host MCU will be able to convert the passphrase to a binary key much faster. If that is desired, then this function should be called to request the MRF24WG to signal the DRV_WIFI_EVENT_KEY_CALCULATION_REQUEST event when the passphrase is ready. Then the host can retrieve the passphrase (see [DRV_WIFI_WPSCredentialsGet\(\)](#)), convert to a binary key (see [DRV_WIFI_ConvPassphraseToKey\(\)](#)), and [send](#) it back to the MRF24WG via the [DRV_WIFI_SetPSK\(\)](#) function.

Preconditions

WiFi initialization must be complete.

Returns

None.

Remarks

None.

Example

```
DRV_WIFI_YieldPassphraseToHost(&info);
```

5.1.13.7.4 Data Types and Constants

5.1.13.7.4.1 DRV_WIFI_BSSID_LENGTH Macro

C

```
#define DRV_WIFI_BSSID_LENGTH (6)
```

Description

This is macro DRV_WIFI_BSSID_LENGTH.

5.1.13.7.4.2 DRV_WIFI_DEAUTH_REASONCODE_MASK Macro

C

```
#define DRV_WIFI_DEAUTH_REASONCODE_MASK ((uint8_t)0x80)
```

Description

This is macro DRV_WIFI_DEAUTH_REASONCODE_MASK.

5.1.13.7.4.3 DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD Macro

C

```
#define DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD (100) // ms
```

Description

ms

5.1.13.7.4.4 DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID Macro

C

```
#define DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID false
```

Description

WiFi AdHoc default settings

These defines identify various default WiFi AdHoc settings that can be used in the [DRV_WIFI_ADHOC_NETWORK_CONTEXT](#) structure.

5.1.13.7.4.5 DRV_WIFI_DEFAULT_ADHOC_MODE Macro

C

```
#define DRV_WIFI_DEFAULT_ADHOC_MODE DRV_WIFI_ADHOC_CONNECT_THEN_START
```

Description

This is macro DRV_WIFI_DEFAULT_ADHOC_MODE.

5.1.13.7.4.6 DRV_WIFI_DEFAULT_PS_DTIM_ENABLED Macro

C

```
#define DRV_WIFI_DEFAULT_PS_DTIM_ENABLED true           // DTIM wake-up enabled (normally  
the case)
```

Description

DTIM wake-up enabled (normally the case)

5.1.13.7.4.7 DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL Macro

C

```
#define DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL ((uint16_t)2) // number of beacon periods
```

Description

number of beacon periods

5.1.13.7.4.8 DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL Macro

C

```
#define DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL ((uint16_t)1) // 100ms multiplier, e.g. 1 *  
100ms = 100ms
```

Description

100ms multiplier, e.g. 1 * 100ms = 100ms

5.1.13.7.4.9 DRV_WIFI_DEFAULT_SCAN_COUNT Macro

C

```
#define DRV_WIFI_DEFAULT_SCAN_COUNT (1)
```

Description

WiFi Scan Context default settings

This enumeration identifies the default WiFi scan context values that can be used in the [DRV_WIFI_SCAN_CONTEXT](#) structure.

5.1.13.7.4.10 DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME Macro

C

```
#define DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME (400) // ms
```

Description

ms

5.1.13.7.4.11 DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME Macro

C

```
#define DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME (200) // ms
```

Description

ms

5.1.13.7.4.12 DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY Macro

C

```
#define DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY (20) // us
```

Description

us

5.1.13.7.4.13 DRV_WIFI_DEFAULT_WEP_KEY_TYPE Macro

C

```
#define DRV_WIFI_DEFAULT_WEP_KEY_TYPE DRV_WIFI_SECURITY_WEP_OPENKEY
```

Description

see [DRV_WIFI_SecurityWepSet\(\)](#) and [DRV_WIFI_WEP_CONTEXT](#)

5.1.13.7.4.14 DRV_WIFI_DISABLED Macro

C

```
#define DRV_WIFI_DISABLED (0)
```

Description

Do not make this an enumerated type!

5.1.13.7.4.15 DRV_WIFI_DISASSOC_REASONCODE_MASK Macro

C

```
#define DRV_WIFI_DISASSOC_REASONCODE_MASK ((uint8_t)0x40)
```

Description

This is macro DRV_WIFI_DISASSOC_REASONCODE_MASK.

5.1.13.7.4.16 DRV_WIFI_ENABLED Macro

C

```
#define DRV_WIFI_ENABLED (1)
```

Description

This is macro DRV_WIFI_ENABLED.

5.1.13.7.4.17 DRV_WIFI_MAX_CHANNEL_LIST_LENGTH Macro

C

```
#define DRV_WIFI_MAX_CHANNEL_LIST_LENGTH (14)
```

Description

This is macro DRV_WIFI_MAX_CHANNEL_LIST_LENGTH.

5.1.13.7.4.18 DRV_WIFI_MAX_NUM_RATES Macro

C

```
#define DRV_WIFI_MAX_NUM_RATES (8)
```

Description

This is macro DRV_WIFI_MAX_NUM_RATES.

5.1.13.7.4.19 DRV_WIFI_MAX_SECURITY_KEY_LENGTH Macro

C

```
#define DRV_WIFI_MAX_SECURITY_KEY_LENGTH (64)
```

Description

This is macro DRV_WIFI_MAX_SECURITY_KEY_LENGTH.

5.1.13.7.4.20 DRV_WIFI_MAX_SSID_LENGTH Macro

C

```
#define DRV_WIFI_MAX_SSID_LENGTH (32)
```

Description

This is macro DRV_WIFI_MAX_SSID_LENGTH.

5.1.13.7.4.21 DRV_WIFI_MAX_WEP_KEY_LENGTH Macro

C

```
#define DRV_WIFI_MAX_WEP_KEY_LENGTH (DRV_WIFI_WEP104_KEY_LENGTH)
```

Description

This is macro DRV_WIFI_MAX_WEP_KEY_LENGTH.

5.1.13.7.4.22 DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH Macro

C

```
#define DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH (64) // must include string terminator
```

Description

must include string terminator

5.1.13.7.4.23 DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH Macro**C**

```
#define DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH (64)
```

Description

This is macro DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH.

5.1.13.7.4.24 DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH Macro**C**

```
#define DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH (8)
```

Description

Key size defines

5.1.13.7.4.25 DRV_WIFI_NETWORK_TYPE_ADHOC Macro**C**

```
#define DRV_WIFI_NETWORK_TYPE_ADHOC (2)
```

Description

This is macro DRV_WIFI_NETWORK_TYPE_ADHOC.

5.1.13.7.4.26 DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE Macro**C**

```
#define DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE (1)
```

Description

WiFi Network Types

This enumeration identifies the WiFi network types that can be selected. Do NOT make these an enumerated type as they are used as a compile switch.

5.1.13.7.4.27 DRV_WIFI_NETWORK_TYPE_P2P Macro**C**

```
#define DRV_WIFI_NETWORK_TYPE_P2P (3)    /* not supported */
```

Description

not supported

5.1.13.7.4.28 DRV_WIFI_NETWORK_TYPE_SOFT_AP Macro**C**

```
#define DRV_WIFI_NETWORK_TYPE_SOFT_AP (4)
```

Description

This is macro DRV_WIFI_NETWORK_TYPE_SOFT_AP.

5.1.13.7.4.29 DRV_WIFI_NO_ADDITIONAL_INFO Macro

C

```
#define DRV_WIFI_NO_ADDITIONAL_INFO ((uint16_t)0xffff)
```

Description

eventInfo define for [DRV_WIFI_ProcessEvent\(\)](#) when no additional info is supplied

5.1.13.7.4.30 DRV_WIFI_RETRY_ADHOC Macro

C

```
#define DRV_WIFI_RETRY_ADHOC (3)
```

Description

This is macro DRV_WIFI_RETRY_ADHOC.

5.1.13.7.4.31 DRV_WIFI_RETRY_FOREVER Macro

C

```
#define DRV_WIFI_RETRY_FOREVER (255)
```

Description

This is macro DRV_WIFI_RETRY_FOREVER.

5.1.13.7.4.32 DRV_WIFI_RTS_THRESHOLD_MAX Macro

C

```
#define DRV_WIFI_RTS_THRESHOLD_MAX (2347) /* maximum RTS threshold size in bytes */
```

Description

maximum RTS threshold size in bytes

5.1.13.7.4.33 DRV_WIFI_SECURITY_EAP Macro

C

```
#define DRV_WIFI_SECURITY_EAP (11)
```

Description

This is macro DRV_WIFI_SECURITY_EAP.

5.1.13.7.4.34 DRV_WIFI_SECURITY_OPEN Macro

C

```
#define DRV_WIFI_SECURITY_OPEN (0)
```

Description

WiFi Security Types

This enumeration identifies the WiFi security types that can be selected. Do NOT make these an enumerated type as they are

used as a compile switch.

5.1.13.7.4.35 DRV_WIFI_SECURITY_WEP_104 Macro

C

```
#define DRV_WIFI_SECURITY_WEP_104 (2)
```

Description

This is macro DRV_WIFI_SECURITY_WEP_104.

5.1.13.7.4.36 DRV_WIFI_SECURITY_WEP_40 Macro

C

```
#define DRV_WIFI_SECURITY_WEP_40 (1)
```

Description

This is macro DRV_WIFI_SECURITY_WEP_40.

5.1.13.7.4.37 DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY Macro

C

```
#define DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY (7)
```

Description

This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY.

5.1.13.7.4.38 DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE Macro

C

```
#define DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE (8)
```

Description

This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE.

5.1.13.7.4.39 DRV_WIFI_SECURITY_WPA_WITH_KEY Macro

C

```
#define DRV_WIFI_SECURITY_WPA_WITH_KEY (3)
```

Description

This is macro DRV_WIFI_SECURITY_WPA_WITH_KEY.

5.1.13.7.4.40 DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE Macro

C

```
#define DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE (4)
```

Description

This is macro DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE.

5.1.13.7.4.41 DRV_WIFI_SECURITY_WPA2_WITH_KEY Macro

C

```
#define DRV_WIFI_SECURITY_WPA2_WITH_KEY (5)
```

Description

This is macro DRV_WIFI_SECURITY_WPA2_WITH_KEY.

5.1.13.7.4.42 DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE Macro

C

```
#define DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE (6)
```

Description

This is macro DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE.

5.1.13.7.4.43 DRV_WIFI_SECURITY_WPS_PIN Macro

C

```
#define DRV_WIFI_SECURITY_WPS_PIN (10)
```

Description

This is macro DRV_WIFI_SECURITY_WPS_PIN.

5.1.13.7.4.44 DRV_WIFI_SECURITY_WPS_PUSH_BUTTON Macro

C

```
#define DRV_WIFI_SECURITY_WPS_PUSH_BUTTON (9)
```

Description

This is macro DRV_WIFI_SECURITY_WPS_PUSH_BUTTON.

5.1.13.7.4.45 DRV_WIFI_VERSION_NUMBER Macro

C

```
#define DRV_WIFI_VERSION_NUMBER "3.0.0"
```

Description

WiFi Driver Version Number

5.1.13.7.4.46 DRV_WIFI_WEP104_KEY_LENGTH Macro

C

```
#define DRV_WIFI_WEP104_KEY_LENGTH (52)           // 4 keys of 13 bytes each
```

Description

4 keys of 13 bytes each

5.1.13.7.4.47 DRV_WIFI_WEP40_KEY_LENGTH Macro

C

```
#define DRV_WIFI_WEP40_KEY_LENGTH (20)           // 4 keys of 5 bytes each
```

Description

4 keys of 5 bytes each

5.1.13.7.4.48 DRV_WIFI_WPA_KEY_LENGTH Macro

C

```
#define DRV_WIFI_WPA_KEY_LENGTH (32)
```

Description

This is macro DRV_WIFI_WPA_KEY_LENGTH.

5.1.13.7.4.49 DRV_WIFI_WPA2_KEY_LENGTH Macro

C

```
#define DRV_WIFI_WPA2_KEY_LENGTH (32)
```

Description

This is macro DRV_WIFI_WPA2_KEY_LENGTH.

5.1.13.7.4.50 DRV_WIFI_WPS_PIN_LENGTH Macro

C

```
#define DRV_WIFI_WPS_PIN_LENGTH (8)           // 7 digits + checksum byte
```

Description

7 digits + checksum byte

5.1.13.7.4.51 DRV_WIFI_ADHOC_MODES Enumeration

C

```
typedef enum adhocMode {
    DRV_WIFI_ADHOC_CONNECT_THEN_START = 0,
    DRV_WIFI_ADHOC_CONNECT_ONLY = 1,
    DRV_WIFI_ADHOC_START_ONLY = 2
} DRV_WIFI_ADHOC_MODES;
```

Description

AdHoc Modes

This enumeration identifies the AdHoc modes that can be selected when connecting in AdHoc mode.

Members

Members	Description
DRV_WIFI_ADHOC_CONNECT_THEN_START = 0	try to connect existing AdHoc network, if not found then start network
DRV_WIFI_ADHOC_CONNECT_ONLY = 1	only connect to existing AdHoc network
DRV_WIFI_ADHOC_START_ONLY = 2	only start a new AdHoc network

5.1.13.7.4.52 DRV_WIFI_ADHOC_NETWORK_CONTEXT Structure

C

```
typedef struct {
    uint8_t mode;
```

```

    bool hiddenSsid;
    uint16_t beaconPeriod;
} DRV_WIFI_ADHOC_NETWORK_CONTEXT;

```

Description

WiFi AdHoc Context

This structure contains MRF24WG AdHoc context data. See [DRV_WIFI_AdhocContextSet\(\)](#).

Members

Members	Description
uint8_t mode;	Defines how to start the AdHoc network. See DRV_WIFI_ADHOC_MODE. Default is DRV_WIFI_ADHOC_CONNECT_THEN_START.
bool hiddenSsid;	When starting an AdHoc network, the SSID can be hidden in the beacons. Set true to hide the SSID, else false. Default is false.
uint16_t beaconPeriod;	Sets the beacon period, in ms. Default is 100ms

5.1.13.7.4.53 DRV_WIFI_CONNECTION_CONTEXT Structure

C

```

typedef struct {
    uint8_t channel;
    uint8_t bssid[6];
} DRV_WIFI_CONNECTION_CONTEXT;

```

Description

WiFi Connection Context

This structure contains MRF24WG connection context data. See [DRV_WIFI_ConnectContextGet\(\)](#).

Members

Members	Description
uint8_t channel;	channel number of current connection
uint8_t bssid[6];	bssid of connected AP

5.1.13.7.4.54 DRV_WIFI_CONNECTION_STATES Enumeration

C

```

typedef enum {
    DRV_WIFI_CSTATE_NOT_CONNECTED = 1,
    DRV_WIFI_CSTATE_CONNECTION_IN_PROGRESS = 2,
    DRV_WIFI_CSTATE_CONNECTED_INFRASTRUCTURE = 3,
    DRV_WIFI_CSTATE_CONNECTED_ADHOC = 4,
    DRV_WIFI_CSTATE_RECONNECTION_IN_PROGRESS = 5,
    DRV_WIFI_CSTATE_CONNECTION_PERMANENTLY_LOST = 6
} DRV_WIFI_CONNECTION_STATES;

```

Description

WiFi Connection States

This enumeration identifies WiFi Connection states states. See [DRV_WIFI_ConnectionStateGet\(\)](#).

Members

Members	Description
DRV_WIFI_CSTATE_NOT_CONNECTED = 1	No WiFi connection exists

DRV_WIFI_CSTATE_CONNECTION_IN_PROGRESS = 2	WiFi connection in progress
DRV_WIFI_CSTATE_CONNECTED_INFRASTRUCTURE = 3	WiFi connected in infrastructure mode
DRV_WIFI_CSTATE_CONNECTED_ADHOC = 4	WiFi connected in adhoc mode
DRV_WIFI_CSTATE_RECONNECTION_IN_PROGRESS = 5	WiFi in process of reconnecting
DRV_WIFI_CSTATE_CONNECTION_PERMANENTLY_LOST = 6	WiFi connection permanently lost

5.1.13.7.4.55 DRV_WIFI_DEVICE_INFO Structure

C

```
typedef struct {
    uint8_t deviceType;
    uint8_t romVersion;
    uint8_t patchVersion;
} DRV_WIFI_DEVICE_INFO;
```

Description

WiFi Device Type/Version

This structure contains MRF24WG device type and version number. See [DRV_WIFI_DeviceInfoGet\(\)](#).

Members

Members	Description
uint8_t deviceType;	MRF24W_DEVICE_TYPE
uint8_t romVersion;	MRF24WG ROM version number
uint8_t patchVersion;	MRF24WG patch version number

5.1.13.7.4.56 DRV_WIFI_DEVICE_TYPES Enumeration

C

```
typedef enum {
    DRV_WIFI_MRF24WB0M_DEVICE = 1,
    DRV_WIFI_MRF24WG0M_DEVICE = 2
} DRV_WIFI_DEVICE_TYPES;
```

Description

WiFi devices types

This enumeration identifies WiFi device types. The only device supported with this driver is DRV_WIFI_MRF24WG0M_DEVICE

5.1.13.7.4.57 DRV_WIFI_DOMAIN_CODES Enumeration

C

```
typedef enum {
    DRV_WIFI_DOMAIN_FCC = 0,
    DRV_WIFI_DOMAIN_ETSI = 2,
    DRV_WIFI_DOMAIN_JAPAN = 7,
    DRV_WIFI_DOMAIN_OTHER = 7
} DRV_WIFI_DOMAIN_CODES;
```

Description

WiFi regional domain codes

This enumeration identifies WiFi regional domain codes. The regional domain can be determined by calling [DRV_WIFI_RegionalDomainGet\(\)](#).

Members

Members	Description
DRV_WIFI_DOMAIN_FCC = 0	FCC, available channels: 1 - 11
DRV_WIFI_DOMAIN_ETSI = 2	ETSI, available Channels: 1 - 13
DRV_WIFI_DOMAIN_JAPAN = 7	Japan, available Channels: 1 - 14
DRV_WIFI_DOMAIN_OTHER = 7	Other, available Channels: 1 - 14

5.1.13.7.4.58 DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES Enumeration

C

```
typedef enum {
    DRV_WIFI_BEACON_TIMEOUT = 1,
    DRV_WIFI_DEAUTH_RECEIVED = 2,
    DRV_WIFI_DISASSOCIATE_RECEIVED = 3
} DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES;
```

Description

'Connection Temporarily Lost' event codes

This enumeration identifies the codes for a connection temporarily lost. These codes are used in [DRV_WIFI_ProcessEvent\(\)](#), case DRV_WIFI_EVENT_CONNECTION_TEMPORARILY_LOST.

Members

Members	Description
DRV_WIFI_BEACON_TIMEOUT = 1	connection temporarily lost due to beacon timeout
DRV_WIFI_DEAUTH_RECEIVED = 2	connection temporarily lost due to deauth received from AP
DRV_WIFI_DISASSOCIATE_RECEIVED = 3	connection temporarily lost due to disassociation received from AP

5.1.13.7.4.59 DRV_WIFI_EVENT_INFO Enumeration

C

```
typedef enum {
    DRV_WIFI_JOIN_FAILURE = 2,
    DRV_WIFI_AUTHENTICATION_FAILURE = 3,
    DRV_WIFI_ASSOCIATION_FAILURE = 4,
    DRV_WIFI_WEP_HANDSHAKE_FAILURE = 5,
    DRV_WIFI_PSK_CALCULATION_FAILURE = 6,
    DRV_WIFI_PSK_HANDSHAKE_FAILURE = 7,
    DRV_WIFI_ADHOC_JOIN_FAILURE = 8,
    DRV_WIFI_SECURITY_MISMATCH_FAILURE = 9,
    DRV_WIFI_NO_SUITABLE_AP_FOUND_FAILURE = 10,
    DRV_WIFI_RETRY_FOREVER_NOT_SUPPORTED_FAILURE = 11,
    DRV_WIFI_LINK_LOST = 12,
    DRV_WIFI_TKIP_MIC_FAILURE = 13,
    DRV_WIFI_RSN_MIXED_MODE_NOT_SUPPORTED = 14,
    DRV_WIFI_RECV_DEAUTH = 15,
    DRV_WIFI_RECV_DISASSOC = 16,
    DRV_WIFI_WPS_FAILURE = 17,
    DRV_WIFI_P2P_FAILURE = 18
} DRV_WIFI_EVENT_INFO;
```

Description

EventInfo types

This enumeration identifies the eventInfo types used in [DRV_WIFI_ProcessEvent\(\)](#), case DRV_WIFI_EVENT_CONNECTION_FAILED.

Members

Members	Description
DRV_WIFI_P2P_FAILURE = 18	not supported

5.1.13.7.4.60 DRV_WIFI_EVENTS Enumeration

C

```
typedef enum {
    DRV_WIFI_EVENT_NONE = 0,
    DRV_WIFI_EVENT_CONNECTION_SUCCESSFUL = 1,
    DRV_WIFI_EVENT_CONNECTION_FAILED = 2,
    DRV_WIFI_EVENT_CONNECTION_TEMPORARILY_LOST = 3,
    DRV_WIFI_EVENT_CONNECTION_PERMANENTLY_LOST = 4,
    DRV_WIFI_EVENT_CONNECTION_REESTABLISHED = 5,
    DRV_WIFI_EVENT_FLASH_UPDATE_SUCCESSFUL = 6,
    DRV_WIFI_EVENT_FLASH_UPDATE_FAILED = 7,
    DRV_WIFI_EVENT_KEY_CALCULATION_REQUEST = 8,
    DRV_WIFI_EVENT_SCAN_RESULTS_READY = 9,
    DRV_WIFI_EVENT_IE_RESULTS_READY = 10,
    DRV_WIFI_EVENT_INVALID_WPS_PIN = 12,
    DRV_WIFI_EVENT_SOFT_AP = 13,
    DRV_WIFI_EVENT_DISCONNECT_DONE = 14,
    DRV_WIFI_EVENT_ERROR = 15
} DRV_WIFI_EVENTS;
```

Description

WiFi Events

This enumeration identifies the WiFi events that can occur and will be sent to [DRV_WIFI_ProcessEvent\(\)](#).

Members

Members	Description
DRV_WIFI_EVENT_NONE = 0	No event has occurred
DRV_WIFI_EVENT_CONNECTION_SUCCESSFUL = 1	Connection attempt to network successful
DRV_WIFI_EVENT_CONNECTION_FAILED = 2	Connection attempt failed
DRV_WIFI_EVENT_CONNECTION_TEMPORARILY_LOST = 3	Connection lost; MRF24W attempting to reconnect
DRV_WIFI_EVENT_CONNECTION_PERMANENTLY_LOST = 4	Connection lost; MRF24W no longer trying to connect
DRV_WIFI_EVENT_CONNECTION_REESTABLISHED = 5	Connection has been reestablished
DRV_WIFI_EVENT_FLASH_UPDATE_SUCCESSFUL = 6	Update to FLASH successful
DRV_WIFI_EVENT_FLASH_UPDATE_FAILED = 7	Update to FLASH failed
DRV_WIFI_EVENT_KEY_CALCULATION_REQUEST = 8	Key calculation is required
DRV_WIFI_EVENT_SCAN_RESULTS_READY = 9	scan results are ready
DRV_WIFI_EVENT_IE_RESULTS_READY = 10	IE data ready
DRV_WIFI_EVENT_INVALID_WPS_PIN = 12	Invalid WPS pin was entered
DRV_WIFI_EVENT_SOFT_AP = 13	Client connection events
DRV_WIFI_EVENT_DISCONNECT_DONE = 14	Disconnect done event
DRV_WIFI_EVENT_ERROR = 15	WiFi error event occurred

5.1.13.7.4.61 DRV_WIFI_GENERAL_ERRORS Enumeration

C

```
typedef enum {
    DRV_WIFI_ERROR_IN_HIBERNATE_MODE = 100
} DRV_WIFI_GENERAL_ERRORS;
```

Description

This is type DRV_WIFI_GENERAL_ERRORS.

Members

Members	Description
DRV_WIFI_ERROR_IN_HIBERNATE_MODE = 100	invalid operation while MRF24WG is in hibernate mode

5.1.13.7.4.62 DRV_WIFI_HIBERNATE_STATE Structure

C

```
typedef struct {
    uint8_t state;
    uint8_t wakeup_notice;
} DRV_WIFI_HIBERNATE_STATE;
```

Description

This is type DRV_WIFI_HIBERNATE_STATE.

5.1.13.7.4.63 DRV_WIFI_HIBERNATE_STATES Enumeration

C

```
typedef enum {
    DRV_WIFI_HB_NO_SLEEP = 0,
    DRV_WIFI_HB_ENTER_SLEEP = 1,
    DRV_WIFI_HB_WAIT_WAKEUP = 2
} DRV_WIFI_HIBERNATE_STATES;
```

Description

WiFi Hibernate states

This enumeration identifies WiFi hibernate states.

5.1.13.7.4.64 DRV_WIFI_MAC_STATS Structure

C

```
typedef struct {
    uint32_t MibWEPExcludeCtr;
    uint32_t MibTxBytesCtr;
    uint32_t MibTxMulticastCtr;
    uint32_t MibTxFailedCtr;
    uint32_t MibTxRtryCtr;
    uint32_t MibTxMultRtryCtr;
    uint32_t MibTxSuccessCtr;
    uint32_t MibRxDupCtr;
    uint32_t MibRxCtsSuccCtr;
    uint32_t MibRxCtsFailCtr;
    uint32_t MibRxAckFailCtr;
    uint32_t MibRxBytesCtr;
    uint32_t MibRxFragCtr;
```

```

uint32_t MibRxMultCtr;
uint32_t MibRxFCSErrCtr;
uint32_t MibRxWEPUndecryptCtr;
uint32_t MibRxFragAgedCtr;
uint32_t MibRxMICFailureCtr;
} DRV_WIFI_MAC_STATS;

```

Description

WiFi MIB states

This structure contains all the MIB data returned from the MRF24WG when [DRV_WIFI_MacStatsGet\(\)](#) is called.

Members

Members	Description
uint32_t MibWEPExcludeCtr;	Number of frames received with the Protected Frame subfield of the Frame Control field set to zero and the value of dot11ExcludeUnencrypted causes that frame to be discarded.
uint32_t MibTxBytesCtr;	Total number of Tx bytes that have been transmitted
uint32_t MibTxMulticastCtr;	Number of frames successfully transmitted that had the multicast bit set in the destination MAC address
uint32_t MibTxFailedCtr;	Number of Tx frames that failed due to the number of transmits exceeding the retry count
uint32_t MibTxRtryCtr;	Number of times a transmitted frame needed to be retried
uint32_t MibTxMultRtryCtr;	Number of times a frame was successfully transmitted after more than one retransmission.
uint32_t MibTxSuccessCtr;	Number of Tx frames successfully transmitted.
uint32_t MibRxDupCtr;	Number of frames received where the Sequence Control field indicates a duplicate.
uint32_t MibRxCTSuccCtr;	Number of CTS frames received in response to an RTS frame.
uint32_t MibRxCTSFailCtr;	Number of times an RTS frame was not received in response to a CTS frame.
uint32_t MibRxAckFailCtr;	Number of times an Ack was not received in response to a Tx frame.
uint32_t MibRxBytesCtr;	Total number of Rx bytes received.
uint32_t MibRxFragCtr;	Number of successful received frames (management or data)
uint32_t MibRxMultCtr;	Number of frames received with the multicast bit set in the destination MAC address.
uint32_t MibRxFCSErrCtr;	Number of frames received with an invalid Frame Checksum (FCS).
uint32_t MibRxWEPUndecryptCtr;	Number of frames received where the Protected Frame subfield of the Frame Control Field is set to one and the WEPOn value for the key mapped to the transmitter's MAC address indicates the frame should not have been encrypted.
uint32_t MibRxFragAgedCtr;	Number of times that fragments aged out, or were not received in the allowable time.
uint32_t MibRxMICFailureCtr;	Number of MIC failures that have occurred.

5.1.13.7.4.65 DRV_WIFI_MGMT_ERRORS Enumeration

C

```

typedef enum {
    DRV_WIFI_SUCCESS = 1,
    DRV_WIFI_ERROR_INVALID_SUBTYPE = 2,
    DRV_WIFI_ERROR_OPERATION_CANCELLED = 3,
    DRV_WIFI_ERROR_FRAME_END_OF_LINE_OCCURRED = 4,
    DRV_WIFI_ERROR_FRAME_RETRY_LIMIT_EXCEEDED = 5,
    DRV_WIFI_ERROR_EXPECTED_BSS_VALUE_NOT_IN_FRAME = 6,
    DRV_WIFI_ERROR_FRAME_SIZE_EXCEEDS_BUFFER_SIZE = 7,
    DRV_WIFI_ERROR_FRAME_ENCRYPT_FAILED = 8,
    DRV_WIFI_ERROR_INVALID_PARAM = 9,
}

```

```

DRV_WIFI_ERROR_AUTH_REQ_ISSUED_WHILE_IN_AUTH_STATE = 10,
DRV_WIFI_ERROR_ASSOC_REQ_ISSUED_WHILE_IN_ASSOC_STATE = 11,
DRV_WIFI_ERROR_INSUFFICIENT_RESOURCES = 12,
DRV_WIFI_ERROR_TIMEOUT_OCCURRED = 13,
DRV_WIFI_ERROR_BAD_EXCHANGE_ENCOUNTERED_IN_FRAME_RECEPTION = 14,
DRV_WIFI_ERROR_AUTH_REQUEST_REFUSED = 15,
DRV_WIFI_ERROR_ASSOCIATION_REQUEST_REFUSED = 16,
DRV_WIFI_ERROR_PRIOR_MGMT_REQUEST_IN_PROGRESS = 17,
DRV_WIFI_ERROR_NOT_IN_JOINED_STATE = 18,
DRV_WIFI_ERROR_NOT_IN_ASSOCIATED_STATE = 19,
DRV_WIFI_ERROR_NOT_IN_AUTHENTICATED_STATE = 20,
DRV_WIFI_ERROR_SUPPLICANT_FAILED = 21,
DRV_WIFI_ERROR_UNSUPPORTED_FEATURE = 22,
DRV_WIFI_ERROR_REQUEST_OUT_OF_SYNC = 23,
DRV_WIFI_ERROR_CP_INVALID_ELEMENT_TYPE = 24,
DRV_WIFI_ERROR_CP_INVALID_PROFILE_ID = 25,
DRV_WIFI_ERROR_CP_INVALID_DATA_LENGTH = 26,
DRV_WIFI_ERROR_CP_INVALID_SSID_LENGTH = 27,
DRV_WIFI_ERROR_CP_INVALID_SECURITY_TYPE = 28,
DRV_WIFI_ERROR_CP_INVALID_SECURITY_KEY_LENGTH = 29,
DRV_WIFI_ERROR_CP_INVALID_WEP_KEY_ID = 30,
DRV_WIFI_ERROR_CP_INVALID_NETWORK_TYPE = 31,
DRV_WIFI_ERROR_CP_INVALID_ADHOC_MODE = 32,
DRV_WIFI_ERROR_CP_INVALID_SCAN_TYPE = 33,
DRV_WIFI_ERROR_CP_INVALID_CP_LIST = 34,
DRV_WIFI_ERROR_CP_INVALID_CHANNEL_LIST_LENGTH = 35,
DRV_WIFI_ERROR_NOT_CONNECTED = 36,
DRV_WIFI_ERROR_ALREADY_CONNECTING = 37,
DRV_WIFI_ERROR_DISCONNECT_FAILED = 38,
DRV_WIFI_ERROR_NO_STORED_BSS_DESCRIPTOR = 39,
DRV_WIFI_ERROR_INVALID_MAX_POWER = 40,
DRV_WIFI_ERROR_CONNECTION_TERMINATED = 41,
DRV_WIFI_ERROR_HOST_SCAN_NOT_ALLOWED = 42,
DRV_WIFI_ERROR_INVALID_WPS_PIN = 44
} DRV_WIFI_MGMT_ERRORS;

```

Description

Mgmt Msg Error Codes

This enumeration identifies the errors that can occur when a DRV_WIFI API function call results in a mgmt message being sent, via SPI, to the MRF24W.

Members

Members	Description
DRV_WIFI_ERROR_DISCONNECT_FAILED = 38	Disconnect failed. Disconnect is allowed only when module is in connected state
DRV_WIFI_ERROR_NO_STORED_BSS_DESCRIPTOR = 39	No stored scan results
DRV_WIFI_ERROR_HOST_SCAN_NOT_ALLOWED = 42	Host Scan Failed. Host scan is allowed only in idle or connected state
DRV_WIFI_ERROR_INVALID_WPS_PIN = 44	WPS pin was invalid

5.1.13.7.4.66 DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT Structure

C

```

typedef struct {
    uint8_t reason;
    uint8_t address[6];
} DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT;

```

Description

WiFi Soft AP Event Information

This structure contains data pertaining to Soft AP event. See [DRV_WIFI_SoftApEventInfoGet\(\)](#).

Members

Members	Description
uint8_t reason;	reason code
uint8_t address[6];	MAC address

5.1.13.7.4.67 DRV_WIFI_MULTICAST_FILTER_IDS Enumeration

C

```
typedef enum {
    DRV_WIFI_MULTICAST_FILTER_1 = 4,
    DRV_WIFI_MULTICAST_FILTER_2 = 5,
    DRV_WIFI_MULTICAST_FILTER_3 = 6,
    DRV_WIFI_MULTICAST_FILTER_4 = 7,
    DRV_WIFI_MULTICAST_FILTER_5 = 8,
    DRV_WIFI_MULTICAST_FILTER_6 = 9,
    DRV_WIFI_MULTICAST_FILTER_7 = 10,
    DRV_WIFI_MULTICAST_FILTER_8 = 11,
    DRV_WIFI_MULTICAST_FILTER_9 = 12,
    DRV_WIFI_MULTICAST_FILTER_10 = 13,
    DRV_WIFI_MULTICAST_FILTER_11 = 14,
    DRV_WIFI_MULTICAST_FILTER_12 = 15,
    DRV_WIFI_MULTICAST_FILTER_13 = 16,
    DRV_WIFI_MULTICAST_FILTER_14 = 17,
    DRV_WIFI_MULTICAST_FILTER_15 = 18,
    DRV_WIFI_MULTICAST_FILTER_16 = 19
} DRV_WIFI_MULTICAST_FILTER_IDS;
```

Description

Multicast Filter ID's

This enumeration identifies the multicast filters that can be selected. See [DRV_WIFI_SWMulticastFilterSet\(\)](#).

5.1.13.7.4.68 DRV_WIFI_MULTICAST_FILTERS Enumeration

C

```
typedef enum {
    DRV_WIFI_MULTICAST_DISABLE_ALL = 0,
    DRV_WIFI_MULTICAST_ENABLE_ALL = 1,
    DRV_WIFI_MULTICAST_USE_FILTERS = 2
} DRV_WIFI_MULTICAST_FILTERS;
```

Description

Multicast Filter Modes

This enumeration identifies the mode of multicast filters that can be selected. See [DRV_WIFI_SWMulticastFilterSet\(\)](#).

Members

Members	Description
DRV_WIFI_MULTICAST_DISABLE_ALL = 0	Discard all received multicast messages
DRV_WIFI_MULTICAST_ENABLE_ALL = 1	forward all multicast messages to host MCU

DRV_WIFI_MULTICAST_USE_FILTERS = 2	Use the MAC filtering capability for multicast messages

5.1.13.7.4.69 DRV_WIFI_POWER_SAVE_STATES Enumeration

C

```
typedef enum {
    DRV_WIFI_PS_HIBERNATE = 1,
    DRV_WIFI_PS_PS_POLL_DTIM_ENABLED = 2,
    DRV_WIFI_PS_PS_POLL_DTIM_DISABLED = 3,
    DRV_WIFI_PS_OFF = 4
} DRV_WIFI_POWER_SAVE_STATES;
```

Description

WiFi Power Save states

This enumeration identifies WiFi Power save states. See [DRV_WIFI_PsPollEnable\(\)](#).

Members

Members	Description
DRV_WIFI_PS_HIBERNATE = 1	enable hibernate mode
DRV_WIFI_PS_PS_POLL_DTIM_ENABLED = 2	enable power save mode with DTIM enabled
DRV_WIFI_PS_PS_POLL_DTIM_DISABLED = 3	enable power save mode with DTIM disabled
DRV_WIFI_PS_OFF = 4	disable Power Save mode

5.1.13.7.4.70 DRV_WIFI_PS_POLL_CONTEXT Structure

C

```
typedef struct {
    uint16_t listenInterval;
    uint16_t dtimInterval;
    bool useDtim;
} DRV_WIFI_PS_POLL_CONTEXT;
```

Description

WiFi PS-Poll Context

This structure contains MRF24WG PS-Poll context data. See [DRV_WIFI_PsPollEnable\(\)](#).

Members

Members	Description
uint16_t listenInterval;	Number of 100ms intervals between instances when the MRF24WG wakes up to received buffered messages from the network. Each count represents 100ms. For example, 1 = 100ms, 2 = 200ms, etc. The default is 1 (100ms).
uint16_t dtimInterval;	Only used if useDtim is true. The DTIM period indicates how often clients serviced by the access point should check for buffered multicast or broadcast messages awaiting pickup on the access point. The DTIM interval is measured in number of beacon periods. Default for DTIM period is 2.
bool useDtim;	True: (default) check for buffered multicast or broadcast messages on the dtimInterval. False: check for buffered multicast or broadcast messages on the listenInterval

5.1.13.7.4.71 DRV_WIFI_REASON_CODES Enumeration

C

```
typedef enum {
    DRV_WIFI_UNSPECIFIED = 1,
    DRV_WIFI_REASON_PREV_AUTH_NOT_VALID = 2,
    DRV_WIFI_DEAUTH_LEAVING = 3,
    DRV_WIFI_DISASSOC_DUE_TO_INACTIVITY = 4,
    DRV_WIFI_DISASSOC_AP_BUSY = 5,
    DRV_WIFI_CLASS2_FRAME_FROM_NONAUTH_STA = 6,
    DRV_WIFI_CLASS3_FRAME_FROM_NONASSOC_STA = 7,
    DRV_WIFI_DISASSOC_STA_HAS_LEFT = 8,
    DRV_WIFI_STA_REQ_ASSOC_WITHOUT_AUTH = 9,
    DRV_WIFI_INVALID_IE = 13,
    DRV_WIFI_MIC_FAILURE = 14,
    DRV_WIFI_4WAY_HANDSHAKE_TIMEOUT = 15,
    DRV_WIFI_GROUP_KEY_HANDSHAKE_TIMEOUT = 16,
    DRV_WIFI_IE_DIFFERENT = 17,
    DRV_WIFI_INVALID_GROUP_CIPHER = 18,
    DRV_WIFI_INVALID_PAIRWISE_CIPHER = 19,
    DRV_WIFI_INVALID_AKMP = 20,
    DRV_WIFI_UNSUPP_RSN_VERSION = 21,
    DRV_WIFI_INVALID_RSN_IE_CAP = 22,
    DRV_WIFI_IEEE8021X_FAILED = 23,
    DRV_WIFI_CIPHER_SUITE_REJECTED = 24
} DRV_WIFI_REASON_CODES;
```

Description

Deauth/Disassociate Reason Codes

This enumeration identifies the reason codes for a connection lost due to a deauthorization or disassociation from the AP.

5.1.13.7.4.72 DRV_WIFI_RECONNECT_MODES Enumeration

C

```
typedef enum {
    DRV_WIFI_DO_NOT_ATTEMPT_TO_RECONNECT = 0,
    DRV_WIFI_ATTEMPT_TO_RECONNECT = 1
} DRV_WIFI_RECONNECT_MODES;
```

Description

WiFi Reconnect Modes

This enumeration identifies the reconnection modes that can be used in [DRV_WIFI_ReconnectModeSet\(\)](#).

5.1.13.7.4.73 DRV_WIFI_SCAN_CONTEXT Structure

C

```
typedef struct {
    uint8_t scanType;
    uint8_t scanCount;
    uint16_t minChannelTime;
    uint16_t maxChannelTime;
    uint16_t probeDelay;
} DRV_WIFI_SCAN_CONTEXT;
```

Description

WiFi Scan Context

This structure contains MRF24WG scan context data. See [DRV_WIFI_ScanContextSet\(\)](#).

Members

Members	Description
uint8_t scanType;	802.11 allows for active scanning, where the device sends out a broadcast probe request seeking an access point. Also allowed is passive scanning where the device only listens to beacons being broadcast from access points. Set to DRV_WIFI_ACTIVE_SCAN (default) or DRV_WIFI_PASSIVE_SCAN
uint8_t scanCount;	The number of times to scan a channel while attempting to find a particular access point. Default is 1
uint16_t minChannelTime;	The minimum time (in milliseconds) the MRF24WG will wait for a probe response after sending a probe request. If no probe responses are received in minChannelTime then the MRF24WG will go on to the next channel, if any are left to scan, or quit. Default is 200ms.
uint16_t maxChannelTime;	If a probe response is received within minChannelTime then the MRF24WG will continue to collect any additional probe responses up to maxChannelTime before going to the next channel in the channelList. Units are in milliseconds. Default is 400ms.
uint16_t probeDelay;	The number of microseconds to delay before transmitting a probe request following the channel change during scanning. Default is 20uS.

5.1.13.7.4.74 DRV_WIFI_SCAN_RESULT Structure**C**

```
typedef struct {
    uint8_t bssid[DRV_WIFI_BSSID_LENGTH];
    uint8_t ssid[DRV_WIFI_MAX_SSID_LENGTH];
    uint8_t apConfig;
    uint8_t reserved;
    uint16_t beaconPeriod;
    uint16_t atimWindow;
    uint8_t basicRateSet[DRV_WIFI_MAX_NUM_RATES];
    uint8_t rssi;
    uint8_t numRates;
    uint8_t DtimPeriod;
    uint8_t bssType;
    uint8_t channel;
    uint8_t ssidLen;
} DRV_WIFI_SCAN_RESULT;
```

Description

WiFi Scan Results

This structure contains the result of WiFi scan operation. See [DRV_WIFI_ScanGetResult\(\)](#).

apConfig Bit Mask

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WPA2	WPA	Preamble	Privacy	Reserved	Reserved	Reserved	IE

IE	1 if AP broadcasting one or more Information Elements, else 0
Privacy	0 : AP is open (no security) 1: AP using security, if neither WPA and WPA2 set then security is WEP.
Preamble	0: AP transmitting with short preamble 1: AP transmitting with long preamble
WPA	Only valid if Privacy is 1. 0: AP does not support WPA 1: AP supports WPA
WPA2	Only valid if Privacy is 1. 0: AP does not support WPA2 1: AP supports WPA2

Members

Members	Description
uint8_t bssid[DRV_WIFI_BSSID_LENGTH];	Network BSSID value
uint8_t ssid[DRV_WIFI_MAX_SSID_LENGTH];	Network SSID value
uint8_t apConfig;	Access Point config (see description)
uint8_t reserved;	not used
uint16_t beaconPeriod;	Network beacon interval
uint16_t atimWindow;	Only valid if bssType = DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE

5.1.13.7.4.75 DRV_WIFI_SCAN_TYPES Enumeration**C**

```
typedef enum {
    DRV_WIFI_ACTIVE_SCAN = 1,
    DRV_WIFI_PASSIVE_SCAN = 2
} DRV_WIFI_SCAN_TYPES;
```

Description

WiFi Scan Types

This enumeration identifies the WiFi scan types that can be selected.

5.1.13.7.4.76 DRV_WIFI_SECURITY_CONTEXT Union**C**

```
typedef union {
    DRV_WIFI_WEP_CONTEXT wepContext;
    DRV_WIFI_WPA_CONTEXT wpaContext;
    DRV_WIFI_WPS_CONTEXT wpsContext;
} DRV_WIFI_SECURITY_CONTEXT;
```

Description

WiFi security context

Structure/union can be used in functions [DRV_WIFI_SecurityWepSet](#), [DRV_WIFI_SecurityWpaSet](#), and [DRV_WIFI_SecurityWpsSet](#)

Members

Members	Description
DRV_WIFI_WEP_CONTEXT wepContext;	set WEP security context
DRV_WIFI_WPA_CONTEXT wpaContext;	set WPA security context
DRV_WIFI_WPS_CONTEXT wpsContext;	set WPS security context

5.1.13.7.4.77 DRV_WIFI_SOFT_AP_EVENT_REASON_CODES Enumeration**C**

```
typedef enum {
    DRV_WIFI_SOFTAP_EVENT_LINK_LOST = 0,
    DRV_WIFI_SOFTAP_EVENT_RECEIVED_DEAUTH = 1
} DRV_WIFI_SOFT_AP_EVENT_REASON_CODES;
```

Description

This is type DRV_WIFI_SOFT_AP_EVENT_REASON_CODES.

5.1.13.7.4.78 DRV_WIFI_SOFT_AP_STATES Enumeration

C

```
typedef enum {
    DRV_WIFI_SOFTAP_EVENT_CONNECTED = 0,
    DRV_WIFI_SOFTAP_EVENT_DISCONNECTED = 1
} DRV_WIFI_SOFT_AP_STATES;
```

Description

This is type DRV_WIFI_SOFT_AP_STATES.

5.1.13.7.4.79 DRV_WIFI_STATUS_CODES Enumeration

C

```
typedef enum {
    DRV_WIFI_UNSPECIFIED_FAILURE = 1,
    DRV_WIFI_CAPS_UNSUPPORTED = 10,
    DRV_WIFI_REASSOC_NO_ASSOC = 11,
    DRV_WIFI_ASSOC_DENIED_UNSPEC = 12,
    DRV_WIFI_NOT_SUPPORTED_AUTH_ALG = 13,
    DRV_WIFI_UNKNOWN_AUTH_TRANSACTION = 14,
    DRV_WIFI_CHALLENGE_FAIL = 15,
    DRV_WIFI_AUTH_TIMEOUT = 16,
    DRV_WIFI_AP_UNABLE_TO_HANDLE_NEW_STA = 17,
    DRV_WIFI_ASSOC_DENIED_RATES = 18,
    DRV_WIFI_ASSOC_DENIED_NOSHORTPREAMBLE = 19,
    DRV_WIFI_ASSOC_DENIED_NOPBCC = 20,
    DRV_WIFI_ASSOC_DENIED_NOAGILITY = 21,
    DRV_WIFI_ASSOC_DENIED_NOSHORTTIME = 25,
    DRV_WIFI_ASSOC_DENIED_NODSSSOFDMA = 26,
    DRV_WIFI_S_INVALID_IE = 40,
    DRV_WIFI_S_INVALID_GROUPCIPHER = 41,
    DRV_WIFI_S_INVALID_PAIRWISE_CIPHER = 42,
    DRV_WIFI_S_INVALID_AKMP = 43,
    DRV_WIFI_UNSUPPORTED_RSN_VERSION = 44,
    DRV_WIFI_S_INVALID_RSN_IE_CAP = 45,
    DRV_WIFI_S_CIPHER_SUITE_REJECTED = 46,
    DRV_WIFI_TIMEOUT = 47
} DRV_WIFI_STATUS_CODES;
```

Description

status codes for connection for association or authentication failure

This enumeration identifies the codes for a connection failure due to association or authentication failure. These codes are used in [DRV_WIFI_ProcessEvent\(\)](#), case DRV_WIFI_EVENT_CONNECTION_FAILED.

5.1.13.7.4.80 DRV_WIFI_SWMULTICAST_CONFIG Structure

C

```
typedef struct {
    uint8_t filterId;
    uint8_t action;
    uint8_t macBytes[6];
    uint8_t macBitMask;
} DRV_WIFI_SWMULTICAST_CONFIG;
```

Description

WiFi SW Multicast Filter Config

This structure contains data pertaining to the configuration of the software multicast config filter.

'action' Field	Description
DRV_WIFI_MULTICAST_DISABLE_ALL	Multicast filter discards all received multicast messages.
DRV_WIFI_MULTICAST_ENABLE_ALL	Multicast filter forwards all received multicast messages to host.
DRV_WIFI_MULTICAST_USE_FILTERS	The MAC filter will be used and the remaining fields define the filter.

Members

Members	Description
uint8_t filterId;	DRV_WIFI_MULTICAST_FILTER_1 thru DRV_WIFI_MULTICAST_FILTER_16
uint8_t action;	configures the multicast filter (see description)
uint8_t macBytes[6];	Array containing the MAC address to filter on (using the destination address of each incoming 802.11 frame). Specific bytes within the MAC address can be designated as 'don't care' bytes. See macBitMask. This field is only used if action = WF_MULTICAST_USE_FILTERS.
uint8_t macBitMask;	A byte where bits 5:0 correspond to macBytes[5:0]. If the bit is zero then the corresponding MAC byte must be an exact match for the frame to be forwarded to the Host PIC. If the bit is one then the corresponding MAC byte is a 'don't care' and not used in the Multicast filtering process. This field is only used if action = WF_MULTICAST_USE_FILTERS.

5.1.13.7.4.81 DRV_WIFI_TX_MODES Enumeration

C

```
typedef enum {
    DRV_WIFI_TXMODE_G_RATES = 0,
    DRV_WIFI_TXMODE_B_RATES = 1,
    DRV_WIFI_TXMODE_LEGACY_RATES = 2
} DRV_WIFI_TX_MODES;
```

Description

Tx Modes

This enumeration identifies the choices the MRF24W Tx mode. It is recommended to use the DRV_WIFI_TXMODE_G_RATES for best performance. See [DRV_WIFI_TxModeSet\(\)](#).

Members

Members	Description
DRV_WIFI_TXMODE_G_RATES = 0	Use 802.11 'g' rates
DRV_WIFI_TXMODE_B_RATES = 1	Use only 802.11 'b' rates
DRV_WIFI_TXMODE_LEGACY_RATES = 2	Use only 1 and 2 mbps rates

5.1.13.7.4.82 DRV_WIFI_WEP_CONTEXT Structure

C

```
typedef struct {
    uint8_t wepSecurityType;
    uint8_t wepKey[DRV_WIFI_MAX_WEP_KEY_LENGTH];
    uint8_t wepKeyLength;
    uint8_t wepKeyType;
} DRV_WIFI_WEP_CONTEXT;
```

Description

WiFi Wep Security Context

This structure contains MRF24WG WEP context. See [DRV_WIFI_SecurityWepSet\(\)](#).

Members

Members	Description
uint8_t wepSecurityType;	DRV_WIFI_SECURITY_WEP_40 or DRV_WIFI_SECURITY_WEP_104
uint8_t wepKey[DRV_WIFI_MAX_WEP_KEY_LENGTH];	Array containing four WEP binary keys. This will be four, 5-byte keys for WEP-40 or four, thirteen-byte keys for WEP-104.
uint8_t wepKeyLength;	number of bytes pointed to by p_wepKey
uint8_t wepKeyType;	DRV_WIFI_SECURITY_WEP_OPENKEY (default) or DRV_WIFI_SECURITY_WEP_SHAREDKEY

5.1.13.7.4.83 DRV_WIFI_WEP_KEY_TYPE Enumeration

C

```
typedef enum {
    DRV_WIFI_SECURITY_WEP_SHAREDKEY = 0,
    DRV_WIFI_SECURITY_WEP_OPENKEY = 1
} DRV_WIFI_WEP_KEY_TYPE;
```

Description

WEP Key Types

This enumeration identifies the choices for the WEP key type when using WEP security. The recommended key type (and default) is Open key.

Members

Members	Description
DRV_WIFI_SECURITY_WEP_SHAREDKEY = 0	use WEP shared key
DRV_WIFI_SECURITY_WEP_OPENKEY = 1	use WEP open key (default)

5.1.13.7.4.84 DRV_WIFI_WPA_CONTEXT Structure

C

```
typedef struct {
    uint8_t wpaSecurityType;
    DRV_WIFI_WPA_KEY_INFO keyInfo;
} DRV_WIFI_WPA_CONTEXT;
```

Description

WiFi WPA context

This structure contains MRF24WG WPA context. See [DRV_WIFI_SecurityWpaSet\(\)](#).

wpaSecurityType	Description
DRV_WIFI_SECURITY_WPA_WITH_KEY	WPA binary key
DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE	WPA passphrase
DRV_WIFI_SECURITY_WPA2_WITH_KEY	WPA2 binary key

DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE	WPA2 passphrase
DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY	Auto-select between WPA/WPA2 with binary key
DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	Auto-select between WPA/WPA2 with passphrase

Members

Members	Description
uint8_t wpaSecurityType;	desired security type (see description)
DRV_WIFI_WPA_KEY_INFO keyInfo;	see DRV_WIFI_WPA_KEY_INFO

5.1.13.7.4.85 DRV_WIFI_WPA_KEY_INFO Structure**C**

```
typedef struct {
    uint8_t key[DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH];
    uint8_t keyLength;
} DRV_WIFI_WPA_KEY_INFO;
```

Description

WiFi WPA Key context

This structure contains MRF24WG WPA key info. This structure is used in the [DRV_WIFI_WPA_CONTEXT](#) and [DRV_WIFI_WPS_CONTEXT](#) structures.

Members

Members	Description
uint8_t key[DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH];	binary key or passphrase
uint8_t keyLength;	number of bytes in binary key (always 32) or passphrase

5.1.13.7.4.86 DRV_WIFI_WPS_AUTH_TYPES Enumeration**C**

```
typedef enum {
    DRV_WIFI_WPS_AUTH_OPEN = 0x01,
    DRV_WIFI_WPS_AUTH_WPA_PSK = 0x02,
    DRV_WIFI_WPS_AUTH_SHARED = 0x04,
    DRV_WIFI_WPS_AUTH_WPA = 0x08,
    DRV_WIFI_WPS_AUTH_WPA2 = 0x10,
    DRV_WIFI_WPS_AUTH_WPA2_PSK = 0x20
} DRV_WIFI_WPS_AUTH_TYPES;
```

Description

This is type DRV_WIFI_WPS_AUTH_TYPES.

5.1.13.7.4.87 DRV_WIFI_WPS_CONTEXT Structure**C**

```
typedef struct {
    uint8_t wpsSecurityType;
    uint8_t wpsPin[DRV_WIFI_WPS_PIN_LENGTH];
    uint8_t wpsPinLength;
    bool getPassPhrase;
    DRV_WIFI_WPA_KEY_INFO * p_keyInfo;
} DRV_WIFI_WPS_CONTEXT;
```

Description

WiFi WPS security context

This structure contains MRF24WG WPS security context. See [DRV_WIFI_SecurityWpsSet\(\)](#).

Members

Members	Description
uint8_t wpsSecurityType;	DRV_WIFI_SECURITY_WPS_PUSH_BUTTON or DRV_WIFI_SECURITY_WPS_PIN
uint8_t wpsPin[DRV_WIFI_WPS_PIN_LENGTH];	if using DRV_WIFI_SECURITY_WPS_PIN then pointer to 8-digit pin
uint8_t wpsPinLength;	should always be 8
bool getPassPhrase;	True if ASCII passphrase should be sent back to host so host can (more quickly) calculate binary key. False if the MRF24WG should calculate the binary key
DRV_WIFI_WPA_KEY_INFO * p_keyInfo;	pointer to where the WiFi driver will store passphrase info (must be global memory)

5.1.13.7.4.88 DRV_WIFI_WPS_CREDENTIAL Structure**C**

```
typedef struct {
    uint8_t ssid[32];
    uint8_t netKey[64];
    uint16_t authType;
    uint16_t encType;
    uint8_t netIdx;
    uint8_t ssidLen;
    uint8_t keyIdx;
    uint8_t keyLen;
    uint8_t bssid[6];
} DRV_WIFI_WPS_CREDENTIAL;
```

Description

WiFi WPS Credentials

This structure contains data pertaining to the configuration of the WiFi WPS credentials.

'authType' Field	Description
DRV_WIFI_WPS_AUTH_OPEN	open security
DRV_WIFI_WPS_AUTH_WPA_PSK	WPA with PSK
DRV_WIFI_WPS_AUTH_SHARED	Shared key
DRV_WIFI_WPS_AUTH_WPA	WPA
DRV_WIFI_WPS_AUTH_WPA2	WPA2
DRV_WIFI_WPS_AUTH_WPA2_PSK	WPA2 with PSK

'encType' Field	Description
DRV_WIFI_WPS_ENC_NONE	No encoding
DRV_WIFI_WPS_ENC_WEP	WEP encoding
DRV_WIFI_WPS_ENC_TKIP	WPS/TKIP encodingShared key

DRV_WIFI_ENC_AES

AES encoding

Members

Members	Description
uint8_t ssid[32];	network SSID
uint8_t netKey[64];	binary security key (not used if security is open)
uint16_t authType;	WPS authorization type (see description)
uint16_t encType;	Encoding type (see description)
uint8_t netIdx;	not used
uint8_t ssidLen;	number of bytes in SSID
uint8_t keyIdx;	Only valid encType = WF_ENC_WEP. This is the index of the WEP key being used.
uint8_t keyLen;	number of bytes in netKey
uint8_t bssid[6];	MAC address of AP

5.1.13.7.4.89 DRV_WIFI_WPS_ENCODE_TYPES Enumeration**C**

```
typedef enum {
    DRV_WIFI_WPS_ENC_NONE = 0x01,
    DRV_WIFI_WPS_ENC_WEP = 0x02,
    DRV_WIFI_WPS_ENC_TKIP = 0x04,
    DRV_WIFI_ENC_AES = 0x08
} DRV_WIFI_WPS_ENCODE_TYPES;
```

Description

This is type DRV_WIFI_WPS_ENCODE_TYPES.

5.1.13.7.4.90 ENABLE_P2P_PRINTS Macro**C**

```
#define ENABLE_P2P_PRINTS ((uint8_t)(1 << 1))    /* not supported */
```

Description

not supported

5.1.13.7.4.91 ENABLE_WPS_PRINTS Macro**C**

```
#define ENABLE_WPS_PRINTS ((uint8_t)(1 << 0))
```

Description

This is macro ENABLE_WPS_PRINTS.

5.1.13.7.4.92 PA6_IO Macro**C**

```
#define PA6_IO LATAbits.LATA6
```

Description

debug, remove later

5.1.13.7.4.93 PA6_TRISTATE Macro

C

```
#define PA6_TRISTATE TRISAbits.TRISA6
```

Description

This is macro PA6_TRISTATE.

5.1.13.7.4.94 SET_PA6_AS_OUTPUT Macro

C

```
#define SET_PA6_AS_OUTPUT TRISAbits.TRISA6 = 0
```

Description

This is macro SET_PA6_AS_OUTPUT.

5.1.13.7.4.95 WF_WPS_PIN_LENGTH Macro

C

```
#define WF_WPS_PIN_LENGTH 8
```

Description

WPS PIN Length

5.1.13.8 Files

Files

Name	Description
drv_wifi.h	WiFi-specific MAC function prototypes called by TCP/IP stack.

Description

5.1.13.8.1 drv_wifi.h

WiFi MAC interface functions

The functions in this header file are accessed by the TCP/IP stack via function pointers.

Enumerations

	Name	Description
	adhocMode	Selection of different AdHoc connection modes
	DRV_WIFI_ADHOC_MODES	Selection of different AdHoc connection modes
	DRV_WIFI_CONNECTION_STATES	Wifi Connection states
	DRV_WIFI_DEVICE_TYPES	Codes for WiFi device types
	DRV_WIFI_DOMAIN_CODES	Regional domain codes.
	DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES	Selection of different codes when WiFi connection is temporarily lost.
	DRV_WIFI_EVENT_INFO	Selection of different EventInfo types

	DRV_WIFI_EVENTS	Selections for events that can occur.
	DRV_WIFI_GENERAL_ERRORS	This is type DRV_WIFI_GENERAL_ERRORS.
	DRV_WIFI_HIBERNATE_STATES	Wifi Hibernate states
	DRV_WIFI_MGMT_ERRORS	Error codes returned when a mgmt message is sent to the MRF24W
	DRV_WIFI_MULTICAST_FILTER_IDS	Selections for software Multicast filter ID's
	DRV_WIFI_MULTICAST_FILTERS	Selections for Software Multicast Filters.
	DRV_WIFI_POWER_SAVE_STATES	Wifi Power Save states
	DRV_WIFI_REASON_CODES	Selection of different codes when a deauth or disassociation event has occurred.
	DRV_WIFI_RECONNECT_MODES	Selection of different Reconnection modes
	DRV_WIFI_SCAN_TYPES	Selection of different WiFi scan types
	DRV_WIFI_SOFT_AP_EVENT_REASON_CODES	This is type DRV_WIFI_SOFT_AP_EVENT_REASON_CODES.
	DRV_WIFI_SOFT_AP_STATES	This is type DRV_WIFI_SOFT_AP_STATES.
	DRV_WIFI_STATUS_CODES	Selection of different codes when WiFi connection fails due to association or authentication failure.
	DRV_WIFI_TX_MODES	Selections for WiFi Tx mode
	DRV_WIFI_WEP_KEY_TYPE	Selections for WEP key type when using WEP security.
	DRV_WIFI_WPS_AUTH_TYPES	This is type DRV_WIFI_WPS_AUTH_TYPES.
	DRV_WIFI_WPS_ENCODE_TYPES	This is type DRV_WIFI_WPS_ENCODE_TYPES.

Functions

	Name	Description
	DRV_WIFI_AdhocContextSet	Sets the AdHoc context.
	DRV_WIFI_BssidGet	Sets the the BSSID set in DRV_WIFI_BssidSet() .
	DRV_WIFI_BssidSet	Sets the the Basic Service Set Identifier (BSSID).
	DRV_WIFI_ChannelListGet	Gets the the channel list.
	DRV_WIFI_ChannelListSet	Sets the the channel list.
	DRV_WIFI_ConfigDataErase	Erases configuration data from the board EEPROM.
	DRV_WIFI_ConfigDataLoad	Loads configuration data from the board EEPROM.
	DRV_WIFI_ConfigDataPrint	Outputs to console the configuration data from the board EEPROM.
	DRV_WIFI_ConfigDataSave	Save configuration data to the board EEPROM.
	DRV_WIFI_Connect	Directs the MRF24WG to connect to a WiFi network.
	DRV_WIFI_ConnectContextGet	Gets the current WiFi connection context
	DRV_WIFI_ConnectionStateGet	Gets the current WiFi connection state
	DRV_WIFI_Deinitialize	Initializes the MRF24WG WiFi driver.
	DRV_WIFI_DeviceInfoGet	Retrieves MRF24WG device information
	DRV_WIFI_Disconnect	Directs the MRF24WG to disconnect from a WiFi network.
	DRV_WIFI_GratuitousArpStart	Starts a periodic gratuitous ARP response
	DRV_WIFI_GratuitousArpStop	Stops a periodic gratuitous ARP.
	DRV_WIFI_HibernateEnable	Puts the MRF24WG into hibernate mode.
	DRV_WIFI_HWMulticastFilterGet	Gets a multicast address filter from one of the two multicast filters.
	DRV_WIFI_HWMulticastFilterSet	Sets a multicast address filter using one of the two hardware multicast filters.
	DRV_WIFI_Initialize	Initializes the MRF24WG WiFi driver.
	DRV_WIFI_IsHibernateEnable	Checks if MRF24W is in hibernate mode
	DRV_WIFI_MacAddressGet	Retrieves the MRF24WG MAC address

	DRV_WIFI_MacAddressSet	Uses a different MAC address for the MRF24W
	DRV_WIFI_MacStatsGet	Gets MAC statistics.
	DRV_WIFI_NetworkTypeGet	Gets the WiFi network type
	DRV_WIFI_NetworkTypeSet	Sets the WiFi network type
	DRV_WIFI_PowerSaveStateGet	Gets the current power-save state.
	DRV_WIFI_ProcessEvent	Processes WiFi event
	DRV_WIFI_PsPollDisable	Disables PS-Poll mode.
	DRV_WIFI_PsPollEnable	Enables PS Poll mode.
	DRV_WIFI_ReconnectModeGet	Gets the WiFi reconnection mode.
	DRV_WIFI_ReconnectModeSet	Sets the WiFi reconnection mode.
	DRV_WIFI_RegionalDomainGet	Retrieves the MRF24WG Regional domain
	DRV_WIFI_RssiGet	Gets RSSI value set in DRV_WIFI_RssiSet()
	DRV_WIFI_RssiSet	Sets RSSI restrictions when connecting
	DRV_WIFI_RtsThresholdGet	Gets the RTS Threshold
	DRV_WIFI_RtsThresholdSet	Sets the RTS Threshold.
	DRV_WIFI_Scan	Commands the MRF24W to start a scan operation. This will generate the WF_EVENT_SCAN_RESULTS_READY event.
	DRV_WIFI_ScanContextGet	Gets the WiFi scan context.
	DRV_WIFI_ScanContextSet	Sets the WiFi scan context.
	DRV_WIFI_ScanGetResult	Read selected scan results back from MRF24W.
	DRV_WIFI_SecurityGet	Gets the current WiFi security setting
	DRV_WIFI_SecurityOpenSet	Sets WiFi security to open (no security)
	DRV_WIFI_SecurityWepSet	Sets WiFi security to use WEP.
	DRV_WIFI_SecurityWpaSet	Sets WiFi security to use WPA or WPA2
	DRV_WIFI_SecurityWpsSet	Sets WiFi security to use WPS
	DRV_WIFI_SetLinkDownThreshold	Sets number of consecutive WiFi Tx failures before link is considered down.
	DRV_WIFI_SetPSK	Sets the binary WPA PSK code in WPS.
	DRV_WIFI_SoftApEventInfoGet	Gets the stored Soft AP event info
	DRV_WIFI_SsidGet	Sets the SSID
	DRV_WIFI_SsidSet	Sets the SSID
	DRV_WIFI_SWMultiCastFilterEnable	Forces the MRF24WG to use software multicast filters instead of hardware multicast filters.
	DRV_WIFI_SWMulticastFilterSet	Sets a multicast address filter using one of the software multicast filters.
	DRV_WIFI_TxModeGet	Gets 802.11 Tx mode
	DRV_WIFI_TxModeSet	Configures 802.11 Tx mode
	DRV_WIFI_TxPowerFactoryMaxGet	Retrieves the factory-set max Tx power from the MRF24W.
	DRV_WIFI_TxPowerMaxGet	Gets the Tx max power on the MRF24WG0M.
	DRV_WIFI_TxPowerMaxSet	Sets the Tx max power on the MRF24WG0M.
	DRV_WIFI_WepKeyTypeGet	Gets the Wep Key type
	DRV_WIFI_WPSCredentialsGet	Gets the WPS credentials
	DRV_WIFI_YieldPassphraseToHost	Allows host to get WPS WPA-PSK passphrase and convert to binary key.

Macros

	Name	Description
	DRV_WIFI_BSSID_LENGTH	This is macro DRV_WIFI_BSSID_LENGTH.

	DRV_WIFI_DEAUTH_REASONCODE_MASK	This is macro DRV_WIFI_DEAUTH_REASONCODE_MASK.
	DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD	ms
	DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID	Default values for WiFi AdHoc settings
	DRV_WIFI_DEFAULT_ADHOC_MODE	This is macro DRV_WIFI_DEFAULT_ADHOC_MODE.
	DRV_WIFI_DEFAULT_PS_DTIM_ENABLED	DTIM wake-up enabled (normally the case)
	DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL	number of beacon periods
	DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL	100ms multiplier, e.g. 1 * 100ms = 100ms
	DRV_WIFI_DEFAULT_SCAN_COUNT	Default values for WiFi scan context
	DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME	ms
	DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME	ms
	DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY	us
	DRV_WIFI_DEFAULT_WEP_KEY_TYPE	see DRV_WIFI_SecurityWepSet() and DRV_WIFI_WEP_CONTEXT
	DRV_WIFI_DISABLED	Do not make this an enumerated type!
	DRV_WIFI_DISASSOC_REASONCODE_MASK	This is macro DRV_WIFI_DISASSOC_REASONCODE_MASK.
	DRV_WIFI_ENABLED	This is macro DRV_WIFI_ENABLED.
	DRV_WIFI_MAX_CHANNEL_LIST_LENGTH	This is macro DRV_WIFI_MAX_CHANNEL_LIST_LENGTH.
	DRV_WIFI_MAX_NUM_RATES	This is macro DRV_WIFI_MAX_NUM_RATES.
	DRV_WIFI_MAX_SECURITY_KEY_LENGTH	This is macro DRV_WIFI_MAX_SECURITY_KEY_LENGTH.
	DRV_WIFI_MAX_SSID_LENGTH	This is macro DRV_WIFI_MAX_SSID_LENGTH.
	DRV_WIFI_MAX_WEP_KEY_LENGTH	This is macro DRV_WIFI_MAX_WEP_KEY_LENGTH.
	DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH	must include string terminator
	DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH	This is macro DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH. H.
	DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH	Key size defines
	DRV_WIFI_NETWORK_TYPE_ADHOC	This is macro DRV_WIFI_NETWORK_TYPE_ADHOC.
	DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE	Selection of different WiFi network types
	DRV_WIFI_NETWORK_TYPE_P2P	not supported
	DRV_WIFI_NETWORK_TYPE_SOFT_AP	This is macro DRV_WIFI_NETWORK_TYPE_SOFT_AP.
	DRV_WIFI_NO_ADDITIONAL_INFO	eventInfo define for DRV_WIFI_ProcessEvent() when no additional info is supplied
	DRV_WIFI_RETRY_ADHOC	This is macro DRV_WIFI_RETRY_ADHOC.
	DRV_WIFI_RETRY_FOREVER	This is macro DRV_WIFI_RETRY_FOREVER.
	DRV_WIFI_RTS_THRESHOLD_MAX	maximum RTS threshold size in bytes
	DRV_WIFI_SECURITY_EAP	This is macro DRV_WIFI_SECURITY_EAP.
	DRV_WIFI_SECURITY_OPEN	Selection of different WiFi security types
	DRV_WIFI_SECURITY_WEP_104	This is macro DRV_WIFI_SECURITY_WEP_104.
	DRV_WIFI_SECURITY_WEP_40	This is macro DRV_WIFI_SECURITY_WEP_40.

	<u>DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY</u>	This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY.
	<u>DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE</u>	This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE.
	<u>DRV_WIFI_SECURITY_WPA_WITH_KEY</u>	This is macro DRV_WIFI_SECURITY_WPA_WITH_KEY.
	<u>DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE</u>	This is macro DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE.
	<u>DRV_WIFI_SECURITY_WPA2_WITH_KEY</u>	This is macro DRV_WIFI_SECURITY_WPA2_WITH_KEY.
	<u>DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE</u>	This is macro DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE.
	<u>DRV_WIFI_SECURITY_WPS_PIN</u>	This is macro DRV_WIFI_SECURITY_WPS_PIN.
	<u>DRV_WIFI_SECURITY_WPS_PUSH_BUTTON</u>	This is macro DRV_WIFI_SECURITY_WPS_PUSH_BUTTON.
	<u>DRV_WIFI_VERSION_NUMBER</u>	WiFi Driver Version Number
	<u>DRV_WIFI_WEP104_KEY_LENGTH</u>	4 keys of 13 bytes each
	<u>DRV_WIFI_WEP40_KEY_LENGTH</u>	4 keys of 5 bytes each
	<u>DRV_WIFI_WPA_KEY_LENGTH</u>	This is macro DRV_WIFI_WPA_KEY_LENGTH.
	<u>DRV_WIFI_WPA2_KEY_LENGTH</u>	This is macro DRV_WIFI_WPA2_KEY_LENGTH.
	<u>DRV_WIFI_WPS_PIN_LENGTH</u>	7 digits + checksum byte
	<u>ENABLE_P2P_PRINTS</u>	not supported
	<u>ENABLE_WPS_PRINTS</u>	This is macro ENABLE_WPS_PRINTS.
	<u>PA6_IO</u>	debug, remove later
	<u>PA6_TRISTATE</u>	This is macro PA6_TRISTATE.
	<u>SET_PA6_AS_OUTPUT</u>	This is macro SET_PA6_AS_OUTPUT.
	<u>WF_WPS_PIN_LENGTH</u>	WPS PIN Length

Structures

	Name	Description
	<u>DRV_WIFI_ADHOC_NETWORK_CONTEXT</u>	Contains data pertaining to WiFi AdHoc context
	<u>DRV_WIFI_CONNECTION_CONTEXT</u>	Contains data pertaining to MRF24WG connection context
	<u>DRV_WIFI_DEVICE_INFO</u>	Contains data pertaining to MRF24WG device type and version number
	<u>DRV_WIFI_HIBERNATE_STATE</u>	This is type DRV_WIFI_HIBERNATE_STATE.
	<u>DRV_WIFI_MAC_STATS</u>	Wifi MIB states
	<u>DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT</u>	Contains data pertaining to WiFi Soft AP event
	<u>DRV_WIFI_PS_POLL_CONTEXT</u>	Contains data pertaining to WiFi PS-Poll context
	<u>DRV_WIFI_SCAN_CONTEXT</u>	Contains data pertaining to WiFi scan context
	<u>DRV_WIFI_SCAN_RESULT</u>	Contains data pertaining to WiFi scan results
	<u>DRV_WIFI_SWMULTICAST_CONFIG</u>	Contains data pertaining to WiFi software multicast filter configuration
	<u>DRV_WIFI_WEP_CONTEXT</u>	Contains data pertaining to WiFi WEP context
	<u>DRV_WIFI_WPA_CONTEXT</u>	Contains data pertaining to WiFi WPA.
	<u>DRV_WIFI_WPA_KEY_INFO</u>	Contains data pertaining to WiFi WPA Key
	<u>DRV_WIFI_WPS_CONTEXT</u>	Contains data pertaining to WiFi WPS security.

	DRV_WIFI_WPS_CREDENTIAL	Contains data pertaining to WiFi WPS Credentials

Unions

	Name	Description
	DRV_WIFI_SECURITY_CONTEXT	Contains data pertaining to WiFi security.

File Name

drv_wifi_mac.h

5.2 Graphics Library Help

5.2.1 Graphics Library Porting Guide

5.2.1.1 Introduction

Porting to the MPLAB Harmony Graphics Library

This help file provides information for porting from the Microchip Library of Applications (MLA) version of the Graphics Library to the Graphics Library within MPLAB Harmony.

5.2.1.2 Release Notes

MPLAB Harmony Version: v0.70b **Release Date:** Graphics Library: 4.00 18Nov2013

New This Release:

This is the first release of the library.

Known Issues:

Nothing to report in this release.

5.2.1.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.2.1.4 Upgrading from the MLA Graphics Library to the MPLAB Harmony Graphics Library

The MPLAB Harmony Graphics Library is a completely new distribution, which is included as part of the MPLAB Harmony installation. No files need to be maintained from an existing MLA Graphics Library installation.

5.2.1.4.1 MPLAB Harmony Graphics Library Key Features

This topic provides a description of the key features of the MPLAB Harmony Graphics Library.

Description

Note: The MPLAB Harmony Graphics Library *does not* support any Microchip 16-bit family devices.

The following are the key features of the MPLAB Harmony Graphics Library:

- Multiple interfaces
- Fully dynamic:
 - Stack initialization/deinitialization
 - Resource management
 - Configurable
- Improved modularity and stack layout
- Interrupt driven operation
- RTOS friendly, with easy RTOS integration

5.2.1.4.2 MPLAB Harmony Graphics Library Structure

This topic describes the structure of the MPLAB Harmony Graphics Library.

Description

The MPLAB Harmony Graphics Library consists of a graphic widget and primitive library, a display layer, a graphics driver, and demonstration applications, all of which support the following structure:

- Lowercase file names
- Underscores are used
- System services are used instead of direct access to peripherals
- Interrupts
- Drivers
- Timer Services
- File System
- Board Support Package (BSP)

By default, the Graphics Library and its corresponding display drivers and demonstration applications are placed into the following locations during installation of MPLAB Harmony:

- `./microchip/harmony/<version>/framework/gfx`
- `./microchip/harmony/<version>/framework/driver/gfx`

- `./microchip/harmony/<version>/apps/gfx`

5.2.1.4.3 MPLAB Harmony Graphics Library Design

This topic discusses the design of the MPLAB Harmony Graphics Library.

Description

The MPLAB Harmony Graphics Library is designed as a part of a system running other applications, middleware, etc. Therefore, it makes use of the system services that are available to other modules in the system or to the application, such as:

- File system
- System interrupts
- System driver services
- System timers
- System device drivers
- System command processor
- System console
- System debug services

Refer to the `./framework/gfx/src/system` folder for the header files that expose the system wide available API.

5.2.1.4.4 MPLAB Harmony Graphics Library API Changes

This topic discusses the reasons behind the API changes to the MPLAB Harmony Graphics Library.

Description

The API is backward compatible in functionality only. All function names and type definitions have changed to reflect the MPLAB Harmony philosophy.

The changes in the library API have been minimized so that the porting process is straightforward and integration with other dependents is made simple. New functionality has been added because of new features, such as dynamic configuration and non-block of the library. Other than these new features, the changes to the API were also made to support commonality between the MPLAB Harmony Graphics Library and the 16-bit Graphics Library.

The API changes are at all layers of the Graphics Library which include the GOL, Primitive, and Driver layers.

5.2.1.4.5 MPLAB Harmony Graphics Library Access Changes

This topic describes the stack access updates that were made to the MPLAB Harmony Graphics Library.

Description

The Graphics Library no longer initializes the display driver, but requires the application to initialize the driver before calling any Graphics Library functions. For example: `GFX_DRV_lcc_Initialize()`

Direct access to the Graphics Driver has been removed.

5.2.1.4.5.1 Graphic Primitive Layer Changes

This topic discusses the changes made to the MLA Graphics Library `primitive.h` file to produce the equivalent MPLAB Harmony Graphics Library `gfx_primitive.h` file.

Description

Changes

The changes are as follows:

- *Prefix:*
 - GFX_ is the generic prefix for functions
 - gfx_ is the generic prefix for primitive level files
- *Non-blocking support* - A task queue has been added to ensure that functions do not block execution of other non-graphics time-critical tasks
- **GFX_Initialize** - This function replaces the InitGraph function. The **GFX_Initialize** function does the following:
 - Sets the line type to SOLID_LINE
 - Sets the current drawing color to **WHITE**
 - Sets the graphic cursor position to the upper left corner of the screen
 - Sets active and visual pages to page 0
 - Clears the active page and disables clipping

The **GFX_Initialize** function should be called before using the Graphics Primitive Layer.

5.2.1.4.5.2 Graphic Object Layer (GOL) Changes

This topic discusses the changes made to the MLA Graphics Library GOL.h file to produce the equivalent MPLAB Harmony Graphics Library **gfx_gol.h** file.

Description

Changes

The changes are as follows.

- *Prefix:*
 - GFX_GOL_ is the generic prefix for all Graphic Object Level functions
 - gfx_gol_ is the generic prefix for all Graphic Object Level files
- *Non-blocking support* – a task queue has been added to ensure that functions do not block execution of other non-graphics time-critical tasks
- **GFX_Initialize** – This function replaces the MLA GOL_Init function, which overrides the equivalent primitive function. This function initializes the graphics library and creates a default style scheme with default settings referenced by the global scheme pointer. **GFX_Initialize** must be called before GOL functions can be used.

5.2.1.4.5.3 Graphic Display Driver Layer Changes

This topic discusses the changes made to the DisplayDriver.h file.

This topic discusses the changes made to the MLA Graphics Library DisplayDriver.h file to produce the equivalent MPLAB Harmony Graphics Library **drv_gfx_display.h** file.

Description

Changes

The changes are as follows:

- *Prefix:*

- DRV_GFX is the generic prefix for all Graphics Driver level functions
- drv_gfx is the generic prefix for all Graphics Driver level files
- *Common API Interface* – This interface establishes the basic set of driver functions and parameters required by the Graphics Primitive Layer to display graphics content on a Liquid Crystal Display. It also lists the initialization and deinitialization functions and parameters required by the application to establish a driver.

5.2.1.4.6 MPLAB Harmony Graphics Library Initialization Changes

This topic describes the initialization updates to the MPLAB Harmony Graphics Library.

Description

The Graphics Library include header files, [gfx_gol.h](#) and [gfx_primitive.h](#), have been updated as follows.

To initialize the library the call is now: [GFX_Initialize\(\)](#)

The default Graphics Library configuration is provided with the library distribution and consists of:

- drv_gfx_<driver>_config_template.h
- [gfx_config_template.h](#)

An example of the graphics initialization can be observed in each of the graphics demonstration distributions.

Note: Both the GOL and primitive layers implement the [GFX_Initialize\(\)](#) function; however, the primitive layer function will be over-ridden and not used if [gfx_gol.c](#) is compiled.

5.2.1.4.7 New MPLAB Harmony Graphics Library API Functions

This topic describes some of the important new API functions in the MPLAB Harmony Graphics Library.

Description

There are many new functions available that have been introduced to support the MPLAB Harmony philosophy. However, there is no concern for the porting process regarding the new API calls as the previous library implementation did not support this kind of service.

Existing applications will not port without using the new APIs:

- Initialization
- Steady-State

5.2.1.4.7.1 Main Program Changes

This topic describes the main program changes in the MPLAB Harmony Graphics Library.

Description

Main Program Changes

There are few changes that must be taken care of by the main program loop. A full working example is given with the file, [main.c](#), which is provided in all of the Graphics Library demonstrations.

The main program should call the following functions:

- [SYS_Initialize\(\)](#)
- [SYS_Tasks\(\)](#)
- [GFX_GOL_ObjectListDraw\(\)](#)

5.2.1.4.8 MPLAB Harmony Graphics Library Configuration Changes

This topic describes the configuration changes in the MPLAB Harmony Graphics Library.

Description

The stack configuration has a completely *new* structure. Each project has its own configuration folder that stores multiple configurations based on CPU platforms and hardware boards.

Note: The `MLA Hardware_Profile.h` file *cannot* be reused in the MPLAB Harmony Graphics Library.

The following is a list of the most important features:

- `system_config.h` – This file contains the working set of configuration definitions
- [gfx_config_template.h](#) – This file lists the entire possible graphics configuration definitions
- `bsp` – This is the parent directory that contains hardware configuration settings for particular Microchip development boards

5.2.1.4.8.1 MPLAB Harmony Graphics Library Configuration

This topic provides information on the configuration of the MPLAB Harmony Graphics Library.

Description

MPLAB Harmony Graphics Library Configuration

Note: This content was not available at time of release and will be added in a future release of MPLAB Harmony.

5.2.1.4.8.2 MPLAB Harmony Graphics Library BSP Configuration

This topic provides information on the BSP configuration of the MPLAB Harmony Graphics Library.

Description

MPLAB Harmony Graphics Library BSP Configuration

The MPLAB Harmony Graphics Library references the following BSP configurations:

- `bsp/meb` - contains board support content for the Multimedia Expansion Board and the Multimedia Expansion Board II (MEB II)
- `bsp/pcap` - contains board support content for the PIC32 mTouch Capacitive Touch Evaluation Board
- `bsp/pictail` - contains board support content for PICTail and subtending connectable liquid crystal displays (LCDs)

5.2.1.4.9 MPLAB Harmony Graphics Library Utilities

This topic describes the MPLAB Harmony Graphics Library utilities.

Description

The MPLAB Harmony Graphics Library includes the following utilities:

- Graphics Display Designer X (GDDX) - This utility can be used for designing and building graphical user interfaces (GUIs) from graphics objects
- Graphics Resource Converter (GRC) - This utility can be used for converting data into logically deployable entities that can be used in a graphics application

5.2.1.5 Porting Applications from the MLA Graphics Library to the MPLAB Harmony Graphics Library

Porting an application from the MLA Graphics Library into the Graphics Library within MPLAB Harmony consists of updating existing Graphics Library function names in your application to the new MPLAB Harmony function names.

5.2.1.5.1 MPLAB Harmony Graphics Library File Name Compliance

This topic provides header file name mapping tables that show the MLA Graphics Library file name and the compliant name within the MPLAB Harmony Graphics Library.

Description

The replacement names chosen to make existing Graphics Library functions compliant with MPLAB Harmony are shown in individual tables per header file name.

The files listed in the tables are located in the following two MPLAB Harmony installation folders:

. \harmony\<version>\framework\gfx

. \harmony\<version>\framework\gfx\src

The following table shows the original and new names used. Note that other than name changes, no other modifications were made to the middleware functions.

MLA Graphics Library File Name	MPLAB Harmony Graphics Library File Name
AnalogClock.h	gfx_gol_analog_clock.h
Button.h	gfx_gol_button.h
Chart.h	gfx_gol_chart.h
CheckBox.h	gfx_gol_check_box.h
DigitalMeter.h	gfx_gol_digital_meter.h
EditBox.h	gfx_gol_edit_box.h
GOL.h	gfx_gol.h
Graphics.h	gfx.h
Grid.h	gfx_gol_grid.h (not supported in this release)
GroupBox.h	gfx_gol_group_box.h
ListBox.h	gfx_gol_list_box.h
Meter.h	gfx_gol_meter.h
Palette.h	gfx_palette.h
Picture.h	gfx_picture.h
Primitive.h	gfx_primitive.h
ProgressBar.h	gfx_gol_progress_bar.h
Radiobutton.h	gfx_gol_radio_button.h
RoundDial.h	gfx_gol_round_dial.h
Slider.h	gfx_gol_scrollbar.h
StaticText.h	gfx_gol_static_text.h

TextEntry.h	gfx_gol_text_entry.h
Transitions.h	gfx_transitions.h
Window.h	gfx_gol_window.h

5.2.1.5.2 MPLAB Harmony Graphics Library Function Name Compliance

This topic provides header file function mapping tables that show the MLA Graphics Library function name and the compliant name within the MPLAB Harmony Graphics Library.

Description

The replacement function names chosen to make the existing MLA Graphics Library function compliant with MPLAB Harmony are shown in individual tables per header file name.

The files listed in the tables are located in the following MPLAB Harmony installation folder:
 .\harmony\<version>\framework\gfx.

Header File: [gfx_display_driver.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
InitGraph()	GFX_PixelsPut()
PutPixel	GFX_BarFill()
GetPixel()	GFX_PixelArrayPut()
SetColor()	GFX_PixelArrayGet()
PutPixel	GFX_PixelPut()
SetColor ()	GFX_SetColor()
GetTransparentColor()	GFX_SetInstance()
TransparentColorEnable()	GFX_SetPage()
TransparentColorDisable()	GFX_Layer()
Bar()	GFX_PixelGet()
(Content pending)	GFX_AlphaBlendWindow()

Header File: [gfx_primitive.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
InitGraph()	GFX_Initialize()
PutPixel	GFX_PixelPut()
GetPixel()	GFX_PixelGet()
SetColor()	GFX_ColorSet()
GetColor	GFX_ColorGet()
TransparentColorEnable()	GFX_TransparentColorSet()
GetTransparentColor()	GFX_TransparentColorGet()
TransparentColorEnable()	GFX_TransparentColorEnable()
TransparentColorDisable()	GFX_TransparentColorDisable()
Bar()	GFX_BarDraw()

Header File: [gfx_palette.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
PaletteInit()	GFX_PaletteInit()
EnablePalette()	GFX_PaletteEnable()
DisablePalette()	GFX_PaletteDisable()
IsPaletteEnabled()	GFX_IsPaletteEnabled()
GetPaletteChangeError()	GFX_PaletteChangeErrorGet()
ClearPaletteChangeError()	GFX_PaletteChangeErrorClear()
SetPaletteBpp()	GFX_PaletteBppSet()
RequestPaletteChange()	GFX_PaletteChangeRequest()
RequestEntirePaletteChange()	GFX_EntirePaletteChangeRequest()
SetPalette()	GFX_PaletteSet()
SetEntirePalette()	GFX_EntirePalette()
SetPaletteFlash()	GFX_PaletteFlashSet()

Header File: [gfx_transitions.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
GFXIsTransitionPending	No equivalent
GFXExecutePendingTransition	No equivalent
GFXSetupTransition	GFX_SetTransitionParameters()
GFXTransition	GFX_Transition()
GFXTranslateDirection	No equivalent

Header File: [gfx_gol.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
GetState()	GFX_GOL_ObjectStateGet()
ClearState()	GFX_GOL_ObjectStateClear()
SetState()	GFX_GOL_ObjectStateSet()
GOLAddObject()	GFX_GOL_ObjectAdd()
GOLFindObject()	GFX_GOL_ObjectFind()
GOLRedraw()	GFX_GOL_ObjectListRedraw()
GOLRedrawRec()	GFX_GOL_ObjectRectangleRedraw()
GOLDraw()	GFX_GOL_ObjectListDraw()
GOLDrawComplete()	GFX_GOL_ObjectDrawComplete()
GOLDrawCallback()	GFX_GOL_DRAW_CALLBACK_FUNC
GOLFree()	GFX_GOL_ObjectListFree()
GetObjType()	GFX_GOL_ObjectTypeGet()
GetObjID	GFX_GOL_ObjectNextGet()
GetObjNext()	GFX_GOL_ObjectNext()

GOLDeleteObject()	GFX_GOL_ObjectDelete()
GOLDeleteObjectByID()	GFX_GOL_ObjectByIDDelete()
GOLNewList()	GFX_GOL_ObjectListNew()
GOLGetList()	GFX_GOL_ObjectListGet()
GOLSetFocus()	GFX_GOL_ObjectFocusSet()
IsObjUpdated()	GFX_GOL_ObjectUpdated()
GOLInit()	GFX_Initialize()
GOLGetFocus()	GFX_GOL_ObjectFocusGet()
GOLCanBeFocused()	GFX_GOL_ObjectCanBeFocused()
GOLGetFocusNext()	GFX_GOL_ObjectFocusNextGet()
GOLGetFocusPrev()	GFX_GOL_ObjectFocusPrevGet()
GOLPanelDraw()	GFX_GOL_PanelParameterSet()
GOLPanelDrawTsk()	GFX_GOL_PanelDraw()
GOLTwoTonePanelDraw()	GFX_GOL_TwoTonePanelDraw()

Header File: [gfx_gol_analog_clock.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
AcSetMinute()	GFX_GOL_AnalogMinuteSet()
AcSetSecond()	GFX_GOL_AnalogSecondSet()
AcCreate()	GFX_GOL_AnalogClockCreate()
AcDraw()	GFX_GOL_AnalogDraw()
AcSetHour()	GFX_GOL_AnalogHourSet()

Header File: [gfx_gol_button.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
BtnCreate()	GFX_GOL_ButtonCreate()
BtnDraw()	GFX_GOL_ButtonDraw()
BtnGetText()	GFX_GOL_ButtonTextGet()
BtnSetText()	GFX_GOL_ButtonTextSet()
BtnGetBitmap()	GFX_GOL_ButtonPressStateImageGet()
BtnSetBitmap()	GFX_GOL_ButtonPressStateImageSet()
BtnMsgDefault()	GFX_GOL_ButtonActionSet()
BtnTranslateMsg()	GFX_GOL_ButtonActionGet()

Header File: [gfx_gol_check_box.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
CbCreate()	GFX_GOL_CheckBoxCreate()
CbDraw()	GFX_GOL_CheckBoxDraw()
CbGetText()	GFX_GOL_CheckBoxTextGet()

CbSetText()	GFX_GOL_CheckBoxTextSet()
CbMsgDefault()	GFX_GOL_CheckBoxActionSet()
CbTranslateMsg()	GFX_GOL_CheckBoxActionGet()

Header File: [gfx_gol_custom_control.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
No equivalent	GFX_GOL_CustomControlGetPos ()
No equivalent	GFX_GOL_CustomControlSetPos ()
No equivalent	GFX_GOL_CustomControlCreate
No equivalent	GFX_GOL_CustomControlActionGet ()
No equivalent	GFX_GOL_CustomControlActionGet ()
No equivalent	GFX_GOL_CustomControlActionSet ()
No equivalent	GFX_GOL_CustomControlDraw ()

Header File: [gfx_gol_digital_meter.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
DmCreate()	GFX_GOL_DigitalMeterCreate()
DmDraw()	GFX_GOL_DigitalMeterDraw()
DmGetValue()	GFX_GOL_DigitalMeterValueGet()
DmSetValue()	GFX_GOL_DigitalMeterValueSet()
DmDecVal()	GFX_GOL_DigitalMeterDecrement()
DmIncVal()	GFX_GOL_DigitalMeterIncrement()
DmTranslateMsg()	GFX_GOL_DigitalMeterActionGet()

Header File: [gfx_gol_edit_box.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
EbCreate()	GFX_GOL_EditBoxCreate()
EbDraw()	GFX_GOL_EditBoxDraw()
EbGetText()	GFX_GOL_EditBoxTextGet()
EbSetText()	GFX_GOL_EditBoxTextSet()
EbAddChar()	GFX_GOL_EditBoxCharAdd()
EbDeleteChar()	GFX_GOL_EditBoxCharDelete()
EbMsgDefault()	GFX_GOL_EditBoxActionSet()
EbTranslateMsg()	GFX_GOL_EditBoxActionGet()

Header File: [gfx_gol_group_box.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
GbCreate()	GFX_GOL_GroupBoxCreate()
GbDraw()	GFX_GOL_GroupBoxDraw()

GbGetText()	GFX_GOL_GroupBoxTextGet()
GbSetText()	GFX_GOL_GroupBoxTextGet()
GbTranslateMsg()	GFX_GOL_GroupBoxActionGet()
GbCreate()	GFX_GOL_GroupBoxCreate()
GbDraw()	GFX_GOL_GroupBoxDraw()

Header File: [gfx_gol_list_box.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
LbCreate()	GFX_GOL_ListBoxCreate()
LbDraw()	GFX_GOL_ListBoxDraw()
LbGetItemList()	GFX_GOL_ListBoxItemListGet()
LbAddItem()	GFX_GOL_ListBoxItemAdd()
LbAddItem()	GFX_GOL_ListBoxItemAdd()
LbDelItem()	GFX_GOL_ListBoxItemRemove()
LbChangeSel()	GFX_GOL_ListBoxChangeSelection()
LbSetSel()	GFX_GOL_ListBoxSelectionSet()
LbGetSel()	GFX_GOL_ListBoxSelectionGet()
LbGetFocusedItem()	GFX_GOL_ListBoxFocusedItemGet()
LbSetFocusedItem()	GFX_GOL_ListBoxFocusedItemSet()
LbGetCount(pLb)	GFX_GOL_ListBoxCountGet()
LbGetVisibleCount()	GFX_GOL_ListBoxVisibleCountGet()
LbSetBitmap()	GFX_GOL_ListBoxBitmapSet()
LbGetBitmap(plItem)	GFX_GOL_ListBoxBitmapSet()

Header File: [gfx_gol_meter.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
MtrCreate()	GFX_GOL_MeterCreate()
MtrDraw()	GFX_GOL_MeterDraw()
MtrSetVa()	GFX_GOL_MeterValueSet()
MtrGetVal()	GFX_GOL_MeterValueGet()
MtrDecVal()	GFX_GOL_MeterDecrement()
MtrIncVal	GFX_GOL_MeterIncrement()
MtrSetScaleColors	GFX_GOL_MeterScaleColorsSet()
MtrSetTitleFont(pMtr, pNewFont)	GFX_GOL_MeterTitleFontSet()
MtrSetValueFont(pMtr, pNewFont)	GFX_GOL_MeterValueFontSet()
MtrMsgDefault()	GFX_GOL_MeterDefaultActionSet()
MtrTranslateMsg()	GFX_GOL_MeterActionGet()

Header File: [gfx_gol_picture.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
PictCreat()	GFX_GOL_PictureControlCreate()
PictDraw()	GFX_GOL_PictureControlDraw()
PictSetBitmap()	GFX_GOL_PictureControlBitmapSet()
PictGetBitmap()	GFX_GOL_PictureControlBitmapGet()
PictGetScale()	GFX_GOL_PictureControlScaleGet()
PictSetScale()	GFX_GOL_PictureControlScaleGet()
PictTranslateMsg()	GFX_GOL_PictureControlActionGet()

Header File: [gfx_gol_progress_bar.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
PbCreate()	GFX_GOL_ProgressBarCreate()
PbDraw()	GFX_GOL_ProgressBarDraw()
PbSetRange()	GFX_GOL_ProgressBarRangeSet()
PbGetRange()	GFX_GOL_ProgressBarRangeGet()
PbSetPos()	GFX_GOL_ProgressBarPositionSet()
PbGetPos()	GFX_GOL_ProgressBarActionSet()
PbTranslateMsg()	GFX_GOL_ProgressBarActionGet()

Header File: [gfx_gol_radio_button.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
RbCreate()	GFX_GOL_RadioButtonCreate()
RbDraw()	GFX_GOL_RadioButtonDraw()
RbGetCheck()	GFX_GOL_RadioButtonCheckGet()
RbSetCheck()	GFX_GOL_RadioButtonCheckGet()
RbGetText(pRb)	GFX_GOL_RadioButtonTextGet()
RbSetText()	GFX_GOL_RadioButtonTextGet()
RbMsgDefault()	GFX_GOL_RadioButtonActionSet()

Header File: [gfx_gol_scrollbar.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
SldCreate()	GFX_GOL_ScrollBarCreate()
SldDraw()	GFX_GOL_ScrollBarDraw()
SldSetPage()	GFX_GOL_ScrollBarPageSet()
SldGetPage()	GFX_GOL_ScrollBarPageGet()
SldSetPos()	GFX_GOL_ScrollBarPositionSet()
SldGetPos(pSld)	GFX_GOL_ScrollBarPositionGet()
SldSetRange()	GFX_GOL_ScrollBarSetRange()
SldGetRange()	GFX_GOL_ScrollBarRangeGet()

SldIncPos()	GFX_GOL_ScrollBarPositionIncrement()
SldDecPos()	GFX_GOL_ScrollBarPositionDecrement()
SldMsgDefault()	GFX_GOL_ScrollBarActionSet()

Header File: [gfx_gol_static_text.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
StCreate()	GFX_GOL_StaticTextCreate()
StDraw()	GFX_GOL_StaticTextDraw()
StGetText()	GFX_GOL_StaticTextGet()
StSetText()	GFX_GOL_StaticTextSet()
StTranslateMsg()	GFX_GOL_StaticTextActionGet()

Header File: [gfx_gol_text_entry.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
TeCreate()	GFX_GOL_TextEntryCreate()
TeDraw()	GFX_GOL_TextEntryDraw()
TeGetBuffer()	GFX_GOL_TextEntryBufferGet()
TeSetBuffer()	GFX_GOL_TextEntryBufferSet()
TeClearBuffer()	GOL_GOL_TextEntryBufferClear()
TeGetKeyCommand()	GOL_GOL_TextEntryKeyCommandGet()
TeSetKeyCommand()	GOL_GOL_TextEntryKeyCommandSet()
TeCreateKeyMembers()	GOL_GOL_TextEntryKeyMembersCreate()
TeAddChar()	GOL_GOL_TextEntryCharAdd()
TeIsKeyPressed()	GOL_GOL_TextEntryKeyPressed()
TeSpaceChar()	GOL_GOL_TextEntryCharSpace()
TeDelKeyMembers()	GOL_GOL_TextEntryKeyMembersDelete()
TeSetKeyText()	GOL_GOL_TextEntrySetKeyText()
TeMsgDefault()	GOL_GOL_TextEntryActionSet()
TeTranslateMsg()	GOL_GOL_TextEntryActionGet()

Header File: [gfx/gfx_gol_window.h](#)

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
WndCreate()	GFX_GOL_WindowCreate()
WndDraw()	GFX_GOL_WindowDraw()
WndGetText()	GFX_GOL_WindowTextGet()
WndSetText()	GFX_GOL_WindowTextSet()
WndTranslateMsg()	GFX_GOL_WindowActionGet()

5.2.2 Graphics Object Library

5.2.2.1 Introduction

Graphics Object Library

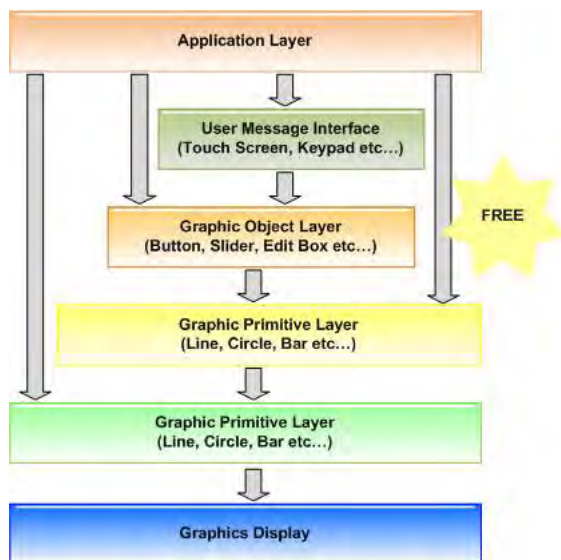
for

Microchip Microcontrollers

This library provides the API of the Graphics Object Library that is available on the Microchip family of microcontrollers with a convenient C language interface.

Description

This document explains how to get started with Microchip Graphics Library solution. It presents the available resources and where to obtain these resources. It also includes the Application Programming Interface (API) of the Microchip Graphics Library.



The Microchip %LibraryName% is highly modular and is optimized for Microchip's microcontrollers. Additionally, the library is free for Microchip customers, easy to use, and has an open documented interface for new driver support, which requires creation of only one C file.

The Microchip %LibraryName% Software %LibraryVersion% supports the following features:

- Configurable graphic resolution
- Up to 24-bit or 65K colors
- 2D objects such as line, circle, text, rectangle, polygon, bar
- 3D objects such as buttons, meter, dial, list box, edit box, check box, radio buttons, window, group box, slider.
- Image, animation
- Variety of user input device such as touch screen, keypad etc...

- Multiple fonts
- Unicode fonts
- PIC24 support, PIC32 support

5.2.2.2 Release Notes

MPLAB Harmony Version: v0.70b **Graphics Object Library Version :** 4.00 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library.

Known Issues:

Nothing to report in this release.

5.2.2.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.2.2.4 Using the Library

This topic describes the basic architecture of the Graphics Object Library and provides information and examples on how to use it.

Interface Header File: gfx.h

The interface to the Graphics Object library is defined in the "gfx.h" header file. This file is included by the "app.h" file. Any C language source (.c) file that uses the Graphics Object library should include "gfx.h".

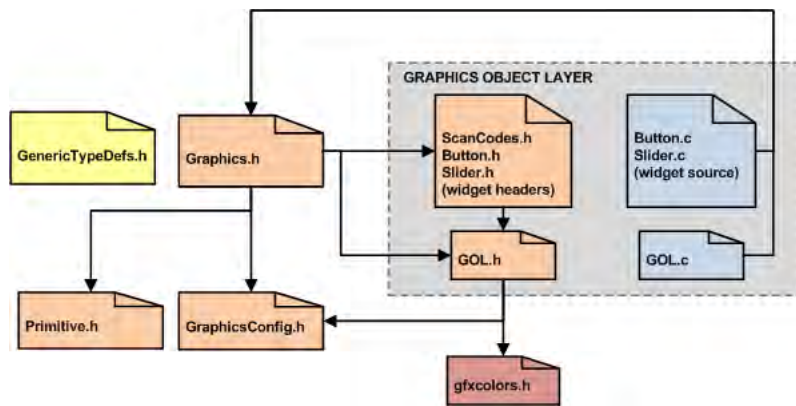
5.2.2.4.1 Abstraction Model

This library provides the API of the Graphics Object Library that is available on the Microchip family of microcontrollers with a convenient C language interface.

Description

Object Layer Software Abstraction Block Diagram

The Graphics Object Layer organization is shown on the figure below:



5.2.2.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Object module.

- General Object Layer Functions
- Analog Clock Object Functions
- Button Object Functions
- Check Box Object Functions
- Chart Object Functions
- Custom Control Object Functions
- Digital Meter Object Functions
- Edit Box Object Functions
- Grid Object Functions
- Group Box Object Functions
- List Box Object Functions
- Meter Object Functions
- Progress Bar Object Functions
- Picture Object Functions
- Radio Button Object Functions
- Round Dial Object Functions
- Scrollbar Object Functions
- Static Text Object Functions
- Text Entry Object Functions
- Window Object Functions
- Draw Functions
- Image Functions

- Palette Functions
- Style Scheme Functions
- Global Variables
- Data Types and Constants

5.2.2.4.3 How the Library Works

5.2.2.4.3.1 Objects Functionality

5.2.2.4.3.1.1 Widget Objects

All the GOL objects that will be shown in the display have a corresponding data structure that keeps and maintains its parameters.

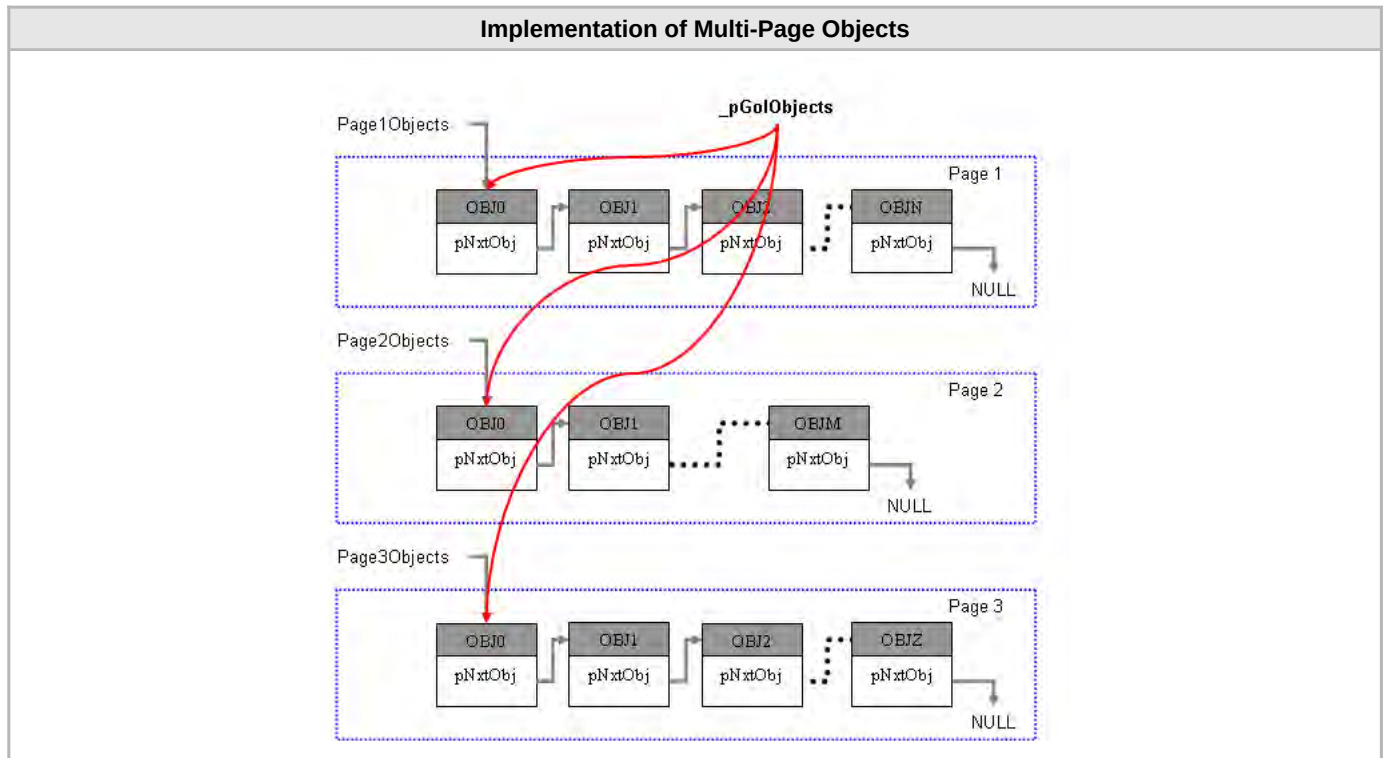
Each object type has a set of functions that enables the user to change the state of the object.

Each GOL object type uses a style scheme. The style scheme defines the font and color settings. User can create new style schemes and assign it to objects. Multiple style schemes can be used for different objects.

To efficiently manage the different objects the first 9 fields of the structure for each object are defined the same. Collectively they are referred to as the object header structure (GFX_OBJ_HEADER). Defining the GFX_OBJ_HEADER enables the common APIs to manage the objects of different types in the same manner.

The GOL operation is centered on two major processes. First is the rendering process and second is the messaging process. These processes make use of linked list objects. The field pNxtObj in GFX_OBJ_HEADER makes this possible. To manage the list a global pointer _pGoObjects is utilized. It defines the current active linked list. Newly created objects are automatically added to the end of this list. The active linked list is the list that the messaging and rendering operation parses to evaluate if the messages received affect the objects and if objects need to be redrawn.

The use of the _pGoObjects pointer also provides a way to easily manage objects. In applications where multiple pages are displayed on the screen, objects can be grouped according to pages. The pointer can be manipulated to switch from one page to another depending on the page currently shown on the screen.



Users can remove an object from a list by pointer manipulation. Objects that are removed from the list are not accessible by the rendering and messaging processes.

5.2.2.4.3.1.2 States

The GOL objects follow two types of states, the Property States and the Drawing States. Property States defines action and appearance of objects. Drawing States on the other hand indicate if the object needs to be hidden, partially redrawn, or fully redrawn in the display. To store the states, the field state defined in GFX_OBJ_HEADER structure is used. Six most significant bits are allocated for Drawing States and the rest is allocated for the Property States. Some common Property States and Drawing States are shown below.

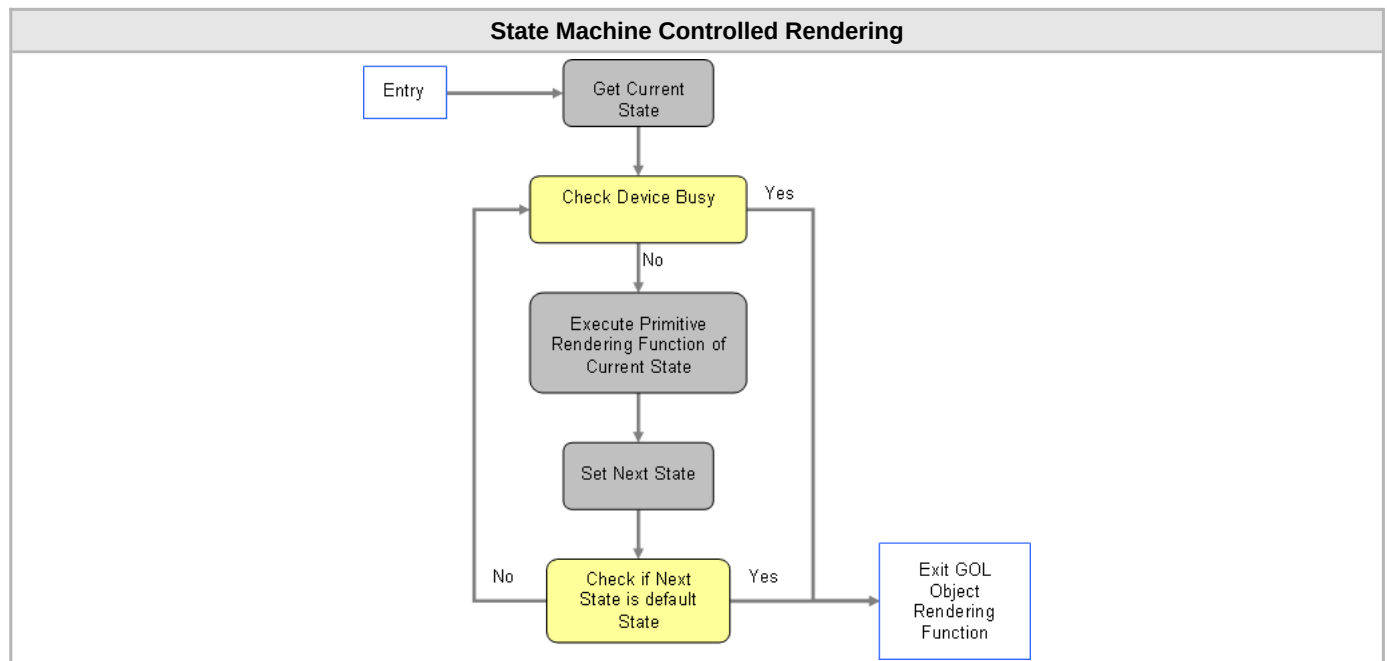
- GFX_OBJ_STATE_FOCUSED
- GFX_OBJ_STATE_DISABLED
- GFX_OBJ_STATE_DRAW_FOCUS
- GFX_OBJ_STATE_DRAW
- GFX_OBJ_STATE_HIDE

Individual Object drawing function (e.g. BtnDraw(), SldDraw(), etc...) does not reset the draw states instead use GFX_OBJ_Draw() to automatically reset and manage the draw states. If the call to individual drawing function cannot be avoided, draw states must be reset manually after the drawing functions returns a 1.

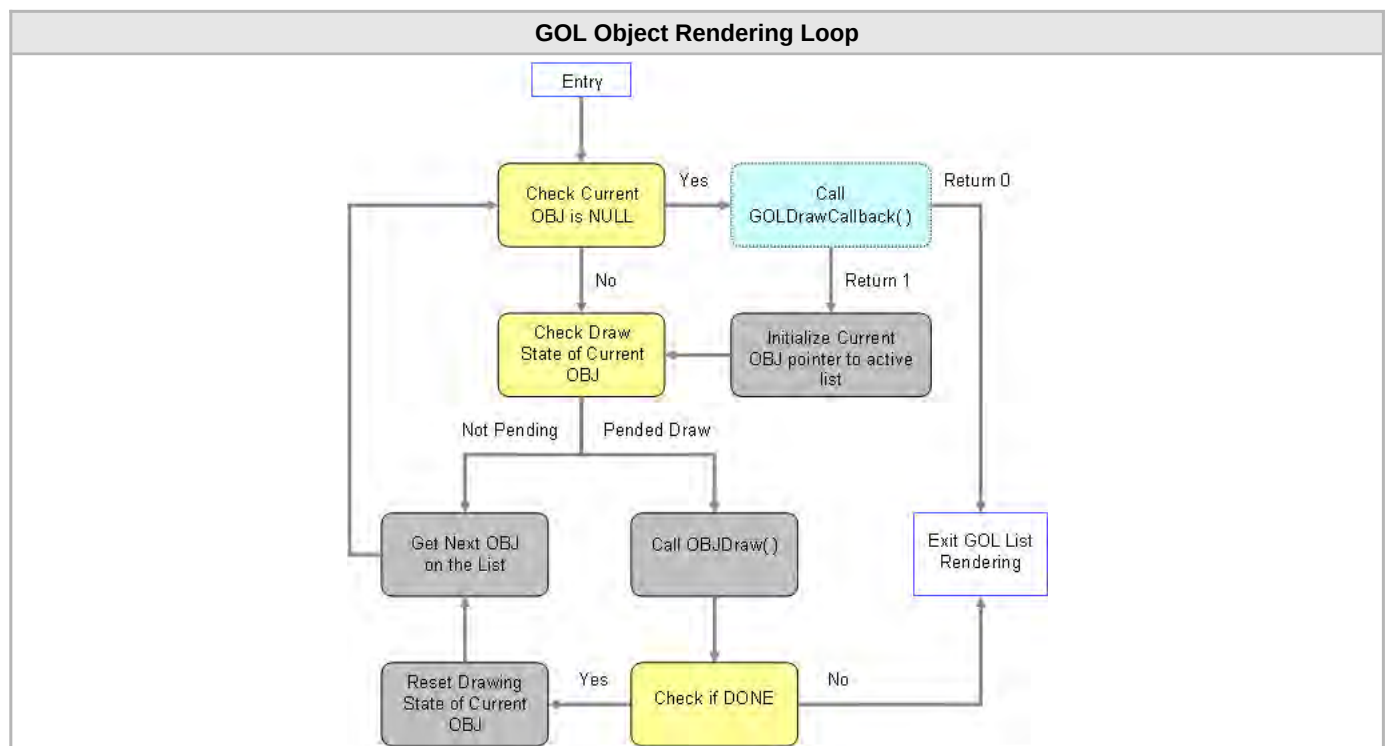
5.2.2.4.3.1.3 Rendering

The library renders objects in a Non-Blocking manner. The Non-Blocking configuration is implemented by the use of drawing state machine. Each drawing functions groups the rendering steps into states. Every time a rendering step is executed, the drawing state is updated. Before each step is executed, the display device is checked if it is still busy with the previous rendering operation. If it is busy it returns a non-zero value. This indicates that the draw function must be called again to complete the

rendering. The drawing function can be called several times until rendering is completed.



The GOL level uses the active object linked list for drawing of objects. Each object's state in the list is parsed to determine if the object needs to be redrawn or not. The drawing order is from the head to the tail of the list. This sequence is executed by `GFX_OBJ_Draw()` function. The figure below explains the rendering loop of the `GFX_OBJ_Draw()` function.



The loop shows two exit points in the sequence. First is when the end of the list is reached and the second is when an `GFX_OBJ_Draw()` returns a NOT DONE status. Reaching the end of the list is a normal exit. This means that all the state

machines of the draw functions of each object have reset to default. Exiting with a NOT DONE status means that the latest executed draw function was pended and the object is not yet fully rendered. To complete the rendering, `GFX_OBJ_Draw()` function should be called again. The next call to `GFX_OBJ_Draw()` will pickup the rendering on the last object that returned a not DONE status. This operation makes the rendering functions non-blocking and gives opportunity to release control to program without waiting for the rendering completion.

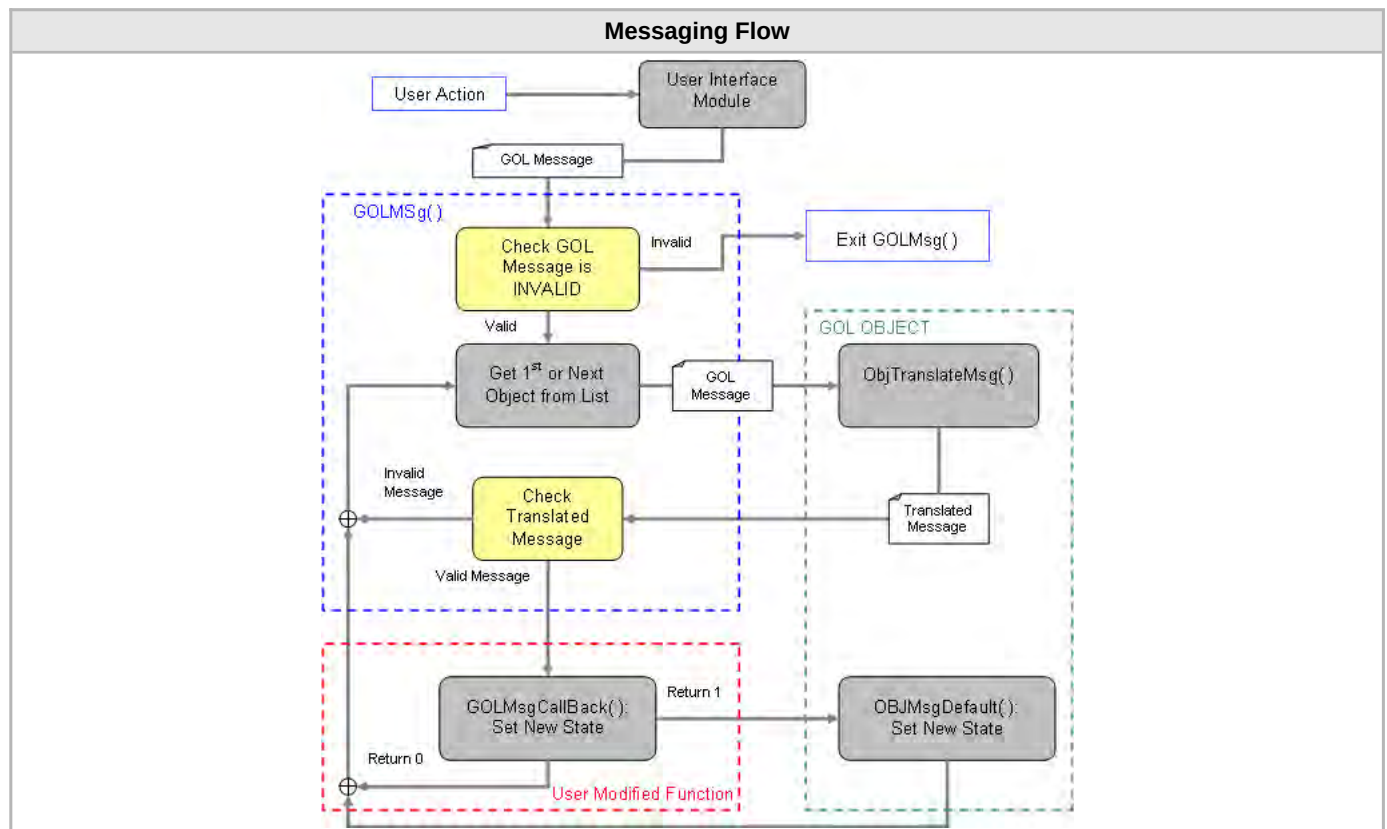
When all objects in the active object linked list are drawn `GFX_OBJ_Draw()` calls user defined `GFX_OBJ_DrawCallback()` function. User drawing can be done in this callback function. If the function returns a zero, drawing of GOL objects in the active list is suspended. In this case color, clipping region, line type and graphic cursor will not be modified by GOL. If it returns a 1 drawing control is returned to GOL. `GFX_OBJ_Draw()` resume rendering of objects in the current active list. Inside the `GFX_OBJ_DrawCallback()` function, the active object list is not used by `GFX_OBJ_Draw()`. It is safe to perform modification of the list. Please refer to Configuration Settings to set Blocking or Non-Blocking configuration.

5.2.2.4.3.2 Messaging Functionality

To facilitate the processing of user actions on the objects, messaging are used.

User passes messages from the input devices to GOL using the GOL message structure. (See: `GFX_OBJ_MESSAGE` for structure information)

`GFX_OBJ_Message()` function accepts this structure and processes the message for all objects in the active list.



The messaging mechanism follows this flow:

1. User Interface Module (touch screen, keypad) sends a GOL message (the `GFX_OBJ_MESSAGE` structure).
2. A loop evaluates which object is affected by the message. This is done inside `GFX_OBJ_Message()` function.
3. Affected object returns the translated message based on the GOL message parameters.

4. User can change default action with the callback function. If the call back function returns a non-zero value message will be processed by default.
5. Object should be redrawn to reflect new state.

The translated message is a set of actions unique to each object type. Please refer to each object translated message ID for details.

Objects that are disabled will not [accept](#) any messages. `GFX_OBJ_Message()` function must be called when GOL drawing is completed. In this case all objects have been drawn and it is safe to change objects states. `GFX_OBJ_Message()` call can be done if `GFX_OBJ_Draw()` function returns non-zero or inside `GFX_OBJ_DrawCallback()` function.

5.2.2.4.3.3 Scheme Functionality

All objects uses a style scheme structure that defines the font and colors used. Upon the object's creation a user defined style scheme can be assigned to the object. In the absence of the user defined scheme, the default scheme is used.

(See: [GFX_GOL_OBJ_SCHEME](#) for structure information)

`TextColorDisabled` and `ColorDisabled` are used when the object is in the disabled state. Otherwise, `TextColor0`, `TextColor1`, `Color0` and `Color1` are used. When object Draw state is set to `GFX_OBJ_STATE_HIDE`, the `CommonBkColor` is used to fill area occupied by object.

Style scheme can be created with `GFX_OBJ_SchemeCreate()` function that returns a pointer to the newly created [GFX_GOL_OBJ_SCHEME](#) structure with default values automatically assigned.

The default values can be changed in the `gfx_gol_scheme_default.c` file.

The application code can define its own default style scheme by defining the macro `GFX_SCHEMEDEFAULT` in the `GraphicsConfig.h`.

Then, [GFX_GOL_OBJ_SCHEME](#) `GFX_OBJECT_SchemeDefault` must be defined in the application code with each structure member initialized to the desired values. See `GOLSchemeDefault.c` file for an example on how to initialize the style scheme.

5.2.2.5 Configuring the Library

The configuration of the Graphics Object Layer is based on the file `system_config.h`

This header file contains the configuration selection for the Graphics Object Layer. Based on the selections made, the Graphics Object Layer will support or not support selected features. These configuration settings will apply to all instances of the Graphics Object Layer.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.2.2.6 Building the Library

This section list the files that are available in the `\src` of the object driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.2.2.7 Library Interface

Data Types and Constants

	Name	Description
	CHARTPARAM	Defines the parameters for the CHART object.
	DATASERIES	Defines a variable for the CHART object. It specifies the number of samples, pointer to the array of samples for the data series and pointer to the next data series. A member of this structure (show) is used as a flag to determine if the series is to be drawn or not. Together with the smplStart and smplEnd it will determine what kind of chart will be drawn.
	FileFeof	This is type FileFeof.
	FileRead	Typedefs of the used File System APIs
	FileSeek	This is type FileSeek.
	FileTell	This is type FileTell.
	FONT_ORIENTATION	Types of orientation
	GFX_ALIGNMENT	Defines the types of text alignment. The text alignment is used in GFX_TextStringBoxDraw() .

5.2.2.7.1 General Object Layer Functions

5.2.2.7.1.1 GFX_GOL_DrawCallbackSet Function

C

```
void GFX_GOL_DrawCallbackSet(
    GFX_GOL_DRAW_CALLBACK_FUNC pFunc
);
```

Description

This function sets the draw callback function that the application will use to call primitive function to implement application specific shapes. See [GFX_GOL_DRAW_CALLBACK_FUNC](#) definition for details on the draw callback function.

Preconditions

None.

Parameters

Parameters	Description
pFunc	pointer to the draw callback function.

Returns

None.

Example

None.

5.2.2.7.1.2 GFX_GOL_MessageCallbackSet Function

C

```
void GFX_GOL_MessageCallbackSet(
    GFX_GOL_MESSAGE_CALLBACK_FUNC pFunc
)
```

```
);
```

Description

This function sets the message callback function that the application will use to evaluate user inputs that affects the objects and application behavior. The callback function location is specified by the function pointer supplied in the call. See GFX_GOL_MESSAGE_CALLBACK_FUNC definition for details on the message callback function.

Preconditions

None.

Parameters

Parameters	Description
pFunc	pointer to the message callback function.

Returns

None.

Example

None.

5.2.2.7.1.3 GFX_GOL_ObjectAdd Function

C

```
void GFX_GOL_ObjectAdd(
    GFX_GOL_OBJ_HEADER * pObject
);
```

Description

This function adds an object to the tail of the currently active list. The new list tail is set to point to NULL after the new object is added.

Preconditions

None.

Returns

None.

Example

```
void MoveObject(    GFX_GOL_OBJ_HEADER *pSrcList,
                   GFX_GOL_OBJ_HEADER *pDstList,
                   GFX_GOL_OBJ_HEADER *pObjtoMove)
{
    GFX_GOL_OBJ_HEADER *pTemp = pSrcList;

    if(pTemp != pObjtoMove)
    {
        while(pTemp->pNxtObj != pObjtoMove)
            pTemp = pTemp->pNxtObj;
    }

    pTemp->pNxtObj = pObjtoMove->pNxt; // remove object from list
    GFX_GOL_ObjectListSet(pDstList); // destination as active list
    GFX_GOL_ObjectAdd(pObjtoMove);   // add object to active list
}
```

5.2.2.7.1.4 GFX_GOL_ObjectBackGroundSet Function

C

```
void GFX_GOL_ObjectBackGroundSet(  
    GFX_GOL_OBJ_HEADER * pObjectHeader  
)  
;
```

Description

This function sets the background information. This is an internal function and should not be called by the application. This function is used by object's drawing functions to set the background information.

Preconditions

None.

Returns

None.

Example

None.

5.2.2.7.1.5 GFX_GOL_ObjectByIDDelete Function

C

```
GFX_STATUS GFX_GOL_ObjectByIDDelete(  
    uint16_t id  
)  
;
```

Description

This function removes an object with the given user defined ID from the currently active list. Aside from the removal of the object from the list, the RAM resources consumed by the object is also freed.

If there is no object with the given ID, the function exits with GFX_STATUS_FAILURE.

Preconditions

None.

Returns

GFX_STATUS_SUCCESS - is returned if the removal was successful. GFX_STATUS_FAILURE - is returned if the removal was not successful.

Example

TODO:Add example code

5.2.2.7.1.6 GFX_GOL_ObjectCanBeFocused Function

C

```
GFX_STATUS GFX_GOL_ObjectCanBeFocused(  
    GFX_GOL_OBJ_HEADER * pObject  
)  
;
```

Description

This function checks if the object can be focused or not. If the object can be focused, it returns GFX_STATUS_SUCCESS. If it cannot be focused, it returns GFX_STATUS_FAILURE. Selected objects have the focus feature. Refer to the object documentation for details.

Objects that do not support focus feature will ignore any focus settings.

If the object is disabled it cannot be set to focused state.

Preconditions

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

GFX_STATUS_SUCCESS - when the object can be focused GFX_STATUS_FAILURE - when the object cannot be focused or do not support the focus feature.

Example

TODO:Add example code here

5.2.2.7.1.7 GFX_GOL_ObjectDelete Function**C**

```
GFX_STATUS GFX_GOL_ObjectDelete(  
    GFX_GOL_OBJ_HEADER * pObject  
);
```

Description

This function removes an object from the currently active list. Aside from the removal of the object from the list, the RAM resources consumed by the object is also freed.

Preconditions

None.

Returns

GFX_STATUS_SUCCESS - is returned if the removal was successful. GFX_STATUS_FAILURE - is returned if the removal was not successful.

Example

TODO:Add example code

5.2.2.7.1.8 GFX_GOL_ObjectDrawDisable Function**C**

```
void GFX_GOL_ObjectDrawDisable(  
    GFX_GOL_OBJ_HEADER * pObject  
);
```

Description

This function resets the drawing state bits of the object. This function can be called to cancel any drawing state bits that has been set or clears all the drawing state bits after the object has been redrawn.

Preconditions

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

None.

5.2.2.7.1.9 GFX_GOL_ObjectDrawEnable Function**C**

```
void GFX_GOL_ObjectDrawEnable(
    GFX_GOL_OBJ_HEADER * pObject
);
```

Description

This function sets the object to be redrawn. For the redraw to be effective, the object must be in the current active list. If not, the redraw action will not be performed until the list where the object is currently inserted will be set as the active list.

Preconditions

None.

Parameters

Parameters	Description
pObject	pointer to the object that will be redrawn.

Returns

None.

Example

```
void GOLRedrawRec(uint16_t left, uint16_t top,
                  uint16_t right, uint16_t bottom)
{
    GFX_GOL_OBJ_HEADER *pCurrentObj;
    int overlapX, overlapY;

    pCurrentObj = _pGolObjects;
    while(pCurrentObj != NULL)
    {
        overlapX = overlapY = 0;

        // check overlaps in x direction
        if (((pCurrentObj->left <= left) && (pCurrentObj->right >= left)) ||
            ((pCurrentObj->left <= right) && (pCurrentObj->right >= right)) ||
            ((pCurrentObj->left <= left) && (pCurrentObj->right >= right)) ||
            ((pCurrentObj->left >= left) && (pCurrentObj->right <= right))
        )
            overlapX = 1;

        // check overlaps in y direction
        if (((pCurrentObj->top <= top) && (pCurrentObj->bottom >= top)) ||
            ((pCurrentObj->top <= bottom) && (pCurrentObj->bottom >= bottom)) ||
            ((pCurrentObj->top <= top) && (pCurrentObj->bottom >= bottom)) ||
            ((pCurrentObj->top >= top) && (pCurrentObj->bottom <= bottom))
        )
            overlapY = 1;
    }
}
```

```

        ((pCurrentObj->top >= top)    && (pCurrentObj->bottom <= bottom))
    )
    overlapY = 1;

    // when any portion of the widget is touched by the defined rectangle the
    // x and y overlaps will exist.
    if (overlapX & overlapY) {
        GFX_GOL_ObjectRedraw(pCurrentObj);
    }

    pCurrentObj = (GFX_GOL_OBJ_HEADER *)pCurrentObj->pNxtObj;
    //end of while
}
}

```

5.2.2.7.1.10 GFX_GOL_ObjectFind Function

C

```

GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectFind(
    uint16_t ID
);

```

Description

This function returns the pointer to object in the list with the user defined ID assigned to it.

Preconditions

None.

Returns

The pointer to the object in the list with the given ID.

Example

```

void CopyObject(GFX_GOL_OBJ_HEADER *pSrcList,
                GFX_GOL_OBJ_HEADER *pDstList,
                uint16_t ID)
{
    GFX_GOL_OBJ_HEADER *pTemp;

    // find the object
    pTemp = GFX_GOL_ObjectFind(ID);

    if (pTemp != NULL)
    {
        // destination as active list
        GFX_GOL_ObjectSetList(pDstList);

        // add object to active list
        GFX_GOL_ObjectAdd(pTemp);
    }
}

```

5.2.2.7.1.11 GFX_GOL_ObjectFocusGet Function

C

```

GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectFocusGet();

```

Description

This function returns the pointer to the object that is currently receiving keyboard input (or focused).

If there are no object that can **accept** keyboard messages, then the function will return NULL.

Objects that can be focused are those objects that can receive keyboard inputs.

Preconditions

None.

Returns

The pointer the currently focused object. Returns NULL if there is no object currently set.

Example

TODO:Add example code here

5.2.2.7.1.12 GFX_GOL_ObjectFocusNextGet Function**C**

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectFocusNextGet();
```

Description

This function returns the pointer to the next object in the active list of objects which can be focused.

The reference point is the currently focused object. If there is no currently focused object, the searched starts from the beginning of the active list of objects.

Objects that can be focused are those objects that can receive keyboard inputs.

If there is no object capable of receiving keyboard inputs (i.e. none can be focused) NULL is returned.

Preconditions

None.

Returns

The pointer to the object that can be focused.

Example

TODO:Add example code here

5.2.2.7.1.13 GFX_GOL_ObjectFocusPrevGet Function**C**

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectFocusPrevGet();
```

Description

This function returns the pointer to the previous object in the active list of objects which can be focused.

The reference point is the currently focused object. If there is no currently focused object, the searched starts from the beginning of the active list of objects.

Objects that can be focused are those objects that can receive keyboard inputs.

If there is no object capable of receiving keyboard inputs (i.e. none can be focused) NULL is returned.

Preconditions

None.

Returns

The pointer to the object that can be focused.

Example

TODO:Add example code here

5.2.2.7.1.14 GFX_GOL_ObjectFocusSet Function

C

```
GFX_STATUS GFX_GOL_ObjectFocusSet(  
    GFX_GOL_OBJ_HEADER * pObject  
);
```

Description

This function sets the specified object to be focused.

If the object cannot [accept](#) keyboard messages, the object will not be set to focused state. If the object can [accept](#) keyboard messages, then the focus state will be set and will be marked to be redrawn to show the focus when the focus feature is enabled.

Objects that can be focused are those objects that can receive keyboard inputs.

Preconditions

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

GFX_STATUS_SUCCESS - when the object can be focused
GFX_STATUS_FAILURE - when the object cannot be focused or do not support the focus feature.

Example

TODO:Add example code here

5.2.2.7.1.15 GFX_GOL_ObjectHideDraw Function

C

```
GFX_STATUS GFX_GOL_ObjectHideDraw(  
    GFX_GOL_OBJ_HEADER * pObject  
);
```

Description

This is function GFX_GOL_ObjectHideDraw.

5.2.2.7.1.16 GFX_GOL_ObjectIDGet Function

C

```
uint16_t GFX_GOL_ObjectIDGet(  
    GFX_GOL_OBJ_HEADER * pObject  
);
```

Description

This function returns the user defined ID assigned to the object.

Preconditions

None.

Returns

The user defined ID of the object.

Example

```

void ExampleUsageOfGettingID(GFX_GOL_OBJ_HEADER *pObject)
{
    uint16_t id;

    switch(id = GFX_GOL_ObjectIDGet(pObject))
    {
        case ID_WINDOW1:
            // do something
        case ID_WINDOW2:
            // do something else
        case ID_WINDOW3:
            // do something else
        default:
            // do something else
    }
}

```

5.2.2.7.1.17 GFX_GOL_ObjectIsRedrawSet Function**C**

```

bool GFX_GOL_ObjectIsRedrawSet(
    GFX_GOL_OBJ_HEADER * pObject
);

```

Description

This function checks if the object needs to be redrawn or not. The function returns true if it is to be redrawn or false if it is not to be redrawn.

Preconditions

None.

Parameters

Parameters	Description
pObject	pointer to the object that will be checked.

Returns

true - when the object needs to be redrawn. false - when the object does not need to be redrawn.

Example

```

int DrawButtonWindowOnly()
{
    static GFX_GOL_OBJ_HEADER *pCurrentObj = NULL;
    uint16_t done = 0;

    if (pCurrentObj == NULL)
    {
        // get current list
        pCurrentObj = GFX_GOL_ObjectListGet();
    }

    while(pCurrentObj != NULL)
    {
        if(GFX_GOL_ObjectIsRedrawSet(pCurrentObj) == true)
        {
            done = pCurrentObj->draw(pCurrentObj);

            // reset state of object if done
            if (done)
                GOLDrawComplete(pCurrentObj)
        }
    }
}

```

```

        // Return if not done. This means that Button Draw function
        // was not able to finish redrawing the object
        // and must be called again to finish rendering of
        // objects in the list that have new states.
    else
        return 0;
    }
    // go to the next object in the list
    pCurrentObj = pCurrentObj->pNxtObj;
}
return 1;
}

```

5.2.2.7.1.18 GFX_GOL_ObjectListDraw Function

C

```
GFX_STATUS GFX_GOL_ObjectListDraw();
```

Description

This function loops through the active list and redraws objects that need to be redrawn. Partial redrawing or full redraw is performed depending on the drawing states of the objects.

GFX_GOL_ObjectDrawCallback() function is called by GFX_GOL_ObjectListDraw() when drawing of objects in the active list is completed. GFX_GOL_ObjectDrawCallback() is an application implemented function that allows the application the opportunity to insert application specific rendering using Primitive Layer rendering functions.

The GFX_GOL_ObjectListDraw() function can return with GFX_STATUS_BUSY. In this case, it indicates that the currently redrawn object is not able to continue. Application needs to call GFX_GOL_ObjectListDraw() again to continue the redraw of the objects in the list.

Preconditions

None.

Returns

GFX_STATUS_SUCCESS - is returned when the active list is completely parsed and redrawn. GFX_STATUS_BUSY - is returned when the active list is not completely parsed and redrawn.

Example

```

// Assume objects are created & states are set to draw objects
while(1)
{
    // parse active list and redraw objects
    // that needs to be redrawn
    if( GFX_GOL_ObjectListDraw() == GFX_STATUS_SUCCESS)
    {
        // at this point drawing is completed
        // it is safe to modify objects states and linked list

        // evaluate messages from touch screen device
        TouchGetMsg(&msg);

        // evaluate each object is affected by the message
        GFX_GOL_ObjectMessage(&msg);
    }
}

```

5.2.2.7.1.19 GFX_GOL_ObjectListFree Function

C

```
GFX_STATUS GFX_GOL_ObjectListFree();
```

Description

This function frees all the memory used by objects in the active list and initializes the active list pointer to NULL to start a new empty list. This function must be called only inside the `GFX_GOL_ObjectDrawCallback()` function when using `GFX_GOL_ObjectListDraw()` and `GFX_GOL_ObjectMessage()` functions. This requirement assures that primitive rendering settings are not altered by the rendering state machines of the objects.

Preconditions

None.

Returns

`GFX_STATUS_SUCCESS` - is returned if the free was successful. `GFX_STATUS_FAILURE` - is returned if the free was not successful.

Example

```
void DeletePage(GFX_GOL_OBJ_HEADER *pPage)
{
    GFX_GOL_OBJ_HEADER *pTemp;

    // assuming pPage is different from the current active list
    // save the active list
    pTemp = GFX_GOL_ObjectListGet();

    // set list as active list
    GFX_GOL_ObjectListSet(pPage);

    // pPage objects are deleted
    GFX_GOL_ObjectListFree();

    // restore the active list
    GFX_GOL_ObjectListSet(pTemp);
}
```

5.2.2.7.1.20 GFX_GOL_ObjectListGet Function**C**

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectListGet();
```

Description

This function returns the pointer to the current active.

Preconditions

None.

Returns

Pointer (type `GFX_GOL_OBJ_HEADER`) to the current active list.

Example

See `GFX_GOL_ObjectListNew()` for example code.

5.2.2.7.1.21 GFX_GOL_ObjectListNew Function**C**

```
GFX_STATUS GFX_GOL_ObjectListNew();
```

Description

This function starts a new linked list of objects and resets the keyboard focus to none. This function assigns the current active list

and current focused object (receiving keyboard inputs) object pointers to NULL. Any keyboard inputs at this point will be ignored.

This function does not erase the objects in the previous list. Application must save the previous list to another pointer if to be referenced later. If not needed anymore, memory used by that list should be freed by [GFX_GOL_ObjectListFree\(\)](#) function. In this case, freeing the list with [GFX_GOL_ObjectListFree\(\)](#) function has the same effect as [GFX_GOL_ObjectListNew\(\)](#) where the current active list is empty.

Preconditions

None.

Returns

GFX_STATUS_SUCCESS - is returned if the new list start was successful. GFX_STATUS_FAILURE - is returned if the new list start was not successful.

Example

```
// assume pointers to objects (pButton, pWindow and pSlider
// are initialized to objects already created
// GFX_GOL_OBJ_HEADER *pButton;
// GFX_GOL_OBJ_HEADER *pWindow;
// GFX_GOL_OBJ_HEADER *pSlider;

GFX_GOL_OBJ_HEADER *pSave;

// save current list
pSave = GFX_GOL_ObjectListGet();

// start the new list, after the start of the list, the
// current active list is empty.
GFX_GOL_ObjectListNew();

// assume that objects are already created
// you can now add objects to the new list
GFX_GOL_ObjectAdd(pButton);
GFX_GOL_ObjectAdd(pWindow);
GFX_GOL_ObjectAdd(pSlider);
```

5.2.2.7.1.22 GFX_GOL_ObjectListSet Function

C

```
GFX_STATUS GFX_GOL_ObjectListSet(
    GFX_GOL_OBJ_HEADER * pList
);
```

Description

This function sets the active list to the new list. The previous list will still exist in memory. Application must save the previous list before the set is called if the previous list will be referenced later. If the previous list is not needed anymore, then the list must be removed from memory by [GFX_GOL_ObjectListFree\(\)](#) function.

Setting the active list to the new list will reset the focused pointer object to NULL.

Preconditions

None.

Returns

GFX_STATUS_SUCCESS - is returned if the set was successful. GFX_STATUS_FAILURE - is returned if the set was not successful.

Example

```
GFX_GOL_OBJ_HEADER *pSave;
```

```

// save current list
pSave = GFX_GOL_ObjectListSet();

// start the new list
GFX_GOL_ObjectListNew();

// you can now add objects to the current list
// assume that objects are already created
GFX_GOL_ObjectAdd(pButton);
GFX_GOL_ObjectAdd(pWindow);
GFX_GOL_ObjectAdd(pSlider);

// do something here on the new list

// return the old list
GOLSetList(pSave);

```

5.2.2.7.1.23 GFX_GOL_ObjectMessage Function

C

```

void GFX_GOL_ObjectMessage(
    GFX_GOL_MESSAGE * pMsg
);

```

Description

This function receives a [GFX_GOL_MESSAGE](#) message from user and loops through the active list of objects to check which object is affected by the message. For affected objects the message is translated and `GFX_GOL_ObjectMessageCallback()` is called. In the call back function, user has the ability to implement action for the message. If the call back function returns non-zero, `OBJMsgDefault()` is called to process message for the object by default. If zero is returned `OBJMsgDefault()` is not called. Please refer to GOL Messages section for details.

Preconditions

None.

Parameters

Parameters	Description
pMessage	Pointer to the message from user.

Returns

None.

Example

```

// Assume objects are created & states are set to draw objects
while (1)
{
    if(GOLDDraw())
    {
        // GOL drawing is completed here
        // it is safe to change objects

        // from user interface module
        TouchGetMsg(&msg);
        // process the message
        GFX_GOL_ObjectMessage(&msg);
    }
}

```

5.2.2.7.1.24 GFX_GOL_ObjectNextGet Function

C

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectNextGet(
    GFX_GOL_OBJ_HEADER * pObject
);
```

Description

This function returns the pointer to next object in the list after the specified object.

Preconditions

None.

Returns

The pointer to the next object in the list.

Example

```
void RedrawButtons(void)
{
    GFX_GOL_OBJ_HEADER *pCurr;

    // get active list
    pCurr = GFX_GOL_ObjectListGet();

    while(pCurr->pNxtObj != NULL)
    {
        // just select button objects and redraw them
        if (GFX_GOL_ObjectTypeGet(pCurr) == BUTTON)
        {
            // set to be redrawn
            pCurr->state = BTN_DRAW;
        }
        pCurr = GFX_GOL_ObjectNextGet(pCurr);
    }
    // redraw all buttons in the active list
    GFX_GOL_ObjectListDraw();
}
```

5.2.2.7.1.25 GFX_GOL_ObjectRectangleRedraw Function

C

```
void GFX_GOL_ObjectRectangleRedraw(
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom
);
```

Description

This function marks all objects in the active list intersected by the given rectangular area to be redrawn.

After calling this function, the next call to [GFX_GOL_ObjectListDraw\(\)](#) will redraw all objects that are marked for redraw.

Preconditions

None.

Parameters

Parameters	Description
left	Defines the left most border of the rectangle area.
top	Defines the top most border of the rectangle area.
right	Defines the right most border of the rectangle area.
bottom	Defines the bottom most border of the rectangle area.

Returns

None.

Example

```

GFX_GOL_OBJ_HEADER *pTemp;
GFX_GOL_OBJ_HEADER *pAllObjects;

// assume *pAllObjects points to a list of all existing objects
// created and initialized

// mark all objects inside the rectangle to be redrawn
GOLRedrawRec(10,10,100,100);

// save the current active list
pTemp = pAllObjects;

// reset active list
GFX_GOL_ObjectListNew();

// build the new active list with only those objects that
// are marked to be redrawn
while(pTemp->pNxtObj != NULL)
{
    if (pTemp->state&0x7C00)
        GFX_GOL_ObjectAdd(pTemp);

    pTemp = pTemp->pNxtObj;
}

// redraw active list
GFX_GOL_ObjectListDraw();

```

5.2.2.7.1.26 GFX_GOL_ObjectTypeGet Function**C**

```

GFX_GOL_OBJ_TYPE GFX_GOL_ObjectTypeGet(
    GFX_GOL_OBJ_HEADER * pObject
);

```

Description

This function returns the object type. The object type is one of the defined enumerated types of [GFX_GOL_OBJ_TYPE](#).

Preconditions

None.

Returns

The type of the object. The type is one of the defined enumerated types of [GFX_GOL_OBJ_TYPE](#).

Example

TODO:Add example code here

5.2.2.7.1.27 GFX_GOL_PanelAlphaParameterSet Function

C

```
void GFX_GOL_PanelAlphaParameterSet(  
    uint16_t alphaValue  
)  
;
```

Description

This is function GFX_GOL_PanelAlphaParameterSet.

5.2.2.7.1.28 GFX_GOL_PanelBackgroundSet Function

C

```
void GFX_GOL_PanelBackgroundSet(  
    GFX_GOL_OBJ_HEADER * pObjectHeader  
)  
;
```

Description

This is function GFX_GOL_PanelBackgroundSet.

5.2.2.7.1.29 GFX_GOL_PanelDraw Function

C

```
GFX_STATUS GFX_GOL_PanelDraw();
```

Description

This is function GFX_GOL_PanelDraw.

5.2.2.7.1.30 GFX_GOL_PanelGradientParameterSet Function

C

```
void GFX_GOL_PanelGradientParameterSet(  
    GFX_COLOR startColor,  
    GFX_COLOR endColor  
)  
;
```

Description

This is function GFX_GOL_PanelGradientParameterSet.

5.2.2.7.1.31 GFX_GOL_PanelParameterSet Function

C

```
void GFX_GOL_PanelParameterSet(  
    uint16_t left,  
    uint16_t top,  
    uint16_t right,  
    uint16_t bottom,  
    uint16_t radius,  
    GFX_COLOR faceClr,  
    GFX_COLOR embossLtClr,  
    GFX_COLOR embossDkClr,  
    GFX_RESOURCE_HDR * pBitmap,  
    GFX_FILL_STYLE fillStyle,  
    uint16_t embossSize  
)  
;
```

Description

This is function GFX_GOL_PanelParameterSet.

5.2.2.7.1.32 GFX_GOL_TwoTonePanelDraw Function**C**

```
GFX_STATUS GFX_GOL_TwoTonePanelDraw();
```

Description

This is function GFX_GOL_TwoTonePanelDraw.

5.2.2.7.2 Analog Clock Object Functions**5.2.2.7.2.1 GFX_AnalogClockCreate Function****C**

```
GFX_GOL_ANALOGCLOCK * GFX_AnalogClockCreate(
    uint16_t ID,
    short left,
    short top,
    short right,
    short bottom,
    short hour,
    short minute,
    short radius,
    bool sechand,
    uint16_t state,
    void * pBitmap,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a analog clock at the given position using state, radius, pBitmap, pText, and pScheme.

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
hour	Initial hour value.
minute	Initial minute value.
radius	Sets the radius of the clock.
sechand	Flag to indicate if the second hand will be drawn or not.
state	Sets the initial state of the object.
pBitmap	Pointer to the bitmap used on the face of the analog clock dimension of the image must match the dimension of the widget.
pScheme	Pointer to the style scheme used

Returns

Returns a pointer to the digital meter.

Example

```
GFX_GOL_SCHEME *pScheme;
uint16_t state;

pScheme = GFX_GOL_SCHEMECreate();
state = AC_DRAW;

AnalogClock = GFX_AnalogClock_Create(ANALOGCLOCK_ID, 20, 64, 50, 118,
1, 20, 30, FALSE, state, NULL, pScheme);
```

Remarks:

5.2.2.7.2.2 GFX_GOL_AnalogClockDraw Function**C**

```
GFX_STATUS GFX_GOL_AnalogClockDraw(
    void * pAc
);
```

Description

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

Preconditions

Object must be created before this function is called.

Parameters

Parameters	Description
pAc	Pointer to the object to be rendered.

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Side Effects

none

Example

```
void MyGOLDraw(){
    static GFX_OBJ_HEADER *pCurrentObj = NULL;
    int done;

    // There are no objects
    if(GFX_OBJ_ListGet() == NULL)
        return;

    // If it's last object jump to head
    if(pCurrentObj == NULL)
        pCurrentObj = GFX_OBJ_ListGet();
```

```

done = 0;

// this only process Button and Window
while(pCurrentObj != NULL){
    // check if object state indicates redrawing
    if(pCurrentObj->state&0xFC00) {
        switch(pCurrentObj->type){
            case OBJ_ANALOGCLOCK:
                done = AcDraw((ANALOGCLOCK*)pCurrentObj);
                break;
            case OBJ_WINDOW:
                done = WndDraw((WINDOW*)pCurrentObj);
                break;
            default:
                done = 1;
                break;
        }
        if(done){
            // reset only the state if drawing was finished
            pCurrentObj->state = 0;
        }else{
            // done processing the list
            return;
        }
    }
    // go to next object
    pCurrentObj = pCurrentObj->pNxtObj;
}
}

```

5.2.2.7.2.3 GFX_GOL_AnalogClockHandsDraw Function

C

```

GFX_STATUS GFX_GOL_AnalogClockHandsDraw(
    GFX_GOL_ANALOGCLOCK * pAc,
    short hand,
    short thickness,
    GFX_COLOR color
);

```

Description

Draws the current position of the clock hands with the given thickness and color.

Preconditions

none

Parameters

Parameters	Description
pAc	The pointer to the object whose hands will be modified.
hand	which hand to be drawn (second, minute, hour)
thickness	thickness to draw the hand
color	color to draw the hand
*pBitmap	bitmap background to be redrawn

Returns

none

Side Effects

none

5.2.2.7.2.4 GFX_GOL_AnalogClockHourSet Function

C

```
void GFX_GOL_AnalogClockHourSet(  
    GFX_GOL_ANALOGCLOCK * pAc,  
    short hour  
);
```

Description

Sets the hour value of the analog clock.

Preconditions

none

Parameters

Parameters	Description
pAc	The pointer to the object whose hands will be modified.
hour	New hour time.

Returns

none

Side Effects

none

5.2.2.7.2.5 GFX_GOL_AnalogClockMinuteSet Function

C

```
void GFX_GOL_AnalogClockMinuteSet(  
    GFX_GOL_ANALOGCLOCK * pAc,  
    short minute  
);
```

Description

Sets the minute value of the analog clock.

Preconditions

none

Parameters

Parameters	Description
pAc	The pointer to the object whose hands will be modified.
minute	New minute time.

Returns

none

Side Effects

none

5.2.2.7.2.6 GFX_GOL_AnalogClockSecondSet Function

C

```
void GFX_GOL_AnalogClockSecondSet(
    GFX_GOL_ANALOGCLOCK * pAc,
    short second
);
```

Description

Sets the second value of the analog clock.

Preconditions

none

Parameters

Parameters	Description
pAc	The pointer to the object whose hands will be modified.
second	New second time.

Returns

none

Side Effects

none

5.2.2.7.3 Button Object Functions

5.2.2.7.3.1 GFX_GOL_ButtonActionGet Function

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_ButtonActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_BUTTON_ACTION_PRESSED	Touch Screen	EVENT_PRESS, EVENT_MOVE	If events occurs and the x,y position falls in the face of the GFX_GOL_BUTTON while the GFX_GOL_BUTTON is not pressed.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED or SCAN_SPACE_PRESSED while the GFX_GOL_BUTTON is not pressed.

GFX_GOL_BUTTON_ACTION_STILLPRESSED	Touch Screen	EVENT_STILLPRESS	If event occurs and the x,y position does not change from the previous press position in the face of the GFX_GOL_BUTTON .
GFX_GOL_BUTTON_ACTION_RELEASED	Touch Screen	EVENT_RELEASE	If the event occurs and the x,y position falls in the face of the GFX_GOL_BUTTON while the GFX_GOL_BUTTON is pressed.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_RELEASED or SCAN_SPACE_RELEASED while the GFX_GOL_BUTTON is pressed.
GFX_GOL_BUTTON_ACTION_CANCELPRESS	Touch Screen	EVENT_MOVE	If the event occurs outside the face of the GFX_GOL_BUTTON and the GFX_GOL_BUTTON is currently pressed.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_BUTTON_ACTION_PRESSED â€œ Object is pressed
- GFX_GOL_BUTTON_ACTION_RELEASED â€œ Object is released
- GFX_GOL_BUTTON_ACTION_CANCELPRESS â€œ Object will be released, user cancels press action on the [GFX_GOL_BUTTON](#)
- GFX_GOL_OBJECT_ACTION_INVALID â€œ Object is not affected

Example

None.

5.2.2.7.3.2 GFX_GOL_ButtonActionSet Function**C**

```
void GFX_GOL_ButtonActionSet(
    GFX_GOL_TRANSLATED_ACTION translatedMsg,
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_BUTTON_ACTION_PRESSED	Touch Screen,	Set GFX_GOL_BUTTON_PRESSED_STATE	Button will be redrawn in the pressed state.
	Keyboard		
GFX_GOL_BUTTON_ACTION_RELEASED	Touch Screen,	Clear GFX_GOL_BUTTON_PRESSED_STATE	Button will be redrawn in the released state.
	Keyboard		
GFX_GOL_BUTTON_ACTION_CANCELPRESS	Touch Screen,	Clear GFX_GOL_BUTTON_PRESSED_STATE	Button will be redrawn in the released state.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Returns

None.

Example

None.

5.2.2.7.3.3 GFX_GOL_ButtonCreate Function**C**

```
GFX_GOL_BUTTON * GFX_GOL_ButtonCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t radius,
    uint16_t state,
    GFX_RESOURCE_HDR * pPressImage,
    GFX_RESOURCE_HDR * pReleaseImage,
    GFX_XCHAR * pText,
    GFX_ALIGNMENT alignment,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GFX_GOL_BUTTON](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The object allows setting two images. One for the pressed state and the other for the release state. If no image is to be used for the object set both pointers to NULL.

If only one image is used for both pressed and released state, set both pPressImage and pReleaseImage to the same image.

The behavior of GFX_GOL_ButtonCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- pPressImage and pReleaseImage is not pointing to a [GFX_RESOURCE_HDR](#).
- pText is an unterminated string

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
radius	Radius of the rounded edge. When using gradient buttons and radius != 0, emboss size <= radius. If this is not met, the the GFX_GOL_BUTTON face will not have gradient effect.
state	Sets the initial state of the object.
pPressImage	Pointer to the image used on the face of the object when it is in the pressed state.
pReleaseImage	Pointer to the image used on the face of the object when it is in the pressed state.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_BUTTON *buttons[3];
GFX_GOL_BUTTON_STATE state;

// assume pScheme is initialized to a scheme in memory.
state = GFX_GOL_BUTTON_DRAW_STATE;

buttons[0] = GFX_GOL_ButtonCreate(
    1, 20, 64, 50, 118, 0,
    state, NULL, NULL, "ON",
    GFX_ALIGN_HCENTER | GFX_ALIGN_VCENTER,
    pScheme);
// check if GFX_GOL_BUTTON 0 is created
if (buttons[0] == NULL)
    return 0;

buttons[1] = GFX_GOL_ButtonCreate(
    2, 52, 64, 82, 118, 0,
    state, NULL, NULL, "OFF",
    GFX_ALIGN_LEFT | GFX_ALIGN_VCENTER,
    pScheme);
// check if GFX_GOL_BUTTON 1 is created
```

```
if (buttons[1] == NULL)
    return 0;

buttons[2] = GFX_GOL_ButtonCreate(
    3, 84, 64, 114, 118, 0,
    state, NULL, NULL, "HI",
    GFX_ALIGN_RIGHT | GFX_ALIGN_VCENTER,
    pScheme);
// check if GFX_GOL_BUTTON 2 is created
if (buttons[2] == NULL)
    return 0;

return 1;
```

5.2.2.7.3.4 GFX_GOL_ButtonDraw Function

C

```
GFX_STATUS GFX_GOL_ButtonDraw(
    void * pObject
);
```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the [GFX_GOL_BUTTON](#) is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.3.5 GFX_GOL_ButtonTextAlignmentGet Function

C

```
GFX_ALIGNMENT GFX_GOL_ButtonTextAlignmentGet(
    GFX_GOL_BUTTON * pObject
);
```

Description

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The text alignment set in the object. See [GFX_ALIGNMENT](#) for more details.

Example

None.

5.2.2.7.3.6 GFX_GOL_ButtonTextAlignmentSet Function**C**

```
void GFX_GOL_ButtonTextAlignmentSet(  
    GFX_GOL_BUTTON * pObject,  
    GFX_ALIGNMENT align  
);
```

Description

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Returns

None.

Example

None.

5.2.2.7.3.7 GFX_GOL_ButtonTextSet Function**C**

```
void GFX_GOL_ButtonTextSet(  
    GFX_GOL_BUTTON * pObject,  
    GFX_XCHAR * pText  
);
```

Description

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Returns

None.

Example

```
GFX_XCHAR Label0[] = "ON";
GFX_XCHAR Label1[] = "OFF";
GFX_GOL_BUTTON GFX_GOL_BUTTON[2];

GFX_GOL_ButtonTextSet(GFX_GOL_BUTTON[0], Label0);
GFX_GOL_ButtonTextSet(GFX_GOL_BUTTON[1], Label1);
```

5.2.2.7.4 Check Box Object Functions

5.2.2.7.4.1 GFX_GOL_CheckBoxActionGet Function

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_CheckBoxActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_CHECKBOX_ACTION_CHECKED	Touch Screen	EVENT_RELEASE	If events occurs and the x,y position falls in the area of the check box while the check box is unchecked.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED or SCAN_SPACE_PRESSED while the check box is unchecked.
GFX_GOL_CHECKBOX_ACTION_UNCHECKED	Touch Screen	EVENT_RELEASE	If events occurs and the x,y position falls in the area of the check box while the check box is checked.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED or SCAN_SPACE_PRESSED while the check box is checked.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_CHECKBOX_ACTION_CHECKED - Check box is checked
- GFX_GOL_CHECKBOX_ACTION_UNCHECKED - Check box is unchecked
- GFX_GOL_OBJECT_ACTION_INVALID - Object is not affected

Example

None.

5.2.2.7.4.2 GFX_GOL_CheckBoxActionSet Function**C**

```
void GFX_GOL_CheckBoxActionSet(
    GFX_GOL_TRANSLATED_ACTION translatedMsg,
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_CHECKBOX_ACTION_CHECKED	Touch Screen,	Set GFX_GOL_CHECKBOX_CHECKED_STATE	Check Box will be redrawn in checked state.
	Keyboard		
GFX_GOL_CHECKBOX_ACTION_UNCHECKED	Touch Screen,	Clear GFX_GOL_CHECKBOX_CHECKED_STATE	Check Box will be redrawn in un-checked state.
	Keyboard		

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Returns

None.

Example

None.

5.2.2.7.4.3 GFX_GOL_CheckBoxCreate Function**C**

```
GFX_GOL_CHECKBOX * GFX_GOL_CheckBoxCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    GFX_XCHAR * pText,
    GFX_ALIGNMENT alignment,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GFX_GOL_CHECKBOX](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of `GFX_GOL_CheckBoxCreate()` will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- pText is an unterminated string

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.

pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_CHECKBOX *pCb[2];

pCb[0] = GFX_GOL_CheckBoxCreate(
    ID_CHECKBOX1,           // ID
    20,135,150,175,        // dimension
    GFX_GOL_CHECKBOX_DRAW_STATE, // Draw the object
    "Scale",               // text
    GFX_ALIGN_CENTER,      // text alignment
    pScheme);              // use this scheme

pCb[1] = GFX_GOL_CheckBoxCreate(
    ID_CHECKBOX2,           // ID
    170,135,300,175,       // dimension
    GFX_GOL_CHECKBOX_DRAW_STATE, // Draw the object
    "Animate",             // text
    GFX_ALIGN_LEFT | GFX_ALIGN_VCENTER, // text alignment
    pScheme);              // use this scheme
```

5.2.2.7.4.4 GFX_GOL_CheckBoxDraw Function**C**

```
GFX_STATUS GFX_GOL_CheckBoxDraw(
    void * pobject
);
```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the [GFX_GOL_CHECKBOX](#) is drawn with the text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.4.5 GFX_GOL_CheckBoxTextAlignmentGet Function

C

```
GFX_ALIGNMENT GFX_GOL_CheckBoxTextAlignmentGet(  
    GFX_GOL_CHECKBOX * pObject  
);
```

Description

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The text alignment set in the object. See [GFX_ALIGNMENT](#) for more details.

Example

None.

5.2.2.7.4.6 GFX_GOL_CheckBoxTextAlignmentSet Function

C

```
void GFX_GOL_CheckBoxTextAlignmentSet(  
    GFX_GOL_CHECKBOX * pObject,  
    GFX_ALIGNMENT align  
);
```

Description

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Returns

None.

Example

None.

5.2.2.7.4.7 GFX_GOL_CheckBoxTextSet Function

C

```
void GFX_GOL_CheckBoxTextSet(  

```

```

    GFX_GOL_CHECKBOX * pObj,
    GFX_XCHAR * pText
);

```

Description

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Returns

None.

Example

```

GFX_XCHAR Label0[] = "Enabled";
GFX_XCHAR Label1[] = "Disabled";
GFX_GOL_CHECKBOX GFX_GOL_CHECKBOX[2];

GFX_GOL_CheckBoxTextSet(GFX_GOL_CHECKBOX[0], Label0);
GFX_GOL_CheckBoxTextSet(GFX_GOL_CHECKBOX[1], Label1);

```

5.2.2.7.5 Chart Object Functions

5.2.2.7.5.1 GFX_GOL_ChartActionGet Function

C

```

uint16_t GFX_GOL_ChartActionGet(
    void * pObj,
    GFX_GOL_MESSAGE * pMsg
);

```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
CH_MSG_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the area of the chart.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Preconditions

none

Parameters

Parameters	Description
pCh	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMsg	Pointer to the message struct containing the message from the user interface.

Returns

Returns the translated message depending on the received GOL message:

- CH_MSG_SELECTED – Chart area is selected
- OBJ_MSG_INVALID – Chart is not affected

none.

Side Effects

none

5.2.2.7.5.2 GFX_GOL_ChartCreate Function**C**

```
GFX_GOL_CHART * GFX_GOL_ChartCreate(
    uint16_t ID,
    short left,
    short top,
    short right,
    short bottom,
    uint16_t state,
    DATASERIES * pData,
    CHARTPARAM * pParam,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a CHART object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Preconditions

none

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pData	Pointer to the data for the contents of the chart. NULL can be assigned initially when creating the object.
pParam	Pointer to the chart parameters. NULL can be assigned initially when creating the object and the chart parameters can be populated using the API provided.
pScheme	Pointer to the style scheme used. When set to NULL, the default style scheme will be used.

Returns

Returns the pointer to the object created.

Side Effects

none

Example

```
extern const FONT_FLASH GOLSmallFont;
extern const FONT_FLASH GOLMediumFont;

// Note that strings are declared as such to cover cases
// where XCHAR type is declared as short (2 bytes).
XCHAR ChTitle[] = {'E', 'x', 'a', 'm', 'p', 'l', 'e', 0};
XCHAR SampleName[] = {'C', 'a', 't', 'e', 'g', 'o', 'r', 'y', 0};
XCHAR ValueName[] = {'#', 'H', 'i', 't', 's', 0};
XCHAR SeriesName[2] = {
    {'V', '1', 0},
    {'V', '2', 0},
};

V1Data[2] = { 50, 100};
V2Data[2] = { 5, 10};

GFX_OBJ_SCHEME *pScheme;
CHART *pChart;
CHARTPARAM Contents;
uint16_t state;

pScheme = GFX_OBJ_SchemeCreate();
state = CH_BAR|CH_DRAW|CH_BAR_HOR; // GFX_PRIM_BarDraw Chart to be drawn with horizontal
orientation

pChart = ChCreate(0x01, // ID
    0, 0, // dimensions
    GFX_PRIM_GetMaxX(),
    GFX_PRIM_GetMaxY(),
    state, // state of the chart
    NULL, // data not initialized yet
    NULL, // no parameters yet
    pScheme); // style scheme used

if (pMyChart == NULL) // check if chart was allocated memory
    return 0;

ChSetTitleFont(pChart, (void*)&GOLMediumFont);
ChSetTitle(pChart, ChTitle); // set the title

// set the grid labels and axis labels font
ChSetGridLabelFont(pChart, (void*)&GOLSmallFont);
ChSetAxisLabelFont(pChart, (void*)&GOLSmallFont);

// set the labels for the X and Y axis
ChSetSampleLabel(pChart, (XCHAR*)SampleName);
ChSetValueLabel(pChart, (XCHAR*)ValueName);

ChAddDataSeries(pChart, 2, V1Data, (XCHAR*)SeriesName[0]);
ChAddDataSeries(pChart, 2, V2Data, (XCHAR*)SeriesName[1]);

// set the range of the sample values
ChSetValueRange(pChart, 0, 100);

// show all two samples to be displayed (start = 1, end = 2)
ChSetSampleRange(pChart, 1, 2);
```

```
GFX_OBJ_Draw();           // draw the chart
```

5.2.2.7.5.3 GFX_GOL_ChartDataSeriesAdd Function

C

```
DATASERIES * GFX_GOL_ChartDataSeriesAdd(
    GFX_GOL_CHART * pCh,
    uint16_t nSamples,
    uint16_t * pData,
    XCHAR * pName
);
```

Description

This function creates a [DATASERIES](#) object and populates the structure with the given parameters.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.
nSamples	The number of samples or data points.
pData	Pointer to the array of samples or data points.
pName	Pointer to the string used to label the data series.

Returns

Returns the pointer to the data variable ([DATASERIES](#)) object created. If NULL is returned, the addition of the new object failed due to not enough memory for the object.

Side Effects

Appends to the list of [DATASERIES](#) that the chart is operating on. By default, the show flag of the newly added data series is set to [SHOW_DATA](#) or enabled.

Example

See ChCreate() example.

5.2.2.7.5.4 GFX_GOL_ChartDataSeriesFree Function

C

```
void GFX_GOL_ChartDataSeriesFree(
    void * pObjj
);
```

Description

This function removes [DATASERIES](#) object from the list of [DATASERIES](#) objects and frees the memory used of that removed object.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.

Returns

none.

Side Effects

none.

Example

```
void ClearChartData(CHART *pCh) {
    if(pCh->pChData != NULL)
        // remove the all data series
        ChFreeDataSeries(pCh;
}
```

5.2.2.7.5.5 GFX_GOL_ChartDataSeriesRemove Function

C

```
void GFX_GOL_ChartDataSeriesRemove(
    GFX_GOL_CHART * pCh,
    uint16_t number
);
```

Description

This function removes DATASERIES object from the list of DATASERIES objects and frees the memory used of that removed object. The position of the object to be removed is specified by the number parameter. If the list has only one member, it removes the member regardless of the number given.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.
number	The position of the object to be removed in the list where the first object in the list is assigned a value of 1. If this parameter is set to zero, the whole list of DATA_SERIES is removed.

Returns

none.

Side Effects

none.

Example

```
void ClearChartData(CHART *pCh) {
    if(pCh->pChData != NULL)
        // remove the all data series
        ChRemoveDataSeries(pCh, 0);
}
```

5.2.2.7.5.6 GFX_GOL_ChartDataSeriesSet Function

C

```
short GFX_GOL_ChartDataSeriesSet(
    GFX_GOL_CHART * pCh,
    uint16_t seriesNum,
    BYTE status
);
```

Description

This is function GFX_GOL_ChartDataSeriesSet.

5.2.2.7.5.7 GFX_GOL_ChartDraw Function**C**

```
uint16_t GFX_GOL_ChartDraw(  
    void * pObj  
);
```

Description

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The colors of the bars of the bar chart or sectors of the pie chart can be the default color table or user defined color table set by ChSetColorTable() function.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Preconditions

Object must be created before this function is called.

Parameters

Parameters	Description
pCh	Pointer to the object to be rendered.

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Side Effects

none.

5.2.2.7.5.8 GFX_GOL_ChartPercentRangeSet Function**C**

```
void GFX_GOL_ChartPercentRangeSet(  
    GFX_GOL_CHART * pCh,  
    uint16_t min,  
    uint16_t max  
);
```

Description

This function sets the minimum and maximum range of percentage that the bar chart will show. The criteria is that min <= max. This affects bar charts only and CH_PERCENTAGE bit state is set.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.
min	Minimum percentage value that will be displayed in the bar chart.
max	Maximum percentage value that will be displayed in the bar chart.

Returns

none.

Side Effects

none.

5.2.2.7.5.9 GFX_GOL_ChartSampleRangeSet Function**C**

```
void GFX_GOL_ChartSampleRangeSet(
    GFX_GOL_CHART * pCh,
    uint16_t start,
    uint16_t end
);
```

Description

This function sets the sample start and sample end when drawing the chart. Together with the data series' [SHOW_DATA](#) flags the different way of displaying the chart data is achieved.

Start & End Value	The # of Data Series Flag Set	Chart Description
Start <= End	1	Show the data indicated by Start and End points of the DATASERIES with the flag set
Start = End	1	Show the data indicated by Start or End points of the DATASERIES with the flag set
Start, End = don't care	> 1	Show the data indicated by Start point of the DATASERIES with the flag set. Each samples of all checked data series are grouped together according to sample number.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.
start	Start point of the data samples to be displayed.
end	End point of the data samples to be displayed.

Returns

none.

Side Effects

none.

Example

See ChCreate() example.

5.2.2.7.5.10 GFX_GOL_ChartValueRangeSet Function**C**

```
void GFX_GOL_ChartValueRangeSet(  
    GFX_GOL_CHART * pCh,  
    uint16_t min,  
    uint16_t max  
);
```

Description

This function sets the minimum and maximum range of values that the bar chart will show. The criteria is that min <= max.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.
min	Minimum value that will be displayed in the bar chart.
max	Maximum value that will be displayed in the bar chart.

Returns

none.

Side Effects

none.

5.2.2.7.6 Custom Control Object Functions**5.2.2.7.6.1 GFX_GOL_CustomControlActionGet Function****C**

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_CustomControlActionGet(  
    void * pObj,  
    GFX_GOL_MESSAGE * pMsg  
);
```

Description

translates the GOL message for the custom control

Preconditions

none

Parameters

Parameters	Description
pMsg	pointer to the GOL message
pCc	the pointer to the custom control

Returns

translated message for the custom control

Side Effects

none

Remarks

none

5.2.2.7.6.2 GFX_GOL_CustomControlActionSet Function

C

```
void GFX_GOL_CustomControlActionSet(
    GFX_GOL_TRANSLATED_ACTION translatedMsg,
    void * pObj,
    GFX_GOL_MESSAGE * pMsg
);
```

Description

changes the state of the object

Preconditions

none

Parameters

Parameters	Description
pMsg	pointer to the GOL message
pCc	the pointer to the custom control

Returns

none

Side Effects

none

Remarks

none

5.2.2.7.6.3 GFX_GOL_CustomControlCreate Function

C

```
GFX_GOL_CUSTOMCONTROL * GFX_GOL_CustomControlCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

creates the custom control

Preconditions

none

Parameters

Parameters	Description
ID	instance unique ID
left, top, right, bottom	dimensions of the object
state	state of the object
pScheme	pointer to the style scheme

Returns

returns the pointer to the object created

Side Effects

none

Remarks

none

5.2.2.7.6.4 GFX_GOL_CustomControlDraw Function

C

```
GFX_STATUS GFX_GOL_CustomControlDraw(  
    void * pObj  
);
```

Description

draws custom control

Preconditions

none

Parameters

Parameters	Description
pCc	pointer to the custom control

Returns

returns the status of the drawing 0 - not completed 1 - done

Side Effects

none

Remarks

none

5.2.2.7.7 Digital Meter Object Functions

5.2.2.7.7.1 GFX_GOL_DigitalMeterActionGet Function

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_DigitalMeterActionGet(  

```

```

    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);

```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_DIGITALMETER_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the Digital Meter.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_DIGITALMETER_ACTION_SELECTED ? Object is selected.
- GFX_GOL_OBJECT_ACTION_INVALID ? Object is not affected

Example

None.

5.2.2.7.2 GFX_GOL_DigitalMeterCreate Function

C

```

GFX_GOL_DIGITALMETER * GFX_GOL_DigitalMeterCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    uint32_t value,
    uint8_t NoOfDigits,
    uint8_t DotPos,
    GFX_ALIGNMENT alignment,
    GFX_GOL_OBJ_SCHEME * pScheme
);

```

Description

This function creates a **GFX_GOL_DIGITALMETER** object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

Preconditions

None.

Parameters

Parameters	Description
instance	Device instance
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
value	Sets the initial value to be displayed
NoOfDigits	Sets the number of digits to be displayed
DotPos	Sets the position of decimal point in the display
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme. Set to NULL if default style scheme is used.

Returns

Pointer to the newly created object.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_DIGITALMETER *pDm;

pScheme = GFX_GOL_SchemeCreate();
state = GFX_GOL_DIGITALMETER_DRAW_STATE |
        GFX_GOL_DIGITALMETER_FRAME_STATE;
GFX_GOL_DigitalMeterCreate(
    ID_DIGITALMETER1,    // ID
    30,80,235,160,       // dimension
    state,               // has frame and center aligned
    789,4,1,             // to display 078.9
    GFX_ALIGN_CENTER,
    pScheme);           // use given scheme

// draw the objects
while(GFX_GOL_ObjectListDraw() != GFX_STATUS_SUCCESS);
```

5.2.2.7.3 GFX_GOL_DigitalMeterDecrement Function**C**

```
void GFX_GOL_DigitalMeterDecrement(
    GFX_GOL_DIGITALMETER * pObject,
    uint16_t delta
);
```

Description

This function decrements the meter value by the given delta value set. If the delta given is less than the minimum value of the meter, the value will remain to be at minimum.

Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

See [GFX_GOL_DigitalMeterIncrement\(\)](#).

5.2.2.7.7.4 GFX_GOL_DigitalMeterDraw Function**C**

```
GFX_STATUS GFX_GOL_DigitalMeterDraw(  
    void * pobject  
);
```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.7.5 GFX_GOL_DigitalMeterIncrement Function**C**

```
void GFX_GOL_DigitalMeterIncrement(  
    GFX_GOL_DIGITALMETER * pobject,  
    uint16_t delta  
);
```

Description

This function increments the scroll bar position by the given delta value set. If the delta given exceeds the maximum value of the meter, the value will remain to be at maximum.

Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

```
void ControlSpeed( GFX_GOL_DIGITALMETER* pObj,
                  int setSpeed,
                  int curSpeed)
{
    // set page size to 1
    GFX_GOL_DigitalMeterValueSet(pObj, 1);

    if (setSpeed < curSpeed)
    {
        while(GFX_GOL_DigitalMeterValueGet(pObj) < SetSpeed)
        {
            // increment by 1
            GFX_GOL_DigitalMeterIncrement(pObj, 1);
        }
    }
    else if (setSpeed > curSpeed)
    {
        while(GFX_GOL_DigitalMeterValueGet(pObj) > SetSpeed)
        {
            // decrement by 1
            GFX_GOL_DigitalMeterDecrement(pObj, 1);
        }
    }
}
```

5.2.2.7.7.6 GFX_GOL_DigitalMeterValueSet Function

C

```
void GFX_GOL_DigitalMeterValueSet(
    GFX_GOL_DIGITALMETER * pObj,
    int16_t value
);
```

Description

This function sets the value of the meter. The value used should be within the range set for the object. The new value is checked to be in the minimum and maximum value range. If the value set is less than the minimum value, the value that will be set is the minimum value. If the value set is more than the maximum value, then the value that will be set is the maximum value.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
value	the new value of the object.

Returns

None.

Example

```

GFX_GOL_DIGITALMETER *pMeter;
uint16_t ctr = 0;

// create slider here and initialize parameters
GFX_GOL_ObjectStateSet(pMeter, GFX_GOL_DIGITALMETER_DRAW_STATE);
GFX_GOL_ObjectListDraw();

while("some condition")
{
    GFX_GOL_DigitalMeterValueSet(pMeter, ctr);
    GFX_GOL_ObjectStateSet( pMeter,
                           GFX_GOL_DIGITALMETER_UPDATE_STATE);

    // redraw the scroll bar
    GFX_GOL_ObjectListDraw();

    // update ctr here
    ctr = "some source of value";
}

```

5.2.2.7.8 Edit Box Object Functions**5.2.2.7.8.1 GFX_GOL_EditBoxActionGet Function****C**

```

GFX_GOL_TRANSLATED_ACTION GFX_GOL_EditBoxActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);

```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_EDITBOX_ACTION_ADD_CHAR	Keyboard	EVENT_CHARCODE	New character should be added.
GFX_GOL_EDITBOX_ACTION_DEL_CHAR	Keyboard	EVENT_KEYPRESS	Last character should be removed.
GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN	Touch Screen	GFX_GOL_EDITBOX_FOCUSED_STATE	Focus will be drawn, caret displayed when enabled.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_EDITBOX_ACTION_ADD_CHAR - New character should be added
- GFX_GOL_EDITBOX_ACTION_DEL_CHAR - Last character should be removed.
- GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN - focus will be drawn when enabled. Careet will be drawn when enabled.
- GFX_GOL_OBJECT_ACTION_INVALID - Object is not affected

Example

None.

5.2.2.7.8.2 GFX_GOL_EditBoxActionSet Function**C**

```
void GFX_GOL_EditBoxActionSet(
    GFX_GOL_TRANSLATED_ACTION translatedMsg,
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit Description
GFX_GOL_EDITBOX_ACTION_ADD_CHAR	Keyboard	Set GFX_GOL_EDITBOX_DRAW_STATE New character is added and Edit Box will be redrawn.
GFX_GOL_EDITBOX_ACTION_DEL_CHAR	KeyBoard	Set GFX_GOL_EDITBOX_DRAW_STATE Last character is removed and Edit Box will be redrawn.
GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN	Touch Screen	Set GFX_GOL_EDITBOX_FOCUSED_STATE When enabled, focus will be redrawn, caret will also be redrawn if enabled.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Returns

None.

Example

None.

5.2.2.7.8.3 GFX_GOL_EditBoxCharAdd Function**C**

```
GFX_STATUS GFX_GOL_EditBoxCharAdd(  
    GFX_GOL_EDITBOX * pObject,  
    GFX_XCHAR ch  
);
```

Description

This function adds a character at the end of the text used by the object. When the object contains the maximum number of characters any addition call to this function will not affect the text in the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
ch	The character to be added.

Returns

The status of the addition.

- GFX_STATUS_SUCCESS - when the addition was successful.
- GFX_STATUS_FAILURE - when the addition was not successful.

Example

None.

5.2.2.7.8.4 GFX_GOL_EditBoxCharRemove Function**C**

```
GFX_STATUS GFX_GOL_EditBoxCharRemove(  
    GFX_GOL_EDITBOX * pObject  
);
```

Description

This function removes a character at the end of the text used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

The status of the addition.

- GFX_STATUS_SUCCESS - when the removal was successful.

- GFX_STATUS_FAILURE - when the removal has no effect and the buffer is empty.

Example

None.

5.2.2.7.8.5 GFX_GOL_EditBoxCreate Function

C

```
GFX_GOL_EDITBOX * GFX_GOL_EditBoxCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    GFX_XCHAR * pText,
    uint16_t charMax,
    GFX_ALIGNMENT alignment,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GFX_GOL_EDITBOX](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of [GFX_GOL_ListBoxCreate\(\)](#) will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- pText is an unterminated string

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object.
charMax	Defines the maximum number of characters in the edit box.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```
// assume pScheme to be initialized with the style scheme
```

```

#define MAX_CHAR_COUNT 15

GFX_GOL_OBJ_SCHEME      *pScheme;
GFX_GOL_EDITBOX         *pEb;
GFX_GOL_EDITBOX_STATE   state;

GFX_XCHAR                TextBuffer[MAX_CHAR_COUNT + 1];

// assume pScheme is initialized to a scheme in memory.
state = GFX_GOL_EDITBOX_DRAW_STATE;

pEb = GFX_GOL_EditBoxCreate(
    1, 20, 64, 50, 118,
    state, TextBuffer, MAX_CHAR_COUNT
    GFX_ALIGN_CENTER,
    pScheme);

// check if object is not created
if (pEb == NULL)
    return 0;

```

5.2.2.7.8.6 GFX_GOL_EditBoxDraw Function

C

```

GFX_STATUS GFX_GOL_EditBoxDraw(
    void * pObject
);

```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.8.7 GFX_GOL_EditBoxTextSet Function

C

```

void GFX_GOL_EditBoxTextSet(
    GFX_GOL_EDITBOX * pObject,

```

```
GFX_XCHAR* pText
);
```

Description

This function sets the text in the object. This function copies the text located in the address pointed to by pText to the object text buffer.

The string length must be less than or equal to the maximum characters allowed in the object. The object will truncate the string once it reaches the maximum length.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be copied.

Returns

None.

Example

```
#define MAX_EDITBOX_CHAR_LENGTH 20
GFX_XCHAR Label0[] = "This is really a long string";

// assume pEb is a pointer initialized to an edit box that
// as a buffer with 20 characters allowed.

GFX_GOL_EditBoxTextSet(pEb, Label0);

// at this point the edit box contains
// This is really a lon" - truncated to only 20 characters.
```

5.2.2.7.9 Grid Object Functions

5.2.2.7.9.1 GridClearCellState Function

C

```
void GridClearCellState(
    GRID * pGrid,
    short column,
    short row,
    uint16_t state
);
```

Description

This function clears the state of the cell (or Grid Item) specified by the column and row.

Preconditions

none

Parameters

Parameters	Description
pGrid	Pointer to the object.
column	column index of the cell
row	row index of the cell

atate	specifies the state to be cleared. See Grid Item State.

Returns

none.

Side Effects

none

5.2.2.7.9.2 GridCreate Function

C

```
GRID * GridCreate(
    uint16_t ID,
    short left,
    short top,
    short right,
    short bottom,
    uint16_t state,
    short numColumns,
    short numRows,
    short cellWidth,
    short cellHeight,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GRID](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Preconditions

none

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the Object.
top	Top most position of the Object.
right	Right most position of the Object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
numColumns	Sets the number of columns for the grid.
numRows	Sets the number of rows for the grid.
cellWidth	Sets the width of each cell of the grid.
cellHeight	Sets the height of each cell of the grid.
pScheme	Pointer to the style scheme used for the object. Set to NULL if default style scheme is used.

Returns

Returns the pointer to the object created.

Side Effects

none

5.2.2.7.9.3 GridDraw Function

C

```
uint16_t GridDraw(  
    void * pObj  
);
```

Description

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Preconditions

Object must be created before this function is called.

Parameters

Parameters	Description
pGb	Pointer to the object to be rendered.

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Side Effects

none

5.2.2.7.9.4 GridFreeItems Function

C

```
void GridFreeItems(  
    void * pObj  
);
```

Description

This function removes all grid items for the given Grid and frees the memory used.

Preconditions

Object must be created before this function is called.

Parameters

Parameters	Description
pGrid	The pointer to the Grid object.

Returns

none.

Side Effects

none

5.2.2.7.9.5 GridGetCell Function**C**

```
void * GridGetCell(
    GRID * pGrid,
    short column,
    short row,
    uint16_t * cellType
);
```

Description

This function removes all grid items for the given Grid and frees the memory used.

Preconditions

Object must be created before this function is called.

Parameters

Parameters	Description
pGrid	The pointer to the Grid object.
column	the column index of the cell
row	the row index of the cell
cellType	pointer that will receive the type of grid item or cell (GRIDITEM_IS_TEXT or GRIDITEM_IS_BITMAP).

Returns

Returns a pointer to the grid item or cell data.

Side Effects

none

5.2.2.7.9.6 GridMsgDefault Function**C**

```
void GridMsgDefault(
    uint16_t translatedMsg,
    void * pObj,
    GFX_GOL_MESSAGE * pMsg
);
```

Description

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GRID_MSG_TOUCHED	Touch Screen	none	Grid will have no state change because of this event.
GRID_MSG_ITEM_SELECTED	Keyboard	Set GRIDITEM_SELECTED , GRID_DRAW_ITEMS	Grid Item selected will be redrawn.

GRID_MSG_UP	Keyboard	Set GRIDITEM_DRAW ,	Grid Item above the currently focused item will be redrawn.
		GRID_DRAW_ITEMS	
GRID_MSG_DOWN	Keyboard	Set GRIDITEM_DRAW ,	Grid Item below the currently focused item will be redrawn.
		GRID_DRAW_ITEMS	
GRID_MSG_LEFT	Keyboard	Set GRIDITEM_DRAW ,	Grid Item to the left of the currently focused item will be redrawn.
		GRID_DRAW_ITEMS	
GRID_MSG_RIGHT	Keyboard	Set GRIDITEM_DRAW ,	Grid Item to the right of the currently focused item will be redrawn.
		GRID_DRAW_ITEMS	

Preconditions

none

Parameters

Parameters	Description
translatedMsg	The translated message.
pGrid	The pointer to the object whose state will be modified.
pMsg	The pointer to the GOL message.

Returns

none

Side Effects

none

5.2.2.7.9.7 GridSetCell Function**C**

```
uint16_t GridSetCell(
    GRID * pGrid,
    short column,
    short row,
    uint16_t state,
    void * data
);
```

Description

This function sets the Grid Item state and data.

Preconditions

Object must be created before this function is called.

Parameters

Parameters	Description
pGrid	The pointer to the Grid object.
column	the column index of the cell
row	the row index of the cell

state	sets the state of the Grid Item specified.
data	pointer to the data used for the Grid Item.

Returns

Returns the status of the operation

- [GRID_SUCCESS](#) - if the set succeeded
- [GRID_OUT_OF_BOUNDS](#) - if the row and column given results in an out of bounds location.

Side Effects

none

5.2.2.7.9.8 GridSetCellState Function

C

```
void GridSetCellState(  
    GRID * pGrid,  
    short column,  
    short row,  
    uint16_t state  
);
```

Description

This function sets the state of the Grid Item or cell.

Preconditions

Object must be created before this function is called.

Parameters

Parameters	Description
pGrid	The pointer to the Grid object.
column	the column index of the cell
row	the row index of the cell
state	sets the state of the Grid Item specified.

Returns

none.

Side Effects

none

5.2.2.7.9.9 GridSetFocus Function

C

```
void GridSetFocus(  
    GRID * pGrid,  
    short column,  
    short row  
);
```

Description

This function sets the focus of the specified Grid Item or cell.

Preconditions

Object must be created before this function is called.

Parameters

Parameters	Description
pGrid	The pointer to the Grid object.
column	the column index of the cell
row	the row index of the cell

Returns

none.

Side Effects

none

5.2.2.7.9.10 GridTranslateMsg Function**C**

```
uint16_t GridTranslateMsg(
    void * pObj,
    GFX_GOL_MESSAGE * pMsg
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GRID_MSG_TOUCHED	Touch Screen	none	If any touch events occurs and the x,y position falls in the face of the grid.
GRID_MSG_ITEM_SELECTED	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_SPACE_PRESSED or SCAN_CR_PRESSED .
GRID_MSG_UP	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_UP_PRESSED .
GRID_MSG_DOWN	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_DOWN_PRESSED .
GRID_MSG_LEFT	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_LEFT_PRESSED .
GRID_MSG_RIGHT	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_RIGHT_PRESSED .
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Preconditions

none

Parameters

Parameters	Description
pGrid	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMsg	Pointer to the message struct containing the message from the user interface.

Returns

Returns the translated message depending on the received GOL message:

- GRID_MSG_TOUCHED - when the grid object is touched.
- GRID_MSG_ITEM_SELECTED – when key scan [SCAN_SPACE_PRESSED](#) or [SCAN_CR_PRESSED](#) are detected.
- GRID_MSG_UP – when key scan [SCAN_UP_PRESSED](#) is detected.
- GRID_MSG_DOWN – when key scan [SCAN_DOWN_PRESSED](#) is detected.
- GRID_MSG_LEFT – when key scan [SCAN_LEFT_PRESSED](#) is detected.
- GRID_MSG_RIGHT – when key scan [SCAN_RIGHT_PRESSED](#) is detected.
- OBJ_MSG_INVALID – Button is not affected

Side Effects

none

5.2.2.7.10 Group Box Object Functions

5.2.2.7.10.1 GFX_GOL_GroupboxActionGet Function

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_GroupboxActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source Events	Description
GFX_GOL_GROUPBOX_ACTION_SELECTED	Touch Screen EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the group box.
GFX_GOL_OBJECT_ACTION_INVALID	Any Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.

pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_GROUPBOX_ACTION_SELECTED // Groupbox area selected action ID.

Example

None.

5.2.2.7.10.2 GFX_GOL_GroupboxCreate Function**C**

```
GFX_GOL_GROUPBOX * GFX_GOL_GroupboxCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    GFX_XCHAR * pText,
    GFX_ALIGNMENT alignment,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GFX_GOL_GROUPBOX](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pImage	Pointer to the image used on the face of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_GROUPBOX *pGroupbox;
GFX_GOL_GROUPBOX_STATE state;

// assume pScheme is initialized to a scheme in memory.
state = GFX_GOL_GROUPBOX_DRAW_STATE;
```



```

pGroupBox = GFX_GOL_GroupboxCreate(1,           // ID
0,0,GFX_Primitive_MaxXGet(),GFX_Primitive_MaxYGet(), // whole screen dimension
state,           // set state to draw all
(char*)myIcon,   // icon
"Place Title Here.", // text
NULL);          // use default GOL scheme

if (pwindow == NULL)
    return 0;

return 1;

```

5.2.2.7.10.3 GFX_GOL_GroupboxDraw Function

C

```

GFX_STATUS GFX_GOL_GroupboxDraw(
    void * pObject
);

```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the [GFX_GOL_GROUPBOX](#) is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.10.4 GFX_GOL_GroupboxTextAlignmentGet Function

C

```

GFX_ALIGNMENT GFX_GOL_GroupboxTextAlignmentGet(
    GFX_GOL_GROUPBOX * pObject
);

```

Description

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The text alignment set in the object. See [GFX_ALIGNMENT](#) for more details.

Example

None.

5.2.2.7.10.5 GFX_GOL_GroupboxTextAlignmentSet Function**C**

```
void GFX_GOL_GroupboxTextAlignmentSet(  
    GFX_GOL_GROUPBOX * pObject,  
    GFX_ALIGNMENT align  
);
```

Description

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Returns

None.

Example

None.

5.2.2.7.10.6 GFX_GOL_GroupboxTextSet Function**C**

```
void GFX_GOL_GroupboxTextSet(  
    GFX_GOL_GROUPBOX * pObject,  
    GFX_XCHAR * pText  
);
```

Description

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Returns

None.

Example

```
GFX_XCHAR Label0[] = ?Group One?;
GFX_XCHAR Label1[] = ?Group Two?;
GFX_GOL_GROUPBOX pGroupbox;

GFX_GOL_GroupboxTextSet(pGroupbox, Label0);
GFX_GOL_GroupboxTextSet(pGroupbox, Label1);
```

5.2.2.7.11 List Box Object Functions**5.2.2.7.11.1 GFX_GOL_ListBoxActionGet Function****C**

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_ListBoxActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN	Touch Screen	Any	Item is selected using touch screen.
GFX_GOL_LISTBOX_ACTION_MOVE	Keyboard	EVENT_KEYSCAN	Focus is moved to the next item depending on the key pressed (UP or DOWN key).
GFX_GOL_LISTBOX_ACTION_SELECTED	Keyboard	EVENT_KEYSCAN	Selection status (GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED) is set to the currently focused item.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN â€” Item is selected using touch screen
- GFX_GOL_LISTBOX_ACTION_MOVE â€” Focus is moved to the next item depending on the key pressed (UP or DOWN key).
- GFX_GOL_LISTBOX_ACTION_SELECTED â€” Selection is set to the currently focused item
- GFX_GOL_OBJECT_ACTION_INVALID â€” Object is not affected

Example

None.

5.2.2.7.11.2 GFX_GOL_ListBoxActionSet Function**C**

```
void GFX_GOL_ListBoxActionSet(
    GFX_GOL_TRANSLATED_ACTION translatedMsg,
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit Description
GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN	Touch Screen	Set GFX_GOL_LISTBOX_FOCUSED_STATE, If focus is enabled, the focus state bit GFX_GOL_LISTBOX_FOCUSED_STATE will be set.
		Set GFX_GOL_LISTBOX_DRAW_FOCUS_STATE GFX_GOL_LISTBOX_DRAW_FOCUS_STATE draw state bit will force the List Box to be redrawn with focus.
		Set GFX_GOL_LISTBOX_DRAW_ITEMS_STATE List Box will be redrawn with selected item(s).
GFX_GOL_LISTBOX_ACTION_MOVE	KeyBoard	Set GFX_GOL_LISTBOX_DRAW_ITEMS_STATE List Box will be redrawn with focus on one item.
GFX_GOL_LISTBOX_ACTION_SELECTED	KeyBoard	Set GFX_GOL_LISTBOX_DRAW_ITEMS_STATE List Box will be redrawn with selection on the current item focused.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Returns

None.

Example

None.

5.2.2.7.11.3 GFX_GOL_ListBoxCreate Function**C**

```
GFX_GOL_LISTBOX * GFX_GOL_ListBoxCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    GFX_XCHAR * pText,
    GFX_ALIGNMENT alignment,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GFX_GOL_LISTBOX](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of `GFX_GOL_ListBoxCreate()` will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- pText is an unterminated string

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object. This is used for the items of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```
#define LISTBOX_ID 10
const XCHAR ItemList[] = "Line1n" "Line2n";

GFX_GOL_OBJ_SCHEME *pScheme;
```

```

GFX_GOL_LISTBOX    *pLb;
GFX_XCHAR          *pTemp;
uint16_t           state, counter;

//assume scheme is a valid style scheme in memory
pScheme = &scheme;
state = GFX_GOL_LISTBOX_DRAW_STATE;

// create an empty listbox with default style scheme
pLb = GFX_GOL_ListBoxCreate(    LISTBOX_ID,    // ID number
                                10,10,150,200,   // dimension
                                state,          // initial state
                                NULL,          // set items to be empty
                                GFX_ALIGN_CENTER,
                                NULL);          // use default style scheme

// check if Listbox was created
if (pLb == NULL)
    return 0;

// create the list of items to be placed in the listbox
// Add items (each line will become one item,
// lines must be separated by 'n' character)
pTemp = ItemList;
counter = 0;

while(*pTemp)
{
    // since each item is appended NULL is assigned to
    // GFX_GOL_LISTITEM pointer.
    if(NULL == GFX_GOL_ListBoxItemAdd(
        pLb,
        NULL,
        pTemp,
        NULL,
        0,
        counter))
    {
        break;
    }

    while(*pTemp++ > (unsigned GFX_XCHAR)31);

    if(*(pTemp-1) == 0)
        break;
    counter++;
}

```

5.2.2.7.11.4 GFX_GOL_ListBoxDraw Function

C

```

GFX_STATUS GFX_GOL_ListBoxDraw(
    void * pObject
);

```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.11.5 GFX_GOL_ListBoxItemAdd Function

C

```
GFX_GOL_LISTITEM * GFX_GOL_ListBoxItemAdd(
    GFX_GOL_LISTBOX * pObject,
    GFX_GOL_LISTITEM * pPrevItem,
    GFX_XCHAR * pText,
    GFX_RESOURCE_HDR * pImage,
    uint16_t status,
    uint16_t data
);
```

Description

This function adds an item to the list box. This function allocates the memory needed for the [GFX_GOL_LISTITEM](#) and adds it to the list box. The newly created [GFX_GOL_LISTITEM](#) will store the location of pText, pImage and other parameters describing the added item.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
pPrevItem	Pointer to the item after which a new item must be inserted, if this pointer is NULL, the item will be appended at the end of the items list.
pText	Pointer to the text that will be inserted. Text must persist in memory for as long as it is referenced by an item in the list box.
pImage	Pointer to the image for the item. Image must persist in memory for as long as it is referenced by the an item in the list box.
status	This parameter specifies if the item being added will be selected or redrawn (LB_STS_SELECTED or LB_STS_REDRAW). Refer to LISTITEM structure for details.
data	User assigned data associated with the item.

Returns

The pointer to the created item.

Example

```

const GFX_XCHAR ItemList[] = "Line1n" "Line2n" "Line3n";

extern GFX_RESOURCE_HDR myIcon;
GFX_GOL_LISTBOX      *pLb;
GFX_GOL_LISTITEM     *pItem, *pItemList;
GFX_XCHAR            *pTemp;

// Assume that pLb is pointing to an existing list box in memory
// that is empty (no list).

// Create the list of the list box

// Initialize this to NULL to indicate that items will be added
// at the end of the list if the list exist on the list box or
// start a new list if the list box is empty.
pItem = NULL;
pTemp = ItemList;
pItem = GFX_GOL_ListBoxItemAdd(
    pLb,
    pItem,
    pTemp,
    NULL,
    LB_STS_SELECTED,
    1);
if(pItem == NULL)
    return 0;
GFX_GOL_ListBoxImageSet(pItem, &myIcon);

// Adjust pTemp to point to the next line
while((uint16_t)*pTemp++ > (uint16_t)31);

// add the next item
pItem = GFX_GOL_ListBoxItemAdd(
    pLb,
    pItem,
    pTemp,
    NULL,
    0,
    2)
if(pItem == NULL)
    return 0;
GFX_GOL_ListBoxImageSet(pItem, &myIcon);

// Adjust pTemp to point to the next line
while((uint16_t)*pTemp++ > (uint16_t)31);

// this time insert the next item after the first item on the list
pItem = LbGetItemList(pLb);
pItem = GFX_GOL_ListBoxItemAdd(
    pLb,
    pItem,
    pTemp,
    NULL,
    0,
    3)
if(pItem == NULL)
    return 0;
GFX_GOL_ListBoxImageSet(pItem, &myIcon);

```

5.2.2.7.11.6 GFX_GOL_ListBoxItemFocusGet Function**C**

```

int16_t GFX_GOL_ListBoxItemFocusGet(

```



```
    GFX_GOL_LISTBOX * pObject
);
```

Description

This function returns the index of the focused item in the list box. First item on the list is always indexed 0. If none of the items in the list is focused, -1 is returned.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object.

Returns

The index of the focused item in the list box.

Example

None.

5.2.2.7.11.7 GFX_GOL_ListBoxItemFocusSet Function

C

```
void GFX_GOL_ListBoxItemFocusSet(
    GFX_GOL_LISTBOX * pObject,
    uint16_t index
);
```

Description

This function sets the focus of the item with the index number specified by index. First item on the list is always indexed 0.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object.
index	The index number of the item to be focused.

Returns

None.

Example

None.

5.2.2.7.11.8 GFX_GOL_ListBoxItemListRemove Function

C

```
void GFX_GOL_ListBoxItemListRemove(
    void * pObject
);
```

Description

This function removes the entire items list of the list box and free the memory used. The memory freed is the memory used for

the list box. The actual text or image used for the item are not removed from memory.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

None.

5.2.2.7.11.9 GFX_GOL_ListBoxItemRemove Function

C

```
void GFX_GOL_ListBoxItemRemove(  
    GFX_GOL_LISTBOX * pObject,  
    GFX_GOL_LISTITEM * pItem  
);
```

Description

This function removes an item from the list box and free the memory used. The memory freed is the memory used for the list box. The actual text or image used for the item are not removed from memory.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pltem	pointer to the list box item that will be removed.

Returns

None.

Example

None.

5.2.2.7.11.10 GFX_GOL_ListBoxSelectionChange Function

C

```
void GFX_GOL_ListBoxSelectionChange(  
    GFX_GOL_LISTBOX * pObject,  
    GFX_GOL_LISTITEM * pItem  
);
```

Description

This function changes the selection status of the given item in the list box. There are two cases that this function checks Case 1: The list box is set to multiple selection.

- The item pointed to by pltem will toggle the selection status.

Case 2: The list box is set to single selection.

- If the currently selected item is not the item pointed to by pItem, the currently selected item will toggle to not selected. The item pointed to by pItem will then be set to selected.
- If the currently selected item is the same item pointed to by pItem, the selection status of that item will be set to not selected.

The change in the item's selection status should be redrawn to reflect the changes.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object.
pItem	The pointer to the item that will have the selection status changed.

Returns

None.

Example

None.

5.2.2.7.11.11 GFX_GOL_ListBoxSelectionGet Function

C

```
GFX_GOL_LISTITEM * GFX_GOL_ListBoxSelectionGet(  
    GFX_GOL_LISTBOX * pObject,  
    GFX_GOL_LISTITEM * pFromItem  
);
```

Description

This function searches for selected items from the list box. A starting position can optionally be given. If starting position is set to NULL, search will begin from the first item on the item list.

The function returns the pointer to the first selected item found or NULL if there are no items selected.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object.
pFromItem	The starting point of the search. If this is set to NULL, the search will start at the beginning of the item list.

Returns

The pointer to the first selected item found. NULL if there are no items selected.

Example

None.

5.2.2.7.11.12 GFX_GOL_ListBoxVisibleItemCountGet Function

C

```
uint16_t GFX_GOL_ListBoxVisibleItemCountGet(
    GFX_GOL_LISTBOX * pObject
);
```

Description

This function returns the count of items visible in the list box.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object.

Returns

The number of visible items in the list box.

Example

None.

5.2.2.7.12 Meter Object Functions

5.2.2.7.12.1 GFX_GOL_MeterActionGet Function

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_MeterActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_METER_ACTION_SET	System	EVENT_SET	If event set occurs and the meter id is sent in parameter 1.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_METER_ACTION_SET â€” Meter id is given in parameter 1 for a TYPE_SYSTEM message.
- GFX_GOL_OBJECT_ACTION_INVALID â€” Object is not affected

Example

None.

5.2.2.7.12.2 GFX_GOL_MeterActionSet Function**C**

```
void GFX_GOL_MeterActionSet(
    GFX_GOL_TRANSLATED_ACTION translatedMsg,
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_METER_ACTION_SET	System	Set GFX_GOL_METER_DRAW_STATE	Meter will be redrawn to update the needle position and value displayed.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Returns

None.

Example

None.

5.2.2.7.12.3 GFX_GOL_MeterCreate Function**C**

```
GFX_GOL_METER * GFX_GOL_MeterCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    GFX_GOL_METER_DRAW_TYPE type,
```

```

    int16_t value,
    int16_t minValue,
    int16_t maxValue,
    GFX_RESOURCE_HDR * pTitleFont,
    GFX_RESOURCE_HDR * pValueFont,
    GFX_XCHAR * pText,
    GFX_GOL_OBJ_SCHEME * pScheme
);

```

Description

This function creates a [GFX_GOL_METER](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of `GFX_GOL_MeterCreate()` will be undefined if one of the following is true:

- left >= right
- top >= bottom
- type is not one of the defined types
- pTitleFont and pValueFont is not defined to a valid font `GFX_RESOURCE_HDR`
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- pText is an unterminated string

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
type	Specifies the type of Meter to be drawn (see <code>GFX_GOL_METER_TYPE</code>).
value	Initial value set to the meter.
minValue	The minimum value the meter will display.
maxValue	The maximum value the meter will display.
pTitleFont	Pointer to the font used for the Title.
pValueFont	Pointer to the font used for the value.
pText	Pointer to the text label of the meter.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```

#define ID_METER 101

extern const FONT_FLASH GOLMediumFont;    // medium font
extern const FONT_FLASH GOLSmallFont;    // small font

```

```

GFX_GOL_OBJ_SCHEME *pMeterScheme;
GFX_GOL_METER *pMtr;

// assume pMeterScheme is initialized to a scheme in memory.

// draw object after creation
state = GFX_GOL_METER_DRAW_STATE | GFX_GOL_METER_RING_STATE;

pMtr = GFX_GOL_MeterCreate(
    ID_METER,                // assign ID
    30, 50, 150, 180,        // set dimension
    state,                   // type of meter
    GFX_GOL_METER_WHOLE_TYPE, // set initial value
    0,                       // set min and max value
    0, 100,                  // set title font
    &GOLMediumFont,          // set value font
    &GOLSmallFont,           // Text Label
    "Speed",                 // style scheme
    pMeterScheme);

// check if meter was created
if (pMtr == NULL)
    return 0;

// Change range colors: Normal values to WHITE
//                      Critical values to BLUE
//                      Danger values to RED
// assume that WHITE, GREEN, YELLOW and RED have been defined.
GFX_GOL_MeterScaleColorSet(pMtr, WHITE, WHITE, WHITE,
                           GREEN, YELLOW, RED);

// use GOLDraw() to draw the meter created
while(!GOLDraw());

```

5.2.2.7.12.4 GFX_GOL_MeterDecrement Function

C

```

void GFX_GOL_MeterDecrement(
    GFX_GOL_METER * pObject,
    uint16_t delta
);

```

Description

This function decrements the meter value by the given delta value set. If the delta given is less than the minimum value of the meter, the value will remain to be at minimum.

Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

See [GFX_GOL_MeterIncrement\(\)](#).

5.2.2.7.12.5 GFX_GOL_MeterDraw Function

C

```
GFX_STATUS GFX_GOL_MeterDraw(  
    void * pObject  
);
```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.12.6 GFX_GOL_MeterIncrement Function

C

```
void GFX_GOL_MeterIncrement(  
    GFX_GOL_METER * pObject,  
    uint16_t delta  
);
```

Description

This function increments the scroll bar position by the given delta value set. If the delta given exceeds the maximum value of the meter, the value will remain to be at maximum.

Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

```

void ControlSpeed( GFX_GOL_METER* pObj,
                  int setSpeed,
                  int curSpeed)
{
    // set page size to 1
    GFX_GOL_MeterValueSet(pObj, 1);

    if (setSpeed < curSpeed)
    {
        while(GFX_GOL_MeterValueGet(pObj) < SetSpeed)
            GFX_GOL_MeterIncrement(pObj, 1); // increment by 1
    }
    else if (setSpeed > curSpeed)
    {
        while(GFX_GOL_MeterValueGet(pObj) > SetSpeed)
            GFX_GOL_MeterDecrement(pObj, 1); // decrement by 1
    }
}

```

5.2.2.7.12.7 GFX_GOL_MeterRangeSet Function**C**

```

void GFX_GOL_MeterRangeSet(
    GFX_GOL_METER * pObj,
    int16_t minValue,
    int16_t maxValue
);

```

Description

This function sets the range of the meter. When the range is modified, object must be completely redrawn to reflect the change. minValue should always be less than maxValue.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
minValue	new minimum value of the object.
maxValue	new maximum value of the object.

Returns

None.

Example

None.

5.2.2.7.12.8 GFX_GOL_MeterScaleColorsSet Function**C**

```

void GFX_GOL_MeterScaleColorsSet(
    GFX_GOL_METER * pObj,
    GFX_COLOR color1,
    GFX_COLOR color2,
    GFX_COLOR color3,
    GFX_COLOR color4,
    GFX_COLOR color5,

```

```
GFX_COLOR color6
);
```

Description

After the object is created, this function must be called to set the arc colors of the object.

Scale colors can be used to highlight values of the meter. User can set these colors to define the arc colors and scale colors. This also sets the color of the meter value when displayed. The color settings are set to the following angles:

Color Boundaries	Type Whole	Type Half	Type Quarter
Arc 6	225 to 180	not used	not used
Arc 5	179 to 135	179 to 135	not used
Arc 4	134 to 90	134 to 90	not used
Arc 3	89 to 45	89 to 45	89 to 45
Arc 2	44 to 0	44 to 0	44 to 0
Arc 1	-45 to -1	not used	not used

As the meter is drawn colors are changed depending on the angle of the scale and label being drawn.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
color1	color assigned to Arc 1 and Scale 1.
color2	color assigned to Arc 2 and Scale 2.
color3	color assigned to Arc 3 and Scale 3.
color4	color assigned to Arc 4 and Scale 4.
color5	color assigned to Arc 5 and Scale 5.
color6	color assigned to Arc 6 and Scale 6.

Returns

None.

Example

None.

5.2.2.7.12.9 GFX_GOL_MeterValueSet Function

C

```
void GFX_GOL_MeterValueSet(
    GFX_GOL_METER * pObject,
    int16_t value
);
```

Description

This function sets the value of the meter. The value used should be within the range set for the object. The new value is checked to be in the minimum and maximum value range. If the value set is less than the minimum value, the value that will be set is the minimum value. If the value set is more than the maximum value, then the value that will be set is the maximum value.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
value	the new value of the object.

Returns

None.

Example

```
GFX_GOL_METER *pMeter;
uint16_t ctr = 0;

// create slider here and initialize parameters
GFX_GOL_ObjectStateSet(pMeter, GFX_GOL_METER_DRAW_STATE);
GFX_GOL_ObjectListDraw();

while("some condition")
{
    GFX_GOL_MeterValueSet(pMeter, ctr);
    GFX_GOL_ObjectStateSet( pMeter,
                           GFX_GOL_METER_UPDATE_DRAW_STATE);

    // redraw the scroll bar
    GFX_GOL_ObjectListDraw();

    // update ctr here
    ctr = "some source of value";
}
```

5.2.2.7.13 Progress Bar Object Functions**5.2.2.7.13.1 GFX_GOL_ProgressBarActionGet Function****C**

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_ProgressBarActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit Description
GFX_GOL_PROGRESSBAR_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE, If events occurs and the x,y position falls in the area of the object.
		EVENT_MOVE
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_PROGRESSBAR_ACTION_SELECTED “Object is selected”
- GFX_GOL_OBJECT_ACTION_INVALID “Object is not affected”

Example

None.

5.2.2.7.13.2 GFX_GOL_ProgressBarCreate Function**C**

```
GFX_GOL_PROGRESSBAR * GFX_GOL_ProgressBarCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    uint16_t pos,
    uint16_t range,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GFX_GOL_PROGRESSBAR](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of [GFX_GOL_RadioButtonCreate\(\)](#) will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pos > range
- range = 0
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- pText is an unterminated string

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.

top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pos	Defines the initial position of the progress.
range	This specifies the maximum value of the progress bar when the progress bar is at 100% position.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```

GFX_GOL_PROGRESSBAR *pPBar;
void CreateProgressBar()
{
    pPBar = PbCreate(ID_PROGRESSBAR1,    // ID
                    50,90,270,140,       // dimension
                    PB_DRAW,             // Draw the object
                    25,                  // position
                    50,                  // set the range
                    NULL);               // use default GOL scheme
}

```

5.2.2.7.13.3 GFX_GOL_ProgressBarDraw Function**C**

```

GFX_STATUS GFX_GOL_ProgressBarDraw(
    void * pObject
);

```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.13.4 GFX_GOL_ProgressBarPositionSet Function

C

```
void GFX_GOL_ProgressBarPositionSet(
    GFX_GOL_PROGRESSBAR * pobject,
    uint16_t position
);
```

Description

This function sets the position of the progress bar.

The value used for the position should be within the range set for the object.

Function will have an undefined behavior if the position is outside the range.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
position	the new position of the scroll bar thumb.

Returns

None.

Example

```
GFX_GOL_PROGRESSBAR *pPb;
uint8_t direction = 1;

// this code increments and decrements the progress bar by 1
// assume progress bar was created and initialized before
while (1)
{
    if(direction)
    {
        if(pPb ->pos == pPb ->range)
            direction = 0;
        else
            GFX_GOL_ProgressBarPositionSet(
                pPb,
                GFX_GOL_ProgressBarPositionGet(pPb)+1);
    }
    else
    {
        if(pPb ->pos == 0)
            direction = 1;
        else
            GFX_GOL_ProgressBarPositionSet(
                pPb,
                GFX_GOL_ProgressBarPositionGet(pPb)-1);
    }
}
```

5.2.2.7.13.5 GFX_GOL_ProgressBarRangeSet Function

C

```
void GFX_GOL_ProgressBarRangeSet(
    GFX_GOL_PROGRESSBAR * pobject,
    uint16_t range
);
```

```
);
```

Description

This function sets the range of the progress bar. When the range is modified, object must be completely redrawn to reflect the change.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
range	new range of the object.

Returns

None.

Example

None.

5.2.2.7.14 Picture Object Functions

5.2.2.7.14.1 GFX_GOL_PictureControlActionGet Function

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_PictureControlActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_PICTURECONTROL_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the area of the picture.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_PICTURECONTROL_ACTION_SELECTED “Object is selected”
- GFX_GOL_OBJECT_ACTION_INVALID “Object is not affected”

Example

None.

5.2.2.7.14.2 GFX_GOL_PictureControlCreate Function**C**

```
GFX_GOL_PICTURECONTROL * GFX_GOL_PictureControlCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    int8_t scaleFactor,
    GFX_RESOURCE_HDR * pImage,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GFX_GOL_PICTURECONTROL](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The object allows creation with the image set to NULL. In this case, nothing will be drawn when the object is set to be drawn and the frame is not enabled (See [GFX_GOL_PICTURECONTROL_FRAME_STATE](#)). If the frame is enabled, then only the frame will be drawn.

When the assigned image's dimension is larger than the dimension of the object, partial image parameters will be set in such a way that the upper left most corner of the image that has the same dimension as the object will be used in the object. This is the default behavior.

The partial parameters can be modified by calling the [GFX_GOL_PictureControlPartialSet\(\)](#) function with the desired partial image parameters. See [GFX_ImagePartialDraw\(\)](#) for details on the partial image rendering.

The behavior of [GFX_GOL_PictureControlCreate\(\)](#) will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- pImage is not pointing to a [GFX_RESOURCE_HDR](#).

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.

bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pImage	Pointer to the image by the object
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

None.

5.2.2.7.14.3 GFX_GOL_PictureControlDraw Function**C**

```
GFX_STATUS GFX_GOL_PictureControlDraw(
    void * pObject
);
```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When the image size The text on the face of the [GFX_GOL_PICTURECONTROL](#) is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.14.4 GFX_GOL_PictureControlPartialSet Function**C**

```
void GFX_GOL_PictureControlPartialSet(
    GFX_GOL_PICTURECONTROL * pObject,
    uint16_t xOffset,
    uint16_t yOffset,
    uint16_t partialWidth,
    uint16_t partialHeight
);
```

```
);
```

Description

This function sets the partial image parameters to be used in the object. This function allows usage of the object to specify a rectangular area of an image to be drawn as part of the object. This is useful when an image is already included in a larger image. To save memory, a separate image is not necessary for the picture object. The location of the smaller image in the larger image can be specified to show up in the picture object.

This function will result in an undefined behavior when one of the following is true:

- xOffset - value must not be greater than the image width.
- yOffset - value must not be greater than the image height.
- partialWidth - value must not be greater than image width - xoffset + 1. Value must also be less than the actual image width.
- partialHeight - value must not be greater than image height - yoffset + 1. Value must also be less than the actual image height.

Preconditions

Object must exist in memory. The image pointer of the object must be initialized properly.

Parameters

Parameters	Description
xOffset	x offset of the smaller portion of a large image
yOffset	y offset of the smaller portion of a large image
partialWidth	width of the selected portion of the image
partialHeight	height of the selected portion of the image

Returns

None.

Example

```
// assume pLargeImage is a valid GFX_RESOURCE_HDR
// assume BigImage has a height and width of 100 pixels.

GFX_RESOURCE_HDR *pLargeImage = &BigImage;
GFX_GOL_PICTURECONTROL *pPicture;
uint16_t          width, height;
uint16_t          xOffset, yOffset;
uint16_t          objectWidth, objectHeight;

objectWidth = 60 - 50; // 10 pixels
objectHeight = 120 - 90; // 30 pixels

// -1 is needed since the object dimension is inclusive
pPicture = GFX_GOL_PictureControlCreate(
    10,
    50, 90,
    50 + objectWidth - 1,
    90 + objectHeight - 1,
    GFX_GOL_PICTURECONTROL_DRAW_STATE,
    largeImage,
    NULL);

// set the parameters of the partial image to be
// shown on the image

// get the large image dimensions
width = GFX_ImageWidthGet(pLargeImage);
height = GFX_ImageHeightGet(pLargeImage);

// get the offset so the middle of the large image with
```

```
// the width and height matching the object will be used.
xOffset = (width - objectWidth) >> 1;
yOffset = (height - objectHeight) >> 1;

GFX_GOL_PictureControlPartialSet( pPicture,
                                   xOffset, yOffset,
                                   objectWidth, objectHeight);
```

5.2.2.7.15 Radio Button Object Functions

5.2.2.7.15.1 GFX_GOL_RadioButtonActionGet Function

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_RadioButtonActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_RADIOBUTTON_ACTION_CHECKED	Touch Screen	EVENT_PRESS	If event occurs and the x,y position falls in the area of the Radio Button while the Radio Button is not checked.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches <u>SCAN_CR_PRESSED</u> or <u>SCAN_SPACE_PRESSED</u> while the Radio Button is not checked.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_RADIOBUTTON_ACTION_CHECKED - Radio Button is checked
- GFX_GOL_OBJECT_ACTION_INVALID - object is not affected

Example

None.

5.2.2.7.15.2 GFX_GOL_RadioButtonActionSet Function

C

```
void GFX_GOL_RadioButtonActionSet(
    GFX_GOL_TRANSLATED_ACTION translatedMsg,
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit Description
GFX_GOL_RADIOBUTTON_ACTION_CHECKED	Touch Screen,	Set GFX_GOL_RADIOBUTTON_DRAW_STATE, Depending on the current value of RB_CHECKED Check Box will be redrawn.
	Keyboard	Set/Clear GFX_GOL_RADIOBUTTON_CHECKED_STATE

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Returns

None.

Example

None.

5.2.2.7.15.3 GFX_GOL_RadioButtonCheckGet Function

C

```
uint16_t GFX_GOL_RadioButtonCheckGet(
    GFX_GOL_RADIOBUTTON * pObject
);
```

Description

This function returns the ID of the currently checked radio button in the group. When there is only one member of the group, then that member will have the check.

When no member of the group is checked, then the id returned is (-1 or 0xFFFF).

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The ID of the member of the group with the check.

Example

```

static GFX_XCHAR  label0[] = "RB1";
static GFX_XCHAR  label1[] = "RB2";
static GFX_XCHAR  label2[] = "RB3";
uint16_t          state;
GFX_GOL_OBJ_SCHEME *pScheme;
RADIOBUTTON *pRb[3];
uint16_t          ID;

pScheme = GFX_GOL_ObjectSchemeCreate();

// Object will be drawn after creation
// Object will be first button in the group
state = GFX_GOL_RADIOBUTTON_DRAW_STATE |
        GFX_GOL_RADIOBUTTON_CHECKED_STATE;

pRb[0] = GFX_GOL_RadioButtonCreate(ID_RADIOBUTTON1,
                                   255, 40, 310, 80,
                                   state,
                                   label0,
                                   GFX_ALIGN_CENTER,
                                   pScheme);

// Object will be drawn after creation
state = GFX_GOL_RADIOBUTTON_DRAW_STATE;

pRb[1] = GFX_GOL_RadioButtonCreate(ID_RADIOBUTTON2,
                                   255, 85, 310, 125,
                                   state,
                                   label1,
                                   GFX_ALIGN_CENTER,
                                   pScheme);

// Object will be drawn after creation
state = GFX_GOL_RADIOBUTTON_DRAW_STATE;

pRb[2] = GFX_GOL_RadioButtonCreate(ID_RADIOBUTTON3,
                                   255, 130, 310, 170,
                                   state,
                                   label2,
                                   GFX_ALIGN_CENTER,
                                   pScheme);

// draw the objects
while(GFX_GOL_ObjectListDraw() != GFX_STATUS_SUCCESS);

// can also use pRb[1] or pRb[0] to search the checked
// radio button of the group. ID here should be ID_RADIOBUTTON1
ID = GFX_GOL_RadioButtonCheckGet(pRb[2]);

if (ID == ID_RADIOBUTTON1)
{
    // do something here then clear the check
    GFX_GOL_ObjectStateClear(pRb[0], RB_CHECKED);
    // Change the checked object. Pointer used is

```

```

        // any of the three. The ID used will find the
        // correct object to be checked
GFX_GOL_RadioButtonCheckSet(pRb[3], ID_RADIOBUTTON2);
}

```

5.2.2.7.15.4 GFX_GOL_RadioButtonCheckSet Function

C

```

void GFX_GOL_RadioButtonCheckSet(
    GFX_GOL_RADIOBUTTON * pObject,
    uint16_t id
);

```

Description

This function sets the ID of the currently checked radio button in the group. When there is only one member of the group, then that member will have the check.

When the id given does not exist in the group, the function will do nothing.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
id	id of the member of the group.

Returns

None.

Example

See [GFX_GOL_RadioButtonCheckGet\(\)](#) example.

5.2.2.7.15.5 GFX_GOL_RadioButtonCreate Function

C

```

GFX_GOL_RADIOBUTTON * GFX_GOL_RadioButtonCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    GFX_XCHAR * pText,
    GFX_ALIGNMENT alignment,
    GFX_GOL_OBJ_SCHEME * pScheme
);

```

Description

This function creates a [GFX_GOL_RADIOBUTTON](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of `GFX_GOL_RadioButtonCreate()` will be undefined if one of the following is true:

- left >= right
- top >= bottom

- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- pText is an unterminated string

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```

static GFX_XCHAR  label0[] = "RB1";
static GFX_XCHAR  label1[] = "RB2";
static GFX_XCHAR  label2[] = "RB3";
uint16_t          state;
GFX_GOL_OBJ_SCHEME *pScheme;
RADIOBUTTON *pRb[3];

pScheme = GFX_GOL_ObjectSchemeCreate();

// Object will be drawn after creation
// Object will be first button in the group
state = GFX_GOL_RADIOBUTTON_DRAW_STATE |
        GFX_GOL_RADIOBUTTON_CHECKED_STATE;

pRb[0] = GFX_GOL_RadioButtonCreate(ID_RADIOBUTTON1,
                                   255, 40, 310, 80,
                                   state,
                                   label0,
                                   GFX_ALIGN_CENTER,
                                   pScheme);

// Object will be drawn after creation
state = GFX_GOL_RADIOBUTTON_DRAW_STATE;

pRb[1] = GFX_GOL_RadioButtonCreate(ID_RADIOBUTTON2,
                                   255, 85, 310, 125,
                                   state,
                                   label1,
                                   GFX_ALIGN_CENTER,
                                   pScheme);

// Object will be drawn after creation
state = GFX_GOL_RADIOBUTTON_DRAW_STATE;

pRb[2] = GFX_GOL_RadioButtonCreate(ID_RADIOBUTTON3,
                                   255, 130, 310, 170,
                                   state,
```

```
        label2,  
        GFX_ALIGN_CENTER,  
        pScheme);  
  
    // draw the objects  
    while(GFX_GOL_ObjectListDraw() != GFX_STATUS_SUCCESS);
```

5.2.2.7.15.6 GFX_GOL_RadioButtonDraw Function

C

```
GFX_STATUS GFX_GOL_RadioButtonDraw(  
    void * pObject  
);
```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the [GFX_GOL_RADIOBUTTON](#) is drawn with the text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.15.7 GFX_GOL_RadioButtonTextAlignmentGet Function

C

```
GFX_ALIGNMENT GFX_GOL_RadioButtonTextAlignmentGet(  
    GFX_GOL_RADIOBUTTON * pObject  
);
```

Description

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The text alignment set in the object. See [GFX_ALIGNMENT](#) for more details.

Example

None.

5.2.2.7.15.8 GFX_GOL_RadioButtonTextAlignmentSet Function**C**

```
void GFX_GOL_RadioButtonTextAlignmentSet(  
    GFX_GOL_RADIOBUTTON * pObject,  
    GFX_ALIGNMENT align  
);
```

Description

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Returns

None.

Example

None.

5.2.2.7.15.9 GFX_GOL_RadioButtonTextSet Function**C**

```
void GFX_GOL_RadioButtonTextSet(  
    GFX_GOL_RADIOBUTTON * pObject,  
    GFX_XCHAR * pText  
);
```

Description

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

pText	pointer to the text string to be used.

Returns

None.

Example

```
GFX_XCHAR Label0[] = "ON";
GFX_XCHAR Label1[] = "OFF";
GFX_GOL_RADIOBUTTON GFX_GOL_RADIOBUTTON[2];

GFX_GOL_RadioButtonTextSet(GFX_GOL_RADIOBUTTON[0], Label0);
GFX_GOL_RadioButtonTextSet(GFX_GOL_RADIOBUTTON[1], Label1);
```

5.2.2.7.16 Round Dial Object Functions

5.2.2.7.16.1 GFX_GOL_RoundDialActionGet Function

C

```
uint16_t GFX_GOL_RoundDialActionGet(
    void * pObj,
    GFX_GOL_MESSAGE * pMsg
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen inputs.

Translated Message	Input Source	Events	Description
RD_MSG_CLOCKWISE	Touch Screen	EVENT_MOVE	If events occurs and the x,y position falls in the face of the Dial and moving in the clockwise rotation.
RD_MSG_CTR_CLOCKWISE	Touch Screen	EVENT_MOVE	If events occurs and the x,y position falls in the face of the Dial and moving in the counter clockwise rotation.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Preconditions

none

Parameters

Parameters	Description
pDia	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMsg	Pointer to the message struct containing the message from the user interface.

Returns

Returns the translated message depending on the received GOL message:

- RD_MSG_CLOCKWISE – Dial is moved in a clockwise direction.
- RD_MSG_CTR_CLOCKWISE – Dial is moved in a counter clockwise direction.
- GFX_GOL_MSG_INVALID – Dial is not affected

Side Effects

none

Example

```

void MyGOLMsg(GFX_GOL_MESSAGE *pMsg){
    GFX_OBJ_HEADER *pCurrentObj;
    uint16_t objMsg;

    if(pMsg->event == EVENT_INVALID)
        return;
    pCurrentObj = GFX_OBJ_ListGet();

    while(pCurrentObj != NULL){
        // Process only ROUND DIAL
        if(!GFX_OBJ_IsUpdatePending(pCurrentObj)){
            switch(pCurrentObj->type){
                case OBJ_ROUND DIAL:
                    objMsg = RdiaTranslateMsg((ROUND DIAL*)pCurrentObj, pMsg);
                    if(objMsg == OBJ_MSG_INVALID)
                        break;
                    if(GFX_GOL_MESSAGECallback(objMsg,pCurrentObj,pMsg))
                        RdiaMsgDefault(objMsg,(ROUND DIAL*)pCurrentObj);
                    break;
                default: break;
            }
        }
        pCurrentObj = pCurrentObj->pNxtObj;
    }
}

```

5.2.2.7.16.2 GFX_GOL_RoundDailActionSet Function**C**

```

void GFX_GOL_RoundDailActionSet(
    uint16_t translatedMsg,
    void * pObj,
    GFX_GOL_MESSAGE * pMsg
);

```

Description

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit Description
#####	#####	#####
RD_MSG_CLOCKWISE	Touch Screen	Set GFX_GOL_ROUND DIAL_ROT_CW , Dial will be redrawn with clockwise update. Set GFX_GOL_ROUND DIAL_DRAW
RD_MSG_CTR_CLOCKWISE	Touch Screen	Set GFX_GOL_ROUND DIAL_ROT_CCW , Dial will be redrawn with counter clockwise update. Set GFX_GOL_ROUND DIAL_DRAW

Preconditions

none

Parameters

Parameters	Description
translatedMsg	The translated message
pDia	The pointer to the object whose state will be modified
pMsg	The pointer to the GOL message

Returns

none

Side Effects

none

Example

See RdiaTranslateMsg() example.

5.2.2.7.16.3 GFX_GOL_RoundDialCreate Function**C**

```
GFX_GOL_ROUNDIAL * GFX_GOL_RoundDialCreate(
    uint16_t ID,
    uint16_t x,
    uint16_t y,
    uint16_t radius,
    uint16_t state,
    uint16_t res,
    uint16_t value,
    uint16_t max,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Preconditions

none

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
x	Location of the center of the dial in the x coordinate.
y	Location of the center of the dial in the y coordinate.
radius	Defines the radius of the dial.
state	Sets the initial state of the object.
res	Sets the resolution of the dial when rotating clockwise or counter clockwise.
value	Sets the initial value of the dial.
max	Sets the maximum value of the dial.
pScheme	Pointer to the style scheme used.

Returns

Returns the pointer to the object created.

Remarks

none

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_ROUNDIAL *pDial;
uint16_t state;

pScheme = GFX_GOL_SchemeCreate();
state = GFX_GOL_ROUNDIAL_DRAW;

// creates a dial at (50,50) x,y location, with an initial value
// of 50, a resolution of 2 and maximum value of 100.
pDial = GFX_GOL_RoundDial_Create(1,50,50,25,118,0, state, 2, 50, 100, pScheme);
// check if dial was created
if (pDial == NULL)
    return 0;

return 1;
```

5.2.2.7.16.4 GFX_GOL_RoundDialDraw Function**C**

```
uint16_t GFX_GOL_RoundDialDraw(
    void * pObj
);
```

Description

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the center (x,y) position and the radius parameters. The colors used are dependent on the state of the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Preconditions

Object must be created before this function is called.

Parameters

Parameters	Description
pDia	Pointer to the object

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Side Effects

none

5.2.2.7.16.5 RdiaCosine Function**C**

```
short RdiaCosine(
    short v
);
```

Description

This is function RdiaCosine.

5.2.2.7.16.6 RdiaSine Function**C**

```
short RdiaSine(
    short v
);
```

Description

This is function RdiaSine.

5.2.2.7.17 Scrollbar Object Functions**5.2.2.7.17.1 GFX_GOL_ScrollBarActionGet Function****C**

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_ScrollBarActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_SCROLLBAR_ACTION_INC	Touch Screen	EVENT_PRESS, EVENT_MOVE	If events occurs and the x,y position falls in the area of the slider and the slider position is to the LEFT of the x,y position for a horizontal slider or BELOW the x,y position for a vertical slider.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_UP_PRESSED or SCAN_LEFT_PRESSED .
GFX_GOL_SCROLLBAR_ACTION_DEC	Touch Screen	EVENT_PRESS, EVENT_MOVE	If events occurs and the x,y position falls in the area of the slider and the slider position is to the RIGHT of the x,y position for a horizontal slider or ABOVE the x,y position for a vertical slider.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_DOWN_PRESSED or SCAN_RIGHT_PRESSED .
GFX_GOL_OBJECT_ACTION_PASSIVE	Touch Screen	EVENT_RELEASE	If events occurs and the x,y position falls in the area of the slider.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_SCROLLBAR_ACTION_INC â€œ Increment scroll bar position.
- GFX_GOL_SCROLLBAR_ACTION_DEC â€œ Decrement scroll bar position.
- GFX_GOL_OBJECT_ACTION_PASSIVE â€œ Object bar is not affected
- GFX_GOL_OBJECT_ACTION_INVALID â€œ Object is not affected

Example

None.

5.2.2.7.17.2 GFX_GOL_ScrollBarActionSet Function**C**

```
void GFX_GOL_ScrollBarActionSet(
    GFX_GOL_TRANSLATED_ACTION translatedMsg,
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_SCROLLBAR_ACTION_INC	Touch Screen,	Set GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE	Scroll Bar will be redrawn with thumb in the incremented position.
	Keyboard		
GFX_GOL_SCROLLBAR_ACTION_DEC	Touch Screen,	Set GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE	Scroll Bar will be redrawn with thumb in the decremented position.
	Keyboard		

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Returns

None.

Example

None.

5.2.2.7.17.3 GFX_GOL_ScrollBarCreate Function**C**

```
GFX_GOL_SCROLLBAR * GFX_GOL_ScrollBarCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    uint16_t range,
    uint16_t page,
    uint16_t pos,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GFX_GOL_SCROLLBAR](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The object can be configured as a scroll bar or a slider. Use the state bit [GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE](#) to enable the usage of the object as a slider. If this state bit is not enabled, the object is set up as a scroll bar.

The object can also be configured with vertical orientation. Use the state bit [GFX_GOL_SCROLLBAR_VERTICAL_STATE](#) to set up the object with vertical orientation. If this state bit is not set, the object is used with horizontal orientation.

The behavior of [GFX_GOL_ScrollBarCreate\(\)](#) will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- page is set to zero.
- range is set to zero.
- page > range.

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
range	This specifies the maximum value of the slider when the thumb is on the rightmost position for a horizontal orientation and bottom most position for the vertical orientation. Minimum value is always at zero.
page	This is the incremental change of the slider when user action requests to move the slider thumb. This value is added or subtracted to the current position of the thumb.
pos	This defines the initial position of the thumb.
pScheme	The pointer to the style scheme used for the Object. Set to NULL if default style scheme is used.

Returns

Pointer to the newly created object.

Example

```

GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_SCROLLBAR *scrollBarArray[3];
uint16_t          state;

pScheme = GFX_GOL_ObjectSchemeCreate();

// create a slider with
// range = [0 - 50000]
// delta = 500
// initial position = 25000
state = GFX_GOL_SCROLLBAR_DRAW_STATE;
scrollBarArray[0] = GFX_GOL_ScrollBarCreate( 5,
                                             150, 145, 285, 181,
                                             state,
                                             50000, 500, 25000,
                                             pScheme);

if (slider[0] == NULL)
    return 0;

// create a slider with
// range = [0 - 100]
// delta = 20
// initial position = 0
state = GFX_GOL_SCROLLBAR_DRAW_STATE |
        GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE;
scrollBarArray[1] = GFX_GOL_ScrollBarCreate( 6,
                                             150, 190, 285, 220,
                                             state,
                                             100, 20, 0,
                                             pScheme);

if (slider[1] == NULL)
    return 0;

// create a vertical scroll bars with
// range = [0 - 30]
// delta = 2
// initial position = 20

```

```

state = GFX_GOL_SCROLLBAR_DRAW_STATE |
        GFX_GOL_SCROLLBAR_VERTICAL_STATE;
scrollBarArray[2] = GFX_GOL_ScrollBarCreate( 7,
                                             120, 145, 140, 220,
                                             state,
                                             30, 2, 20,
                                             pScheme);

if (slider[2] == NULL)
    return 0;

// draw the sliders
while (GFX_GOL_ObjectListDraw() == 0);

return 1;

```

5.2.2.7.17.4 GFX_GOL_ScrollBarDraw Function

C

```

GFX_STATUS GFX_GOL_ScrollBarDraw(
    void * pobject
);

```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the [GFX_GOL_SCROLLBAR](#) is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.17.5 GFX_GOL_ScrollBarPageGet Function

C

```

uint16_t GFX_GOL_ScrollBarPageGet(
    GFX_GOL_SCROLLBAR * pObject
);

```

Description

This function returns the page size of the object. Page size defines the delta change of the thumb position when incremented via [GFX_GOL_ScrollBarPositionIncrement\(\)](#) or decremented via [GFX_GOL_ScrollBarPositionDecrement\(\)](#). Page size minimum value is 1. Maximum value is range/2.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The page size of the object.

Example

```
uint16_t page;
GFX_GOL_SCROLLBAR *pScrollBar;

// assume pScrollBar is initialized to a scroll bar in memory
page = GFX_GOL_ScrollBarPageGet(pScrollBar);
```

5.2.2.7.17.6 GFX_GOL_ScrollBarPageSet Function**C**

```
void GFX_GOL_ScrollBarPageSet(
    GFX_GOL_SCROLLBAR * pObject,
    uint16_t page
);
```

Description

This function sets the page size of the object. Page size defines the delta change of the thumb position when incremented via [GFX_GOL_ScrollBarPositionIncrement\(\)](#) or decremented via [GFX_GOL_ScrollBarPositionDecrement\(\)](#). Page size minimum value is 1. Maximum value is range/2.

Modifying the page size at run time may require a redraw of the object to show the visual effect of the change.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
page	value of the page size of the object.

Returns

None.

Example

See [GFX_GOL_ScrollBarPositionIncrement\(\)](#) for code example.

5.2.2.7.17.7 GFX_GOL_ScrollBarPositionDecrement Function

C

```
void GFX_GOL_ScrollBarPositionDecrement(
    GFX_GOL_SCROLLBAR * pObj
);
```

Description

This function decrements the scroll bar position by the delta change (page) value set. Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObj	pointer to the object.

Returns

None.

Example

```
void ControlSpeed(  GFX_GOL_SCROLLBAR* pObj,
                    int  setSpeed,
                    int  curSpeed)
{
    // set page size to 1
    GFX_GOL_ScrollBarPageSet(pObj, 1);

    if (setSpeed < curSpeed)
    {
        while(GFX_GOL_ScrollBarPositionGet(pObj) < SetSpeed)
        {
            // increment by 1
            GFX_GOL_ScrollBarPositionIncrement(pObj);
        }
    }
    else if (setSpeed > curSpeed)
    {
        while(GFX_GOL_ScrollBarPositionGet(pObj) > SetSpeed)
        {
            // decrement by 1
            GFX_GOL_ScrollBarPositionDecrement(pObj);
        }
    }
}
```

5.2.2.7.17.8 GFX_GOL_ScrollBarPositionGet Function

C

```
uint16_t GFX_GOL_ScrollBarPositionGet(
    GFX_GOL_SCROLLBAR * pObj
);
```

Description

This function returns the current position of the scroll bar thumb. The thumb is the rectangular area that slides left or right (for horizontal orientation) or slides up or down (for vertical orientation).

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The current position of the scroll bar thumb.

Example

```
#define MAXVALUE 100;

GFX_GOL_SCROLLBAR *pScrollBar;
uint32_t ctr = 0;

// create scroll bar here and initialize parameters
pScrollBar = GFX_GOL_ScrollBarCreate(...)
GFX_GOL_ObjectStateSet(pScrollBar, GFX_GOL_SCROLLBAR_DRAW_STATE);

// draw the scroll bar
GFX_GOL_ObjectListDraw();

// a routine that updates the position of the thumb through some
// conditions
while("some condition")
{
    GFX_GOL_ScrollBarPositionSet(pScrollBar, ctr);
    GFX_GOL_ObjectStateSet( pScrollBar,
                           GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE);

    // update the screen
    GFX_GOL_ObjectListDraw();

    // update ctr here
    ctr = "some source of value";
}

if (GFX_GOL_ScrollBarPositionGet(pScrollBar) > MAXVALUE)
    return 0;
else
    "do something else"
```

5.2.2.7.17.9 GFX_GOL_ScrollBarPositionIncrement Function**C**

```
void GFX_GOL_ScrollBarPositionIncrement(
    GFX_GOL_SCROLLBAR * pObject
);
```

Description

This function increments the scroll bar position by the delta change (page) value set. Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

```
void ControlSpeed( GFX_GOL_SCROLLBAR* pObj,
                  int setSpeed,
                  int curSpeed)
{
    // set page size to 1
    GFX_GOL_ScrollBarPageSet(pObj, 1);

    if (setSpeed < curSpeed)
    {
        while(GFX_GOL_ScrollBarPositionGet(pObj) < SetSpeed)
            GFX_GOL_ScrollBarPositionIncrement(pObj); // increment by 1
    }
    else if (setSpeed > curSpeed)
    {
        while(GFX_GOL_ScrollBarPositionGet(pObj) > SetSpeed)
            GFX_GOL_ScrollBarPositionDecrement(pObj); // decrement by 1
    }
}
```

5.2.2.7.17.10 GFX_GOL_ScrollBarPositionSet Function**C**

```
void GFX_GOL_ScrollBarPositionSet(
    GFX_GOL_SCROLLBAR * pObj,
    uint16_t position
);
```

Description

This function sets the position of the scroll bar thumb. The thumb is the rectangular area that slides left or right (for horizontal orientation) or slides up or down (for vertical orientation).

The value used for the position should be within the range set for the object.

Function will have an undefined behavior if the position is outside the range.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
position	the new position of the scroll bar thumb.

Returns

None.

Example

```
GFX_GOL_SCROLLBAR *pScrollBar;
uint16_t ctr = 0;
```

```
// create slider here and initialize parameters
GFX_GOL_ObjectStateSet(pScrollBar, GFX_GOL_SCROLLBAR_DRAW_STATE);
GFX_GOL_ObjectListDraw();

while("some condition")
{
    GFX_GOL_ScrollBarPositionSet(pScrollBar, ctr);
    GFX_GOL_ObjectStateSet( pScrollBar,
                           GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE);

    // redraw the scroll bar
    GFX_GOL_ObjectListDraw();

    // update ctr here
    ctr = "some source of value";
}
```

5.2.2.7.17.11 GFX_GOL_ScrollBarRangeGet Function

C

```
uint16_t GFX_GOL_ScrollBarRangeGet(
    GFX_GOL_SCROLLBAR * pObject
);
```

Description

This function returns the range of the thumb of the scroll bar.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The range of the scroll bar.

Example

```
GFX_GOL_SCROLLBAR *pSld;
uint16_t range;

range = GFX_GOL_ScrollBarRangeGet(pSld);
```

5.2.2.7.17.12 GFX_GOL_ScrollBarRangeSet Function

C

```
void GFX_GOL_ScrollBarRangeSet(
    GFX_GOL_SCROLLBAR * pObject,
    uint16_t range
);
```

Description

This function sets the range of the thumb of the scroll bar. When the range is modified, object must be completely redrawn to reflect the change.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
range	new range of the scroll bar.

Returns

None.

Example

```
GFX_GOL_SCROLLBAR *pSld;

GFX_GOL_ScrollBarRangeSet(pSld, 100);

// to completely redraw the object when
// GFX_GOL_ObjectListDraw() is executed.
GFX_GOL_ObjectStateSet(pSld, SLD_DRAW);
```

5.2.2.7.18 Static Text Object Functions**5.2.2.7.18.1 GFX_GOL_StaticTextActionGet Function****C**

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_StaticTextActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen inputs.

Translated Message	Input Source	Events	Description
GFX_GOL_STATICTEXT_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the static text.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_STATICTEXT_ACTION_SELECTED â€œ Object is selected
- GFX_GOL_OBJECT_ACTION_INVALID â€œ Object is not affected

Example

None.

5.2.2.7.18.2 GFX_GOL_StaticTextAlignmentGet Function

```
C
GFX_ALIGNMENT GFX_GOL_StaticTextAlignmentGet(
    GFX_GOL_STATICTEXT * pObject
);
```

Description

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The text alignment set in the object. See [GFX_ALIGNMENT](#) for more details.

Example

None.

5.2.2.7.18.3 GFX_GOL_StaticTextAlignmentSet Function

```
C
void GFX_GOL_StaticTextAlignmentSet(
    GFX_GOL_STATICTEXT * pObject,
    GFX_ALIGNMENT align
);
```

Description

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Returns

None.

Example

None.

5.2.2.7.18.4 GFX_GOL_StaticTextCreate Function

C

```
GFX_GOL_STATICTEXT * GFX_GOL_StaticTextCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    GFX_XCHAR * pText,
    GFX_ALIGNMENT alignment,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This function creates a [GFX_GOL_STATICTEXT](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The text can be configured to have text aligned. See [GFX_ALIGNMENT](#) for details. Text can also have multiple lines by inserting the new line character to the text string supplied to the object. Any string that exceeds the dimension of the object will be clipped.

When the object is used with no background, application must manage the object when text is modified and redrawn. i.e. the previous text must be removed. Use [GFX_GOL_STATICTEXT_NOBACKGROUND_STATE](#) state bit to disable the background.

The behavior of [GFX_GOL_StaticTextCreate\(\)](#) will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- pText is an unterminated string

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```
// assume pScheme is initialized
GFX_GOL_OBJ_SCHEME *pScheme;
```

```
GFX_GOL_STATICTEXT *pSt;  
  
pScheme = GOLCreateScheme();  
StCreate(  
    ID_STATICTEXT1,           // ID  
    30,80,235,160,           // dimension  
    GFX_GOL_STATICTEXT_DRAW_STATE, // draw the object  
    "Static Textn Example",    // 2 lines of text  
    GFX_ALIGN_CENTER,        // align text on the center  
    pScheme);                // use given scheme
```

5.2.2.7.18.5 GFX_GOL_StaticTextDraw Function

C

```
GFX_STATUS GFX_GOL_StaticTextDraw(  
    void * pObject  
);
```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.18.6 GFX_GOL_StaticTextGet Function

C

```
GFX_XCHAR * GFX_GOL_StaticTextGet(  
    GFX_GOL_STATICTEXT * pObject  
);
```

Description

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to text string.

Example

```
GFX_XCHAR *pChar;
GFX_GOL_STATICTEXT OBJECT_ARRAY[2];

pChar = GFX_GOL_StaticTextGet(OBJECT_ARRAY[0]);
```

5.2.2.7.18.7 GFX_GOL_StaticTextSet Function**C**

```
void GFX_GOL_StaticTextSet(
    GFX_GOL_STATICTEXT * pObject,
    GFX_XCHAR * pText
);
```

Description

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Returns

None.

Example

None.

5.2.2.7.19 Text Entry Object Functions**5.2.2.7.19.1 GFX_GOL_TextEntryActionGet Function****C**

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_TextEntryActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_TEXTENTRY_ACTION_PRESS	Touch Screen	EVENT_PRESS, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed.
GFX_GOL_TEXTENTRY_ACTION_RELEASED	Touch Screen	EVENT_MOVE	If the event occurs and the x,y position falls outside the face of one of the keys of the object while the key is pressed.
GFX_GOL_TEXTENTRY_ACTION_RELEASED	Touch Screen	EVENT_RELEASE	If the event occurs and the x,y position falls does not falls inside any of the faces of the keys of the object.
GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with no commands.
GFX_GOL_TEXTENTRY_ACTION_DELETE	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with delete command.
GFX_GOL_TEXTENTRY_ACTION_SPACE	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with space command.
GFX_GOL_TEXTENTRY_ACTION_ENTER	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with enter command.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_TEXTENTRY_ACTION_PRESS - A key is pressed
- GFX_GOL_TEXTENTRY_ACTION_RELEASED - A key was released (generic for keys with no commands or characters assigned)
- GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR - A key was released with character assigned
- GFX_GOL_TEXTENTRY_ACTION_DELETE - A key was released with delete command assigned
- GFX_GOL_TEXTENTRY_ACTION_SPACE - A key was released with space command assigned
- GFX_GOL_TEXTENTRY_ACTION_ENTER - A key was released with enter command assigned
- GFX_GOL_OBJECT_ACTION_INVALID - Text Entry is not affected

Example

None.

5.2.2.7.19.2 GFX_GOL_TextEntryActionSet Function**C**

```
void GFX_GOL_TextEntryActionSet(
    uint16_t translatedMsg,
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined `GFX_GOL_MESSAGE_CALLBACK_FUNC`. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message Input Source	Set/Clear State Bit	Description
GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR Touch Screen,	Set GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE,	Add a character in the buffer and update the text displayed.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	
GFX_GOL_TEXTENTRY_ACTION_SPACE Touch Screen,	Set GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE,	Insert a space character in the buffer and update the text displayed.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	
GFX_GOL_TEXTENTRY_ACTION_DELETE Touch Screen,	Set GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE,	Delete the most recent character in the buffer and update the text displayed.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	
GFX_GOL_TEXTENTRY_ACTION_ENTER Touch Screen,	Set GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE,	User can define the use of this event in the message callback. Object will just update the key.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	
GFX_GOL_TEXTENTRY_ACTION_RELEASED Touch Screen,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	A Key in the object will be redrawn in the unpressed state.
	Set GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE	
GFX_GOL_TEXTENTRY_ACTION_PRESSED Touch Screen,	Set GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	A Key in the object will be redrawn in the pressed state.

GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE		

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Returns

None.

Example

None.

5.2.2.7.19.3 GFX_GOL_TextEntryBufferClear Function**C**

```
void GFX_GOL_TextEntryBufferClear(  
    GFX_GOL_TEXTENTRY * pObject  
)  
;
```

Description

This function will clear the data in the display. Object must be redrawn to reflect the change in the buffer. Use the drawing state bit GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE to update the text on the screen.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

None.

5.2.2.7.19.4 GFX_GOL_TextEntryBufferGet Function**C**

```
GFX_XCHAR * GFX_GOL_TextEntryBufferGet(  
    GFX_GOL_TEXTENTRY * pObject  
)  
;
```

Description

This function returns the buffer used to display text.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The pointer to the text buffer used to display text.

Example

None.

5.2.2.7.19.5 GFX_GOL_TextEntryBufferSet Function**C**

```
void GFX_GOL_TextEntryBufferSet(  
    GFX_GOL_TEXTENTRY * pObject,  
    GFX_XCHAR * pText,  
    uint16_t MaxSize  
);
```

Description

This function sets the buffer used to display text. If the buffer is initialized with a string, the string must be a null terminated string. If the string length is greater than MaxSize, string will be truncated to MaxSize. pText must point to a valid memory location with size equal to MaxSize. The total number of characters that will be displayed on the object will be MaxSize - 1 since the last character will be the string terminator character.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the new text buffer to be displayed.
maxSize	maximum length of the new buffer to be used.

Returns

None.

Example

None.

5.2.2.7.19.6 GFX_GOL_TextEntryCharAdd Function**C**

```
void GFX_GOL_TextEntryCharAdd(  
    GFX_GOL_TEXTENTRY * pObject  
);
```

Description

This function will insert a character to the end of the buffer. Drawing states GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE or GFX_GOL_TEXTENTRY_DRAW_STATE must be set to see the effect of the addition.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

None.

5.2.2.7.19.7 GFX_GOL_TextEntryCreate Function**C**

```
GFX_GOL_TEXTENTRY * GFX_GOL_TextEntryCreate(  
    uint16_t ID,  
    uint16_t left,  
    uint16_t top,  
    uint16_t right,  
    uint16_t bottom,  
    uint16_t state,  
    uint16_t horizontalKeys,  
    uint16_t verticalKeys,  
    GFX_XCHAR * pText[],  
    GFX_XCHAR * pBuffer,  
    GFX_ALIGNMENT alignment,  
    uint16_t bufferLength,  
    GFX_RESOURCE_HDR * pDisplayFont,  
    GFX_GOL_OBJ_SCHEME * pScheme  
);
```

Description

This function creates a [GFX_GOL_TEXTENTRY](#) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of `GFX_GOL_TextEntryCreate()` will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a [GFX_GOL_OBJ_SCHEME](#)
- horizontal key or vertical key count is 0
- pText is an unterminated string
- pBuffer is initialized to an allocated memory.

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.

left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
radius	Radius of the rounded edge. When using gradient buttons and radius != 0, emboss size <= radius. If this is not met, the the GFX_GOL_BUTTON face will not have gradient effect.
state	Sets the initial state of the object.
horizontalKeys	Number of horizontal keys
verticalKeys	Number of vertical keys
pText	array of pointer to the custom "text" assigned by the user.
pBuffer	pointer to the buffer that holds the text to be displayed.
alignment	text alignment of the text used in the object.
bufferLength	length of the buffer assigned by the user. The choice of the length should include the string null terminator.
For example	if the bufferLength is set to 3, only two characters can be shown on the object since the last character will be the string terminator character.
pDisplayFont	pointer to the font image to be used on the edit box section of the object.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

TODO:Add example code.

5.2.2.7.19.8 GFX_GOL_TextEntryDraw Function**C**

```
GFX_STATUS GFX_GOL_TextEntryDraw(
    void * pObject
);
```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not

yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.19.9 GFX_GOL_TextEntryKeyCommandGet Function

C

```
GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE GFX_GOL_TextEntryKeyCommandGet(  
    GFX_GOL_TEXTENTRY * pObject,  
    uint16_t index  
);
```

Description

This function will return the currently assigned command to the key with the given index. (See [GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE](#))

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
index	index or position of the key.

Returns

Command assigned to the key (See [GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE](#)).

Example

None.

5.2.2.7.19.10 GFX_GOL_TextEntryKeyCommandSet Function

C

```
bool GFX_GOL_TextEntryKeyCommandSet(  
    GFX_GOL_TEXTENTRY * pObject,  
    uint16_t index,  
    GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE command  
);
```

Description

This function will assign a command to a key with the given index. (See [GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE](#))

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
index	index or position of the key.
command	command assigned for the key.

Returns

TRUE - if the assignment was a success. FALSE - if the assignment was not successful.

Example

None.

5.2.2.7.19.11 GFX_GOL_TextEntryKeyIsPressed Function**C**

```
bool GFX_GOL_TextEntryKeyIsPressed(  
    GFX_GOL_TEXTENTRY * pObject,  
    uint16_t index  
);
```

Description

This function will test if a key given by its index in the object is pressed.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
index	index or position of key that is being tested.

Returns

TRUE - if the key is pressed. FALSE - if the key is not pressed.

Example

None.

5.2.2.7.19.12 GFX_GOL_TextEntryKeyListCreate Function**C**

```
KEYMEMBER * GFX_GOL_TextEntryKeyListCreate(  
    GFX_GOL_TEXTENTRY * pObject,  
    GFX_XCHAR * pText[]  
);
```

Description

This function will create the list of KEYMEMBERS that holds the information on each key. The number of keys is determined by the equation ($\text{keycount} = \text{verticalKeys} * \text{horizontalKeys}$). The object creates the information holder for each key automatically and assigns each entry in the *pText[] array with the first entry automatically assigned to the key with an index of 1.

The number of entries to *pText[] must be at least equal to keycount. The last key is assigned with an index of $\text{keycount}-1$.

No checking is performed on the length of *pText[] entries to match ($\text{verticalKeys} * \text{horizontalKeys}$).

The function behavior is undefined if the pText[] array is less than the keycount value.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text defined by the application.

Returns

Returns the pointer to the newly created [KEYMEMBER](#) list. A NULL is returned if the list is not created successfully.

Example

None.

5.2.2.7.19.13 GFX_GOL_TextEntryKeyMemberListDelete Function**C**

```
void GFX_GOL_TextEntryKeyMemberListDelete(  
    void * pObject  
);
```

Description

This function will delete the [KEYMEMBER](#) list assigned to the object from memory. Pointer to the [KEYMEMBER](#) list is then initialized to NULL. All memory resources allocated to the key member list is freed.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

None.

5.2.2.7.19.14 GFX_GOL_TextEntryKeyTextSet Function**C**

```
bool GFX_GOL_TextEntryKeyTextSet(  
    GFX_GOL_TEXTENTRY * pObject,  
    uint16_t index,  
    GFX_XCHAR * pText  
);
```

Description

This function will set the text assigned to a key with the given index.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
index	index or position of the key.
pText	pointer to the text that will be assigned to the key.

Returns

TRUE - if the assignment was a success. FALSE - if the assignment was not successful.

Example

None.

5.2.2.7.19.15 GFX_GOL_TextEntryLastCharDelete Function**C**

```
void GFX_GOL_TextEntryLastCharDelete(  
    GFX_GOL_TEXTENTRY * pObject  
);
```

Description

This function will remove the last character of the buffer and replace it with a string terminator. Drawing states `GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE` or `GFX_GOL_TEXTENTRY_DRAW_STATE` must be set to see the effect of the addition.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

None.

5.2.2.7.19.16 GFX_GOL_TextEntrySpaceCharAdd Function**C**

```
void GFX_GOL_TextEntrySpaceCharAdd(  
    GFX_GOL_TEXTENTRY * pObject  
);
```

Description

This function will insert a space character to the end of the buffer. Drawing states `GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE` or `GFX_GOL_TEXTENTRY_DRAW_STATE` must be set to see the effect of the addition.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

None.

Example

None.

5.2.2.7.20 Window Object Functions

5.2.2.7.20.1 GFX_GOL_WindowActionGet Function

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_WindowActionGet(
    void * pObject,
    GFX_GOL_MESSAGE * pMessage
);
```

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source Events Description
GFX_GOL_WINDOW_ACTION_TITLE	Touch Screen EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE If events occurs and the x,y position falls in the TITLE area of the window
GFX_GOL_WINDOW_ACTION_CLIENT	Touch Screen EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE If events occurs and the x,y position falls in the CLIENT area of the window
GFX_GOL_OBJECT_ACTION_INVALID	Any Any If the message did not affect the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Returns

- GFX_GOL_WINDOW_ACTION_CLIENT // Window client area selected action ID.
- GFX_GOL_WINDOW_ACTION_TITLE // Window title bar selected action ID.

Example

None.

5.2.2.7.20.2 GFX_GOL_WindowCreate Function

C

```
GFX_GOL_WINDOW * GFX_GOL_WindowCreate(
    uint16_t ID,
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t state,
    void * pBitmap,
    GFX_XCHAR * pText,
    GFX_ALIGNMENT alignment,
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

```
);
```

Description

This function creates a `GFX_GOL_WINDOW` object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

Preconditions

None.

Parameters

Parameters	Description
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pImage	Pointer to the image used on the face of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Returns

Pointer to the newly created object.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_WINDOW *pWindow;
GFX_GOL_WINDOW_STATE state;

// assume pScheme is initialized to a scheme in memory.
state = GFX_GOL_WINDOW_DRAW_STATE;

pWindow = GFX_GOL_WindowCreate(1,           // ID
                                0,0,GFX_Primitive_MaxXGet(),GFX_Primitive_MaxYGet(), // whole screen dimension
                                state,       // set state to draw all
                                (char*)myIcon, // icon
                                "Place Title Here.", // text
                                NULL);         // use default GOL scheme

if (pWindow == NULL)
    return 0;

return 1;
```

5.2.2.7.20.3 GFX_GOL_WindowDraw Function

C

```
GFX_STATUS GFX_GOL_WindowDraw(
    void * pobject
);
```

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by

the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the [GFX_GOL_WINDOW](#) is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call [GFX_GOL_ObjectListDraw\(\)](#) to allow the Graphics Library to manage all object rendering. See [GFX_GOL_ObjectListDraw\(\)](#) for more information on object rendering.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Example

None.

5.2.2.7.20.4 GFX_GOL_WindowTextAlignmentGet Function

C

```
GFX_ALIGNMENT GFX_GOL_WindowTextAlignmentGet(  
    GFX_GOL_WINDOW * pObject  
);
```

Description

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The text alignment set in the object. See [GFX_ALIGNMENT](#) for more details.

Example

None.

5.2.2.7.20.5 GFX_GOL_WindowTextAlignmentSet Function

C

```
void GFX_GOL_WindowTextAlignmentSet(  
    GFX_GOL_WINDOW * pObject,  
    GFX_ALIGNMENT align
```

```
);
```

Description

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Returns

None.

Example

None.

5.2.2.7.20.6 GFX_GOL_WindowTextSet Function

C

```
void GFX_GOL_WindowTextSet(  
    GFX_GOL_WINDOW * pObject,  
    GFX_XCHAR * pText  
);
```

Description

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Returns

None.

Example

```
GFX_XCHAR Label0[] = ?ON?;  
GFX_XCHAR Label1[] = ?OFF?;  
GFX_GOL_WINDOW pWindow;  
  
GFX_GOL_WindowTextSet(pWindow, Label0);  
GFX_GOL_WindowTextSet(pWindow, Label1);
```

5.2.2.7.21 Draw Functions

5.2.2.7.21.1 ClearPaletteChangeError Function

C

```
void ClearPaletteChangeError();
```

Description

Clears the Palette change error status

Preconditions

none

Returns

none

Side Effects

none

5.2.2.7.21.2 DisablePalette Function

C

```
void DisablePalette();
```

Description

Disables the Palette mode.

Preconditions

none

Returns

none

5.2.2.7.21.3 DrawArc Function

C

```
uint16_t DrawArc(
    int16_t cx,
    int16_t cy,
    int16_t r1,
    int16_t r2,
    int16_t startAngle,
    int16_t endAngle
);
```

Description

This renders an arc with from startAngle to endAngle with the thickness of r2-r1. The function returns 1 when the arc is rendered successfully and returns a 0 when it is not yet finished. The next call to the function will continue the rendering.

Preconditions

none

Parameters

Parameters	Description
cx	the location of the center of the arc in the x direction.
cy	the location of the center of the arc in the y direction.

r1	the smaller radius of the arc.
r2	the larger radius of the arc.
startAngle	start angle of the arc.
endAngle	end angle of the arc.

Returns

Returns 1 if the rendering is done, 0 if not yet done.

Side Effects

none

Remarks

none

5.2.2.7.21.4 EnablePalette Function

C

```
void EnablePalette();
```

Description

Enables the Palette mode.

Preconditions

none

Returns

none

5.2.2.7.21.5 GetPaletteChangeError Function

C

```
BYTE GetPaletteChangeError();
```

Description

Returns the Palette change error status

Preconditions

none

Returns

Returns the palette change status. 1 - If the palette change error occurred 0 - If no palette change error occurred

Side Effects

none

5.2.2.7.21.6 GFX_AlphaBlendingValueGet Function

C

```
uint16_t GFX_AlphaBlendingValueGet();
```

Description

This function returns the current alpha value set for alpha blending rendering. See [GFX_AlphaBlendingValueSet\(\)](#) for details of alpha blending values.

Preconditions

None.

Returns

The alpha blending value.

Example

TODO:Add example code

5.2.2.7.21.7 GFX_AlphaBlendingValueSet Function**C**

```
GFX_STATUS GFX_AlphaBlendingValueSet(  
    uint16_t alpha  
);
```

Description

This function sets the alpha value for alpha blending rendering. Accepted values are dependent on the used alpha blending routines at build time. When using the software routines in the Primitive Layer, accepted values are 0, 25, 50, 75 and 100.

If using a specific implementation in the display driver used, implementation may support the full range (0-100). When this is the case, refer to the driver alpha blending solution for details.

Function operation will ignore unsupported values of alpha.

Preconditions

None.

Parameters

Parameters	Description
alpha	Defines the alpha blending percentage to be used for alpha blending routines. Accepted values are dependent on the alpha blending routines used. For Primitive Layer

5.2.2.7.21.8 GFX_AlphaBlendWindow Function**C**

```
GFX_STATUS GFX_AlphaBlendWindow(  
    GFX_ALPHA_PARAMS * alphaParams,  
    uint16_t width,  
    uint16_t height,  
    uint8_t alphaPercent  
);
```

Description

None

Preconditions

None

Parameters

Parameters	Description
None	None

Returns

None

Example

None

5.2.2.7.21.9 GFX_BackgroundColorGet Function**C**

```
GFX_COLOR GFX_BackgroundColorGet();
```

Description

This function returns the color used in the current background.

Preconditions

None.

Returns

The color used in the current background.

Example

None.

5.2.2.7.21.10 GFX_BackgroundImageGet Function**C**

```
GFX_RESOURCE_HDR * GFX_BackgroundImageGet();
```

Description

This function returns the pointer to the image used in the current background.

Preconditions

None.

Returns

The pointer to the image used in the current background.

Example

None.

5.2.2.7.21.11 GFX_BackgroundImageLeftGet Function**C**

```
uint16_t GFX_BackgroundImageLeftGet();
```

Description

This function returns the horizontal starting position of the current background. This position defines the x position of the upper left corner of the background.

Preconditions

None.

Returns

The horizontal starting position of the current background.

Example

None.

5.2.2.7.21.12 GFX_BackgroundImageTopGet Function

C

```
uint16_t GFX_BackgroundImageTopGet();
```

Description

This function returns the vertical starting position of the current background. This position defines the y position of the upper left corner of the background.

Preconditions

None.

Returns

The vertical starting position of the current background.

Example

None.

5.2.2.7.21.13 GFX_BackgroundSet Function

C

```
void GFX_BackgroundSet(
    uint16_t left,
    uint16_t top,
    GFX_RESOURCE_HDR * pImage,
    GFX_COLOR color
);
```

Description

This function sets the background information. Note that the width and height of the background is only needed when the background is an image. When the background is a color (image pointer is NULL), the width and height is not needed here since the purpose of the background information is to record only the color used.

Preconditions

None.

Parameters

Parameters	Description
left	Horizontal starting position of the background.
top	Vertical starting position of the background.
pImage	Pointer to the background image used. Set this to NULL if not used.
color	If the background image is NULL, background is assumed to be this color.

Returns

None.

Example

None.

5.2.2.7.21.14 GFX_BackgroundTypeGet Function**C**

```
GFX_BACKGROUND_TYPE GFX_BackgroundTypeGet();
```

Description

This function returns the type of the current background.

Preconditions

None.

Returns

The type of the current background set.

Example

None.

5.2.2.7.21.15 GFX_BackgroundTypeSet Function**C**

```
void GFX_BackgroundTypeSet(  
    GFX_BACKGROUND_TYPE type  
);
```

Description

This function sets the background type.

Preconditions

None.

Parameters

Parameters	Description
type	Type of background that will be used.

Returns

None.

Example

None.

5.2.2.7.21.16 GFX_BarAlphaDraw Function**C**

```
uint16_t GFX_BarAlphaDraw(  
    uint16_t left,  
    uint16_t top,  
    uint16_t right,  
    uint16_t bottom  
);
```


Description

This function renders an alpha blended bar shape with the currently set color and the currently set background type (see [GFX_BACKGROUND_TYPE](#)).

The parameters left, top, right bottom will define the shape dimension.

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top or right,bottom falls outside the frame buffer.
- Colors are not set before this function is called.
- When right < left
- When bottom < top
- When pixel locations defined by left, top and/or right, bottom are not on the frame buffer.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Alpha blending value must be set by [GFX_AlphaBlendingValueSet\(\)](#).

Parameters

Parameters	Description
left	defines the left most pixel of the shape.
top	defines the top most pixel of the shape.
right	defines the right most pixel of the shape.
bottom	defines the bottom most pixel of the shape.

Returns

TBA (see [GFX_STATUS](#)).

Example

```
// assume RED is a macro that define GFX_COLOR types
GFX_STATUS status;
// assume BackGroundImage is a valid image already draw
// on the screen
GFX_RESOURCE_HDR *pMyBackgroundImage = &BackGroundImage;

// render an alpha blended bar with
// the current contents of the frame buffer
GFX_AlphaBlendingValueSet(50);
GFX_BackgroundTypeSet(GFX_BACKGROUND_DISPLAY_BUFFER);
GFX_ColorSet(RED);
status = GFX_BarAlphaDraw(10, 110, 100, 200);

// render an alpha blended bar with a background image
GFX_AlphaBlendingValueSet(75);
GFX_FillStyleSet(GFX_FILL_STYLE_ALPHA_COLOR);

// color value here has no effect since the background set
// is type GFX_BACKGROUND_IMAGE
GFX_BackgroundSet(0, 0, pMyBackgroundImage, 0);

GFX_BackgroundTypeSet(GFX_BACKGROUND_IMAGE);
GFX_ColorSet(RED);
status = GFX_BarAlphaDraw(10, 110, 100, 200);
```

5.2.2.7.21.17 GFX_BarDraw Function**C**

GFX_STATUS **GFX_BarDraw**(

```

uint16_t left,
uint16_t top,
uint16_t right,
uint16_t bottom
);

```

Description

This function renders a bar shape with:

- solid color - when the fill style is set to `GFX_FILL_STYLE_COLOR`
- alpha blended fill - when the fill style is set to `GFX_FILL_STYLE_ALPHA_COLOR`.

Any other selected fill style will be ignored and will assume a solid color fill will be used. The parameters left, top, right bottom will define the shape dimension.

When fill style is set to `GFX_FILL_STYLE_ALPHA_COLOR`, the bar can also be rendered with an option to select the type of background.

Preconditions

Fill style must be set by [GFX_FillStyleSet\(\)](#) when alpha blended fill is desired. Color must be set by [GFX_ColorSet\(\)](#).

Parameters

Parameters	Description
left	defines the left most pixel of the shape.
top	defines the top most pixel of the shape.
right	defines the right most pixel of the shape.
bottom	defines the bottom most pixel of the shape.

Returns

TBA (see [GFX_STATUS](#)).

Example

```

// assume RED is a macro that define GFX_COLOR types
GFX_STATUS status;
// assume BackgroundImage is a valid image already draw
// on the screen
GFX_RESOURCE_HDR *pMyBackgroundImage = &BackgroundImage;

// render a RED bar
GFX_FillStyleSet(GFX_FILL_STYLE_COLOR);
GFX_ColorSet(RED);
status = GFX_BarDraw(10, 110, 100, 200);

// render an alpha blended bar with
// the current contents of the frame buffer
GFX_FillStyleSet(GFX_FILL_STYLE_ALPHA_COLOR);
GFX_BackgroundTypeSet(GFX_BACKGROUND_DISPLAY_BUFFER);
GFX_ColorSet(RED);
status = GFX_BarDraw(10, 110, 100, 200);

// render an alpha blended bar with a background image
GFX_FillStyleSet(GFX_FILL_STYLE_ALPHA_COLOR);

// color value here has no effect since the background set
// is type GFX_BACKGROUND_IMAGE
GFX_BackgroundSet(0, 0, pMyBackgroundImage, 0);

GFX_BackgroundTypeSet(GFX_BACKGROUND_IMAGE);
GFX_ColorSet(RED);
status = GFX_BarDraw(10, 110, 100, 200);

```

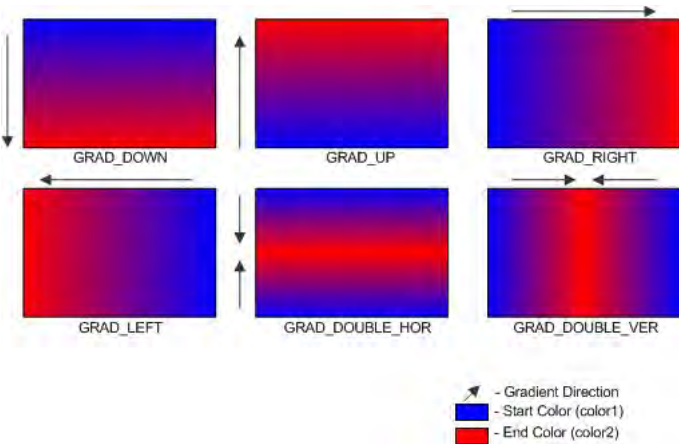
5.2.2.7.21.18 GFX_BarGradientDraw Function

C

```
GFX_STATUS GFX_BarGradientDraw(  
    uint16_t left,  
    uint16_t top,  
    uint16_t right,  
    uint16_t bottom  
);
```

Description

This renders a bar onto the screen, but instead of one color, a gradient is drawn depending on the direction (GFX_GRADIENT_TYPE), length, and colors chosen. This function is a blocking call.



Preconditions

Gradient feature must be enabled.

Parameters

Parameters	Description
left	x position of the left top corner.
top	y position of the left top corner.
right	x position of the right bottom corner.
bottom	y position of the right bottom corner.
color1	start color for the gradient
color2	end color for the gradient
length	From 0-100%. How much of a gradient is wanted
direction	Gradient Direction

Returns

Always returns a 1 since it is a blocking function.

Side Effects

none

Remarks

none

Example

```

// draw a full screen gradient background
// with color transitioning from BRIGHTRED to
// BLACK in the upward direction.

GFX_GRADIENT_STYLE gradScheme;

gradScheme.gradientType      = GFX_GRADIENT_TYPE_UP;
gradScheme.gradientStartColor = BRIGHTRED;
gradScheme.gradientEndColor   = BLACK;

BarGradient(0,                                //left position
             0,                                //top position
             GetMaxX(),                       //right position
             GetMaxY(),                       //bottom position
             gradScheme.gradientStartColor,
             gradScheme.gradientEndColor,
             50,                                // at the halfway point (50%) of the
rectangular area                               // defined by the first 4 parameters
(full screen),                               // the color becomes BLACK and BLACK
color is used until                           // the rectangle defined is filled up
                                               // see GFX_GRADIENT_TYPE

             gradScheme.gradientType);

```

5.2.2.7.21.19 GFX_BevelDraw Function**C**

```

GFX_STATUS GFX_BevelDraw(
    uint16_t x1,
    uint16_t y1,
    uint16_t x2,
    uint16_t y2,
    uint16_t rad
);

```

Description

This is function GFX_BevelDraw.

5.2.2.7.21.20 GFX_BevelDrawTypeGet Function**C**

```

GFX_BEVEL_RENDER_TYPE GFX_BevelDrawTypeGet();

```

5.2.2.7.21.21 GFX_BevelDrawTypeSet Function**C**

```

void GFX_BevelDrawTypeSet(
    GFX_BEVEL_RENDER_TYPE type
);

```

5.2.2.7.21.22 GFX_BevelGradientDraw Function**C**

```

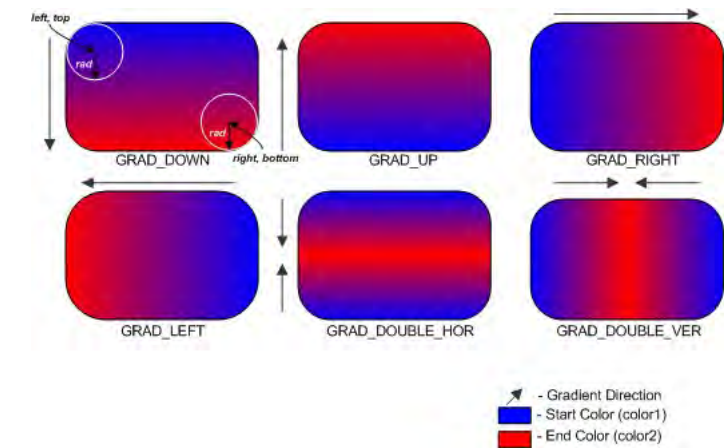
GFX_STATUS GFX_BevelGradientDraw(
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,

```

```
uint16_t rad
);
```

Description

This renders a filled bevel with gradient color on the fill. It works the same as the fillbevel function, except a gradient out of color1 and color2 is drawn depending on the direction (GFX_GRADIENT_TYPE). This function is a blocking call.



Preconditions

Gradient feature must be enabled.

Parameters

Parameters	Description
left	x coordinate position of the upper left center of the circle that draws the rounded corners.
top	y coordinate position of the upper left center of the circle that draws the rounded corners.
right	x coordinate position of the lower right center of the circle that draws the rounded corners.
bottom	y coordinate position of the lower right center of the circle that draws the rounded corners.
rad	defines the radius of the circle, that draws the rounded corners. When rad = 0, the object drawn is a rectangular gradient.
color1	start color for the gradient
color2	end color for the gradient
length	From 0-100%. How much of a gradient is wanted
direction	see GFX_GRADIENT_TYPE

Returns

Always returns a 1 since it is a blocking function.

Side Effects

none

Remarks

none

5.2.2.7.21.23 GFX_CircleDraw Function

C

```
GFX_STATUS GFX_CircleDraw(  
    uint16_t x,  
    uint16_t y,  
    uint16_t radius  
);
```

Description

This function renders a circular shape using the center (x,y) and radius. The shape is rendered using the currently set line style by [GFX_LineStyleSet\(\)](#). The color used is the color set by the last call to [GFX_ColorSet\(\)](#).

When x,y falls outside the buffer, the behavior is undefined. When color is not set before this function is called, the behavior is undefined. When any of the following x+radius, x-radius, y+radius and y-radius falls outside the buffer, the behavior is undefined.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Line style must be set by [GFX_LineStyleSet\(\)](#).

Parameters

Parameters	Description
x	defines the x-coordinate position of the center of the circle.
y	defines the y-coordinate position of the center of the circle.
radius	defines the radius of the circle.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.24 GFX_CircleFillDraw Function

C

```
GFX_STATUS GFX_CircleFillDraw(  
    uint16_t x,  
    uint16_t y,  
    uint16_t radius  
);
```

Description

This function renders a filled circle shape with the currently set fill style (see [GFX_FILL_STYLE](#)) with the given left, top, right and bottom parameters to define the shape dimension. The shape is rendered depending on the fill style. If a flat color is used, color must be set (see [GFX_ColorSet\(\)](#)) before calling this function. If gradient color is used, gradient start and end color must be set (see [GFX_GradientColorSet\(\)](#)) before calling this function. After the fill style and colors are set, multiple calls to this function can be performed.

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top or right,bottom falls outside the frame buffer.
- Fill style is not set ([GFX_FillStyleSet\(\)](#)), before this function is called.
- Colors are not set before this function is called.
- When the center defined by x,y is not on the frame buffer.

Preconditions

Fill style must be set by [GFX_FillStyleSet\(\)](#). Color must be set by [GFX_ColorSet\(\)](#).

Parameters

Parameters	Description
x	defines the x-coordinate position of the center of the circle.
y	defines the y-coordinate position of the center of the circle.
radius	defines the radius of the circle.

Returns

TBA (see [GFX_STATUS](#)).

Example

```
// assume BLUE and RED are macros that define GFX_COLOR types
GFX_STATUS status;

APP_TEST_FillStyleSet(GFX_FILL_STYLE_GRADIENT_UP);
GFX_GradientColorSet(BLUE, RED);
status = GFX_CircleFillDraw(50, 110, 150, 200, 20);
if (status == GFX_STATUS_SUCCESS)
    // Filled circle was drawn.
else
    // Filled circle is not drawn or not yet
    // finished rendering. To finish the rendering call the
    // function again with the same parameters.
```

5.2.2.7.21.25 GFX_ColorGet Function**C**

```
GFX_COLOR GFX_ColorGet();
```

Description

This function returns the color currently set to render primitive shapes and text (See [GFX_ColorSet\(\)](#) for more information).

Preconditions

None.

Returns

The currently set color.

Example

TBA

5.2.2.7.21.26 GFX_ColorSet Function**C**

```
void GFX_ColorSet(
    GFX_COLOR newColor
);
```

Description

This function sets the color to be used to render primitive shapes and text. Any primitive shape that is set to any of the following:
[GFX_LINE_STYLE](#)

- [GFX_LINE_STYLE_THIN_SOLID](#)
- [GFX_LINE_STYLE_THIN_DOTTED](#)

- GFX_LINE_STYLE_THIN_DASHED
- GFX_LINE_STYLE_THICK_SOLID
- GFX_LINE_STYLE_THICK_DOTTED
- GFX_LINE_STYLE_THICK_DASHED

[GFX_FILL_STYLE](#)

- GFX_FILL_STYLE_COLOR

will be rendered using the set color.

Rendering text using the [GFX_TextCharDraw\(\)](#), [GFX_TextStringDraw\(\)](#) and [GFX_TextStringBoxDraw\(\)](#) will also use the set color.

Preconditions

None.

Parameters

Parameters	Description
color	the rendering color set to render primitive shapes.

Returns

None.

Example

TBA

5.2.2.7.21.27 GFX_DoubleBufferAreaGet Function

C

```
GFX_RECTANGULAR_AREA * GFX_DoubleBufferAreaGet(
    uint16_t count
);
```

5.2.2.7.21.28 GFX_DoubleBufferAreaMark Function

C

```
void GFX_DoubleBufferAreaMark(
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom
);
```

5.2.2.7.21.29 GFX_DoubleBufferDisable Function

C

```
GFX_STATUS GFX_DoubleBufferDisable();
```

5.2.2.7.21.30 GFX_DoubleBufferEnable Function

C

```
GFX_STATUS GFX_DoubleBufferEnable();
```


5.2.2.7.21.31 GFX_DoubleBufferStatusGet Function**C**

```
GFX_FEATURE_STATUS GFX_DoubleBufferStatusGet();
```

5.2.2.7.21.32 GFX_DoubleBufferSyncAllStatusClear Function**C**

```
GFX_STATUS GFX_DoubleBufferSyncAllStatusClear();
```

5.2.2.7.21.33 GFX_DoubleBufferSyncAllStatusGet Function**C**

```
GFX_FEATURE_STATUS GFX_DoubleBufferSyncAllStatusGet();
```

5.2.2.7.21.34 GFX_DoubleBufferSyncAllStatusSet Function**C**

```
GFX_STATUS GFX_DoubleBufferSyncAllStatusSet();
```

5.2.2.7.21.35 GFX_DoubleBufferSyncAreaCountGet Function**C**

```
unsigned int GFX_DoubleBufferSyncAreaCountGet();
```

5.2.2.7.21.36 GFX_DoubleBufferSyncAreaCountSet Function**C**

```
void GFX_DoubleBufferSyncAreaCountSet(  
    unsigned int count  
);
```

5.2.2.7.21.37 GFX_DoubleBufferSynchronize Function**C**

```
GFX_STATUS GFX_DoubleBufferSynchronize();
```

5.2.2.7.21.38 GFX_DoubleBufferSynchronizeCancel Function**C**

```
GFX_STATUS GFX_DoubleBufferSynchronizeCancel();
```

5.2.2.7.21.39 GFX_DoubleBufferSynchronizeRequest Function**C**

```
GFX_STATUS GFX_DoubleBufferSynchronizeRequest();
```

5.2.2.7.21.40 GFX_DoubleBufferSynchronizeSet Function**C**

```
GFX_STATUS GFX_DoubleBufferSynchronizeSet();
```

5.2.2.7.21.41 GFX_DoubleBufferSynchronizeStatusGet Function**C**

```
GFX_FEATURE_STATUS GFX_DoubleBufferSynchronizeStatusGet();
```

5.2.2.7.21.42 GFX_DoubleBufferVisualPageUpdate Function

C

```
GFX_STATUS GFX_DoubleBufferVisualPageUpdate();
```

Description

None

Preconditions

None

Parameters

Parameters	Description
None	None

Returns

None

Example

None

5.2.2.7.21.43 GFX_DrawBufferGet Function

C

```
int GFX_DrawBufferGet();
```

5.2.2.7.21.44 GFX_DrawBufferSelect Function

C

```
GFX_STATUS GFX_DrawBufferSelect(  
    int index  
);
```

5.2.2.7.21.45 GFX_DrawBufferSet Function

C

```
GFX_STATUS GFX_DrawBufferSet(  
    int index,  
    uint32_t address  
);
```

5.2.2.7.21.46 GFX_DRV_FontSet Function

C

```
void GFX_DRV_FontSet(  
    GFX_RESOURCE_HDR * pFont  
);
```

Description

This is function GFX_DRV_FontSet.

5.2.2.7.21.47 GFX_DRV_TextStringWidthGet Function

C

```
uint16_t GFX_DRV_TextStringWidthGet(  
    const char * pText,  
    uint16_t x,  
    uint16_t y,  
    uint16_t width,  
    uint16_t height,  
    uint16_t * pWidth,  
    uint16_t * pHeight  
);
```

```
GFX_XCHAR* textString,
GFX_RESOURCE_HDR * pFont
);
```

Description

This is function GFX_DRV_TextStringWidthGet.

5.2.2.7.21.48 GFX_ExternalCharInfoGet Function**C**

```
void GFX_ExternalCharInfoGet(
    GFX_XCHAR ch,
    GFX_FONT_OUTCHAR * pParam
);
```

Description

This is function GFX_ExternalCharInfoGet.

5.2.2.7.21.49 GFX_ExternalResourceCallback Function**C**

```
GFX_STATUS GFX_ExternalResourceCallback(
    GFX_RESOURCE_HDR * pResource,
    uint32_t offset,
    uint16_t nCount,
    void * pBuffer
);
```

Description

This function must be implemented in the application. The library will call this function each time when the external memory data will be required. The application must copy requested uint8_ts quantity into the buffer provided. Data start address in external memory is a sum of the address in GFX_EXTDATA structure and offset.

Parameters

Parameters	Description
memory	Pointer to the external memory bitmap or font structures (FONT_EXTERNAL or BITMAP_EXTERNAL).
offset	Data offset.
nCount	Number of uint8_ts to be transferred into the buffer.
buffer	Pointer to the buffer.

Returns

Returns the number of uint8_ts were transferred.

Side Effects

none

Example

```
// If there are several memories in the system they can be selected by IDs.
// In this example, ID for memory device used is assumed to be 0.
#define X_MEMORY 0

uint16_t ExternalMemoryCallback(GFX_IMAGE_HEADER* memory, int32_t offset, uint16_t nCount,
void* buffer) {
    int i;
    long address;
```

```

    // Address of the requested data is a start address of the object referred by GFX_EXTDATA
    // structure plus offset
    address = memory->address+offset;

    if(memory->ID == X_MEMORY){
        // MemoryXReaduint8_t() is some function implemented to access external memory.
        // Implementation will be specific to the memory used. In this example
        // it reads uint8_t each time it is called.
        i = 0;
        while (i < nCount) {
            (uint8_t*)buffer = MemoryXReaduint8_t(address++);
            i++;
        }
        // return the actual number of uint8_ts retrieved
        return (i);
    }
}

```

5.2.2.7.21.50 GFX_FillStyleGet Function

C

```
GFX_FILL_STYLE GFX_FillStyleGet();
```

Description

This function returns the fill style used (see [GFX_FILL_STYLE](#)) when rendering filled shapes.

Preconditions

None.

Returns

The current fill style used (see [GFX_FILL_STYLE](#)).

Example

TBA

5.2.2.7.21.51 GFX_FillStyleSet Function

C

```
GFX_STATUS GFX_FillStyleSet(
    GFX_FILL_STYLE style
);
```

Description

This function sets the fill style to be used (see [GFX_FILL_STYLE](#)) when rendering filled shapes. All calls to the following:

- [GFX_RectangleFillDraw\(\)](#)
- [GFX_RectangleRoundFillDraw\(\)](#)
- [GFX_CircleFillDraw\(\)](#)

will use the fill style set by this function.

Preconditions

None.

Parameters

Parameters	Description
style	The specified fill style to be used.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.52 GFX_FlashCharInfoGet Function

C

```
void GFX_FlashCharInfoGet(  
    GFX_XCHAR ch,  
    GFX_FONT_OUTCHAR * pParam  
);
```

Description

This is function GFX_FlashCharInfoGet.

5.2.2.7.21.53 GFX_FontAntiAliasGet Function

C

```
GFX_FONT_ANTIALIAS_TYPE GFX_FontAntiAliasGet();
```

Description

This function returns the font anti-aliasing mode used when rendering anti-aliased strings and characters. Default setting at initialization is GFX_FONT_ANTIALIAS_OPAQUE.

Preconditions

None.

Returns

The font anti-aliasing mode used.

Example

TBA

5.2.2.7.21.54 GFX_FontAntiAliasSet Function

C

```
GFX_STATUS GFX_FontAntiAliasSet(  
    GFX_FONT_ANTIALIAS_TYPE type  
);
```

Description

This function sets the font anti-aliasing mode used when rendering anti-aliased strings and characters.

Preconditions

None.

Parameters

Parameters	Description
type	anti-aliasing mode selected. See GFX_FONT_ANTIALIAS_TYPE for details.

Returns

The status of the set anti-aliasing mode action.

Example

TBA

5.2.2.7.21.55 GFX_FontGet Function

C

```
GFX_RESOURCE_HDR* GFX_FontGet();
```

Description

This function returns the current font used to render strings and characters.

Preconditions

[GFX_FontSet\(\)](#) must be called prior to any call to this function. If this function is called first, the returned value is undefined.

Returns

The pointer to the current font used in rendering strings and characters.

Example

TBA

5.2.2.7.21.56 GFX_FontSet Function

C

```
GFX_STATUS GFX_FontSet(  
    GFX_RESOURCE_HDR * pFont  
);
```

Description

This function sets the current font used to render strings and characters.

Preconditions

None.

Parameters

Parameters	Description
pFont	pointer to the font to be used.

Returns

The status of the set font action.

Example

TBA

5.2.2.7.21.57 GFX_FrameBufferGet Function

C

```
int GFX_FrameBufferGet();
```

5.2.2.7.21.58 GFX_FrameBufferSelect Function

C

```
GFX_STATUS GFX_FrameBufferSelect(  
    int index  
);
```

5.2.2.7.21.59 GFX_GradientColorSet Function

```
C
GFX_STATUS GFX_GradientColorSet(
    GFX_COLOR startColor,
    GFX_COLOR endColor
);
```

Description

This function sets the gradient fill start and end colors (see [GFX_FILL_STYLE](#)) when rendering gradient filled shapes. All subsequent calls to the following using gradient fills:

- [GFX_RectangleFillDraw\(\)](#)
- [GFX_RectangleRoundFillDraw\(\)](#)
- [GFX_CircleFillDraw\(\)](#)

will use the start and end colors set by this function.

Preconditions

None.

Parameters

Parameters	Description
startColor	Gradient start color to be used.
endColor	Gradient end color to be used.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.60 GFX_GradientEndColorGet Function

```
C
GFX_COLOR GFX_GradientEndColorGet();
```

Description

This function returns the end color used when rendering gradient filled shapes.

Preconditions

None.

Returns

The current gradient end color used.

Example

TBA

5.2.2.7.21.61 GFX_GradientStartColorGet Function

```
C
GFX_COLOR GFX_GradientStartColorGet();
```

Description

This function returns the start color used when rendering gradient filled shapes.

Preconditions

None.

Returns

The current gradient start color used.

Example

TBA

5.2.2.7.21.62 GFX_ImageDraw Function

C

```
GFX_STATUS GFX_ImageDraw(
    uint16_t left,
    uint16_t top,
    GFX_RESOURCE_HDR * pImage
);
```

Description

This function renders an image to the frame buffer with the left-top corner of the image located at given left, top parameters.

The rendering of this shape becomes undefined when any one of the following is true:

- left, top pixel position falls outside the frame buffer.
- pointer is not properly initialized to a [GFX_RESOURCE_HDR](#) object.

Preconditions

None.

Parameters

Parameters	Description
left	Horizontal starting position of the image.
top	Vertical starting position of the image.
pImage	Pointer to the image to be rendered.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.63 GFX_ImageHeaderGet Function

C

```
inline void GFX_ImageHeaderGet(
    GFX_RESOURCE_HDR * pImage,
    GFX_MCHP_BITMAP_HEADER * pBitmapHdr
);
```

Description

This function fills the given bitmap header with the image's header information. This function results in an undefined behavior if the pointer to the image is invalid.

Preconditions

None.

Parameters

Parameters	Description
pImage	Pointer to the image.
pBitmap	Pointer to the destination of the header information.

Returns

None.

Example

TBA

5.2.2.7.21.64 GFX_ImageHeightGet Function**C**

```
inline int16_t GFX_ImageHeightGet(  
    GFX_RESOURCE_HDR * pImage  
)  
;
```

Description

This function returns the height of the given image in pixels. This function results in an undefined behavior if the pointer to the image is invalid.

This function return value is undefined if the given pointer does not point to a valid image resource.

Preconditions

None.

Parameters

Parameters	Description
pImage	Pointer to the image.

Returns

The image height in pixels.

Example

TBA

5.2.2.7.21.65 GFX_ImageOffsetAddressGet Function**C**

```
uint32_t GFX_ImageOffsetAddressGet(  
    uint32_t address,  
    uint16_t xOffset,  
    uint16_t yOffset,  
    uint16_t width  
)  
;
```

Description

This is function GFX_ImageOffsetAddressGet.

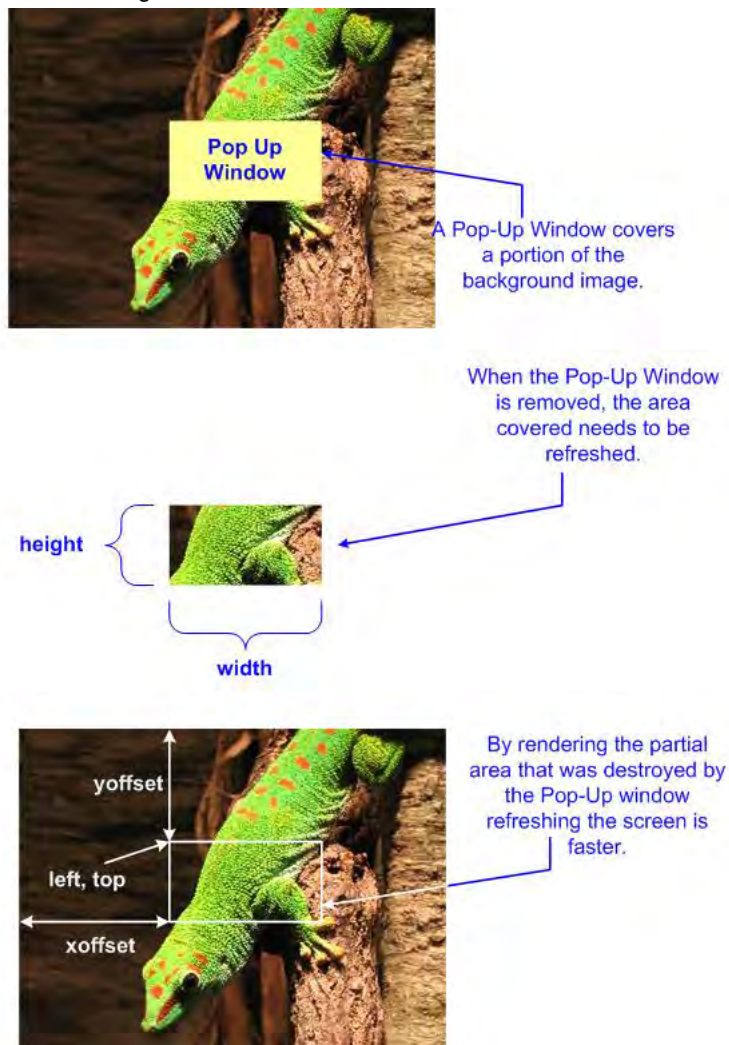
5.2.2.7.21.66 GFX_ImagePartialDraw Function

C

```
GFX_STATUS GFX_ImagePartialDraw(
    uint16_t destination_x,
    uint16_t destination_y,
    uint16_t source_x_offset,
    uint16_t source_y_offset,
    uint16_t source_width,
    uint16_t source_height,
    GFX_RESOURCE_HDR * pImage
);
```

Description

This function renders an image or a portion of an image to the frame buffer with the top, left corner of the image located at `destination_x`, `destination_y`. To render a full image, `source_x_offset`, `source_y_offset`, width and height are set to all 0. Using the actual image's width and height will also work if these are known. If they are not known, avoid the extra extra step to get the image's width and height by assigning 0 to the parameters. Another way to render a full image is to use the [GFX_ImageDraw\(\)](#). The image to be rendered is defined by the pointer `pImage`.



The rendering of this shape becomes undefined when any one of the following is true:

- destination_x, destination_y pixel position falls outside the frame buffer.
- source_x_offset, source_y_offset results in a starting position beyond the image's dimension.
- source_width and/or source_height is larger than the actual image's width and height.
- pointer is not properly initialized to a [GFX_RESOURCE_HDR](#) object.

Preconditions

None.

Parameters

Parameters	Description
destination_x	Horizontal starting position of the image.
destination_y	Vertical starting position of the image.
source_x_offset	Horizontal offset in pixels of the starting position of the partial area of the image to be rendered. Set to 0 along with source_y_offset when rendering the full image.
source_y_offset	Vertical offset in pixels of the starting position of the partial area of the image to be rendered. Set to 0 along with source_x_offset when rendering the full image.
source_width	The width of the partial area of the image to be rendered. Set to 0 along with source_height when rendering the full image.
source_height	The height of the partial area of the image to be rendered. Set to 0 along with source_width when rendering the full image.
pImage	Pointer to the image to be rendered.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.67 GFX_ImageWidthGet Function**C**

```
inline int16_t GFX_ImageWidthGet(
    GFX_RESOURCE_HDR * pImage
);
```

Description

This function returns the width of the given image in pixels. This function results in an undefined behavior if the pointer to the image is invalid.

This function return value is undefined if the given pointer does not point to a valid image resource.

Preconditions

None.

Parameters

Parameters	Description
pImage	Pointer to the image.

Returns

The image width in pixels.

Example

TBA

5.2.2.7.21.68 GFX_Initialize Function

C

```
void GFX_Initialize();
```

Description

This function initialize the Graphics Library primitive layer and Graphics Object Layer if it is enabled. The following default settings are set when this function is called.

1. font - Set to NULL. [GFX_FontSet\(\)](#) must be called prior to any text rendering.
2. line type - Set to GFX_LINE_TYPE_THIN_SOLID (see [GFX_LINE_STYLE](#)).
3. fill type - Set to GFX_FILL_TYPE_COLOR (see [GFX_FILL_STYLE](#)).
4. text anti-alias type - Set to GFX_FONT_ANTIALIAS_OPAQUE (see [GFX_FONT_ANTIALIAS_TYPE](#)). This only affects fonts with anti-aliasing enabled.

This function does not clear the screen and does not assign any color to the currently set color. Application should set the color and clear the screen.

Preconditions

None.

Returns

None.

Example

TBA

5.2.2.7.21.69 GFX_Layer Function

C

```
GFX_STATUS GFX_Layer(  
    uint8_t layer,  
    uint8_t action,  
    uint16_t param1,  
    uint16_t param2  
);
```

Description

None

Preconditions

None

Parameters

Parameters	Description
None	None

Returns

None

Example

None

5.2.2.7.21.70 GFX_LineDraw Function

C

```
GFX_STATUS GFX_LineDraw(  
    uint16_t x1,  
    uint16_t y1,  
    uint16_t x2,  
    uint16_t y2  
);
```

Description

This function renders a line from x1,y1 to x2,y2 using the currently set line style set by [GFX_LineStyleSet\(\)](#). The color used is the color set by the last call to [GFX_ColorSet\(\)](#).

If x1,y1 and/or x2,y2 is not on the frame buffer, then the behavior is undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Line style must be set by [GFX_LineStyleSet\(\)](#).

Parameters

Parameters	Description
x1	x coordinate of the line start point.
y1	y coordinate of the line start point.
x2	x coordinate of the line end point.
y2	y coordinate of the line end point.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.71 GFX_LinePositionRelativeSet Function

C

```
GFX_STATUS GFX_LinePositionRelativeSet(  
    int16_t x,  
    int16_t y  
);
```

Description

This function sets the line cursor to a new (x,y) position relative to the current cursor position. The new position is calculated by (x+dX, y + dY).

Line cursor is used as a starting point of the line rendered by the [GFX_LineToDraw\(\)](#) and [GFX_LineToRelativeDraw\(\)](#) functions. Note that the parameters dX and dY are signed integers. This allows the new line cursor position to be placed to any direction from the current line cursor position.

If (x+dX) and/or (y+dY) results in a position that is not on the frame buffer, then the behavior of [GFX_LineToDraw\(\)](#) and [GFX_LineToRelativeDraw\(\)](#) functions are undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

None.

Parameters

Parameters	Description
dX	the offset for the x position that will define the new x-coordinate position of the line cursor.
dY	the offset for the y position that will define the new y-coordinate position of the line cursor.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.72 GFX_LinePositionSet Function**C**

```
GFX_STATUS GFX_LinePositionSet(  
    uint16_t x,  
    uint16_t y  
);
```

Description

This function sets the line cursor to a new (x,y) position. Line cursor is used as a starting point of the line rendered by the [GFX_LineToDraw\(\)](#) and [GFX_LineToRelativeDraw\(\)](#) functions.

If x and/or y does not lie on the frame buffer, then the behavior of [GFX_LineToDraw\(\)](#) and [GFX_LineToRelativeDraw\(\)](#) functions are undefined.

Preconditions

None.

Parameters

Parameters	Description
x	new x coordinate position of the line cursor.
y	new y coordinate position of the line cursor.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.73 GFX_LinePositionXGet Function**C**

```
int16_t GFX_LinePositionXGet();
```

Description

This function returns the current x position of the line cursor. Line cursor is used as a starting point of the line rendered by the [GFX_LineToDraw\(\)](#) and [GFX_LineToRelativeDraw\(\)](#) functions.

Preconditions

None.

Returns

The current line cursor x position.

Example

TBA

5.2.2.7.21.74 GFX_LinePositionYGet Function**C**

```
int16_t GFX_LinePositionYGet();
```

Description

This function returns the current y position of the line cursor. Line cursor is used as a starting point of the line rendered by the [GFX_LineToDraw\(\)](#) and [GFX_LineToRelativeDraw\(\)](#) functions.

Preconditions

None.

Returns

The current line cursor y position.

Example

TBA

5.2.2.7.21.75 GFX_LineStyleGet Function**C**

```
GFX_LINE_STYLE GFX_LineStyleGet();
```

Description

This function returns the line style used (see [GFX_LINE_STYLE](#)) when rendering lines.

Preconditions

None.

Returns

The current line style used (see [GFX_LINE_STYLE](#)).

Example

TBA

5.2.2.7.21.76 GFX_LineStyleSet Function**C**

```
GFX_STATUS GFX_LineStyleSet(  
    GFX_LINE_STYLE style  
);
```

Description

This function sets the line style to be used (see [GFX_LINE_STYLE](#)) when rendering lines. All calls to the following functions

- [GFX_LineDraw\(\)](#)
- [GFX_LineToDraw\(\)](#)
- [GFX_LineToRelativeDraw\(\)](#)

- [GFX_CircleDraw\(\)](#)
- [GFX_RectangleDraw\(\)](#)
- [GFX_PolygonDraw\(\)](#)

will use the line style set by this function. In addition, all unfilled shapes that does specify the line style to be used will use the line style specified by this function.

Preconditions

None.

Parameters

Parameters	Description
style	The specified line style to be used.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.77 GFX_LineToDraw Function**C**

```
GFX_STATUS GFX_LineToDraw(  
    int16_t x2,  
    int16_t y2  
);
```

Description

This function renders a line from current line cursor position (x,y) to (x2,y2) using the currently set line style set by [GFX_LineStyleSet\(\)](#). The color used is the color set by the last call to [GFX_ColorSet\(\)](#).

If x2 and/or y2 does not lie on the frame buffer, then the behavior is undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Line style must be set by [GFX_LineStyleSet\(\)](#).

Parameters

Parameters	Description
x2	x coordinate of the line end point.
y2	y coordinate of the line end point.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.78 GFX_LineToRelativeDraw Function**C**

```
GFX_STATUS GFX_LineToRelativeDraw(  
    int16_t dX,  
    int16_t dY
```



```
);
```

Description

This function renders a line from current line cursor position (x,y) to (x+dX,y+dY) using the currently set line style set by [GFX_LineStyleSet\(\)](#). The color used is the color set by the last call to [GFX_ColorSet\(\)](#). Note that the parameters dX and dY are signed integers. This allows the line to be drawn from the line cursor to any direction.

If (x+dX) and/or (y+dY) results in a position that is not on the frame buffer, then the behavior is undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Line style must be set by [GFX_LineStyleSet\(\)](#).

Parameters

Parameters	Description
dX	the offset for the x starting position that will define the x-coordinate of the end of the line.
dY	the offset for the y starting position that will define the y-coordinate of the end of the line.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.79 GFX_PageSet Function

C

```
GFX_STATUS GFX_PageSet(
    uint8_t  pageType,
    uint8_t  page
);
```

Example

Remarks:

5.2.2.7.21.80 GFX_PolygonDraw Function

C

```
GFX_STATUS GFX_PolygonDraw(
    uint16_t numPoints,
    uint16_t * polyPoints
);
```

Description

This function renders a polygon using the currently set line style (see [GFX_LineStyleSet\(\)](#)) and color (see [GFX_ColorSet\(\)](#)). The shape of the polygon is determined by the polygon points (an ordered array of x,y pairs) where the pair count is equal to the parameter sides.

If any of the x,y pairs do not lie on the frame buffer, then the behavior is undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Line style must be set by [GFX_LineStyleSet\(\)](#).

Parameters

Parameters	Description
sides	the number of sides of the polygon.
pPoints	Pointer to the array of polygon points. The array defines the x,y points of the polygon. The sequence should be x0, y0, x1, y1, x2, y2, ... xn, yn where n is the # of polygon sides.

Returns

TBA (see [GFX_STATUS](#)).

Example

```
uint16_t OpenShapeXYPoints[6] = {10, 10, 20, 10, 20, 20};
uint16_t ClosedShapeXYPoints[8] = {10, 10, 20, 10, 20, 20, 10, 10};

GFX_ColorSet(WHITE);                // set color
SetLineType(GFX_LINE_STYLE_THIN_DOTTED); // set line style
GFX_PolygonDraw(3, OpenShapeXYPoints); // draw an open shape
GFX_PolygonDraw(4, ClosedShapeXYPoints); // draw a closed shape
```

5.2.2.7.21.81 GFX_Primitive_Initialize Function**C**

```
inline void GFX_Primitive_Initialize();
```

Description

This is function `GFX_Primitive_Initialize`.

5.2.2.7.21.82 GFX_RectangleDraw Function**C**

```
GFX_STATUS GFX_RectangleDraw(
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom
);
```

Description

This function renders a rectangular shape using the given left, top, right and bottom parameters to define the shape dimension. The shape is rendered using the currently set line style by [GFX_LineStyleSet\(\)](#). The color used is the color set by the last call to [GFX_ColorSet\(\)](#).

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top, right,bottom falls outside the frame buffer.
- Color is not set, before this function is called.
- right < left
- bottom < top

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Line style must be set by [GFX_LineStyleSet\(\)](#).

Parameters

Parameters	Description
left	defines the left most pixel of the shape.

top	defines the top most pixel of the shape.
right	defines the right most pixel of the shape.
bottom	defines the bottom most pixel of the shape.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.83 GFX_RectangleFillDraw Function**C**

```
GFX_STATUS GFX_RectangleFillDraw(
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom
);
```

Description

This function renders a filled rectangular shape with the currently set fill style (see [GFX_FILL_STYLE](#)) with the given left, top, right, and bottom parameters to define the shape dimension. The shape is rendered depending on the fill style. If a flat color is used, color must be set (see [GFX_ColorSet\(\)](#)) before calling this function. If gradient color is used, gradient start and end color must be set (see [GFX_GradientColorSet\(\)](#)) before calling this function. After the fill style and colors are set, multiple calls to this function can be performed.

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top or right,bottom falls outside the frame buffer.
- Fill style is not set ([GFX_FillStyleSet\(\)](#)), before this function is called.
- Colors are not set before this function is called.
- When right < left
- When bottom < top
- When pixel locations defined by left, top and/or right, bottom are not on the frame buffer.

Preconditions

Fill style must be set by [GFX_FillStyleSet\(\)](#). Color must be set by [GFX_ColorSet\(\)](#).

Parameters

Parameters	Description
left	defines the left most pixel of the shape.
top	defines the top most pixel of the shape.
right	defines the right most pixel of the shape.
bottom	defines the bottom most pixel of the shape.

Returns

TBA (see [GFX_STATUS](#)).

Example

```
// assume BLUE and RED are macros that define GFX_COLOR types
GFX_STATUS status;

// render a rectangle with gradient colors from BLUE to RED
```

```

GFX_FillStyleSet(GFX_FILL_STYLE_GRADIENT_LEFT);
GFX_GradientColorSet(BLUE, RED);
status = GFX_RectangleFillDraw(10, 110, 100, 200);

// render an alpha blended rounded rectangle with
// the current contents of the frame buffer
GFX_FillStyleSet(GFX_FILL_STYLE_ALPHA_COLOR);
GFX_AlphaBlendingValueSet(50);
GFX_BackgroundTypeSet(GFX_BACKGROUND_DISPLAY_BUFFER);
GFX_ColorSet(RED);
status = GFX_RectangleRoundFillDraw(10, 110, 100, 200);

// render an alpha blended rectangle with an image
GFX_FillStyleSet(GFX_FILL_STYLE_ALPHA_COLOR);
GFX_AlphaBlendingValueSet(50);
GFX_BackgroundTypeSet(GFX_BACKGROUND_IMAGE);
GFX_ColorSet(RED);
status = GFX_RectangleFillDraw(10, 110, 100, 200);

// render a plain RED rounded rectangle
GFX_FillStyleSet(GFX_FILL_STYLE_COLOR);
GFX_ColorSet(RED);
status = GFX_RectangleFillDraw(10, 110, 100, 200);

```

5.2.2.7.21.84 GFX_RectangleRoundDraw Function

C

```

GFX_STATUS GFX_RectangleRoundDraw(
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t radius
);

```

Description

This function renders a rectangular shape with rounded corner using the given left, top, right, bottom and radius parameters to define the shape dimension. radius defines the rounded corner shape. The shape is rendered using the currently set line style by [GFX_LineStyleSet\(\)](#). The color used is the color set by the last call to [GFX_ColorSet\(\)](#).

Left most pixel location is defined by left - radius. Top most pixel location is defined by top - radius. Right most pixel location is defined by right + radius. Bottom most pixel location is defined by bottom + radius. When radius = 0, there are no rounded corners. In this case (left,top) will define the left, top corner and (right,bottom) will define the right, bottom corner of the shape.

When left = right and top = bottom, with radius > 0, a circular object is drawn. When left < right and top < bottom and radius = 0, a rectangular object is drawn.

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left-rad , top-rad, right+rad, bottom+rad falls outside the frame buffer.
- Color is not set, before this function is called.
- right < left
- bottom < top

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Line style must be set by [GFX_LineStyleSet\(\)](#).

Parameters

Parameters	Description
left	Along with rad (left - rad), defines the left most pixel of the shape.

top	Along with rad (top - rad), defines the top most pixel of the shape.
right	Along with rad (right + rad), defines the right most pixel of the shape.
bottom	Along with rad (bottom + rad), defines the bottom most pixel of the shape.
radius	defines the rounded corners. When this is set to zero, there will be no rounded corners.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.85 GFX_RectangleRoundFillDraw Function**C**

```
GFX_STATUS GFX_RectangleRoundFillDraw(
    uint16_t left,
    uint16_t top,
    uint16_t right,
    uint16_t bottom,
    uint16_t radius
);
```

Description

This function renders a filled rounded rectangular shape with the currently set fill style (see [GFX_FILL_STYLE](#)) with the given left, top, right, bottom and radius parameters to define the shape dimension. The shape is rendered depending on the fill style. If a flat color is used, color must be set (see [GFX_ColorSet\(\)](#)) before calling this function. If gradient color is used, gradient start and end color must be set (see [GFX_GradientColorSet\(\)](#)) before calling this function. After the fill style and colors are set, multiple calls to this function can be performed.

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top or right,bottom falls outside the frame buffer.
- Fill style is not set ([GFX_FillStyleSet\(\)](#)), before this function is called.
- Colors are not set before this function is called.
- When right < left
- When bottom < top
- When pixel locations defined by left, top and/or right, bottom are not on the frame buffer.

Preconditions

Fill style must be set by [GFX_FillStyleSet\(\)](#). Color must be set by [GFX_ColorSet\(\)](#).

Parameters

Parameters	Description
left	defines the left most pixel of the shape.
top	defines the top most pixel of the shape.
right	defines the right most pixel of the shape.
bottom	defines the bottom most pixel of the shape.
radius	defines the radius of the rounded corner. A zero value will result in a rectangular shape drawn.

Returns

TBA (see [GFX_STATUS](#)).

Example

```
// assume BLUE and RED are macros that define GFX_COLOR types
GFX_STATUS status;

APP_TEST_FillStyleSet(GFX_FILL_STYLE_GRADIENT_DOUBLE_VER);
GFX_GradientColorSet(BLUE, RED);
status = GFX_RectangleFillDraw(50, 110, 150, 200, 20);
if (status == GFX_STATUS_SUCCESS)
    // Filled rounded rectangle shape was drawn.
else
    // Filled rounded rectangle shape is not drawn or not yet
    // finished rendering. To finish the rendering call the
    // function again with the same parameters.
```

5.2.2.7.21.86 GFX_RenderingBufferGet Function**C**

```
GFX_COLOR * GFX_RenderingBufferGet();
```

Description

Returns the location of the internal rendering buffer used in rendering images and filled objects with alpha blending.

Preconditions

None.

Returns

The address of the rendering buffer.

Example

None.

5.2.2.7.21.87 GFX_RenderToDisplayBufferDisable Function**C**

```
void GFX_RenderToDisplayBufferDisable();
```

Description

This is function GFX_RenderToDisplayBufferDisable.

5.2.2.7.21.88 GFX_RenderToDisplayBufferDisableFlagGet Function**C**

```
uint16_t GFX_RenderToDisplayBufferDisableFlagGet();
```

Description

This is function GFX_RenderToDisplayBufferDisableFlagGet.

5.2.2.7.21.89 GFX_RenderToDisplayBufferEnable Function**C**

```
void GFX_RenderToDisplayBufferEnable();
```

Description

This is function GFX_RenderToDisplayBufferEnable.

5.2.2.7.21.90 GFX_ScreenClear Function

C

```
GFX_STATUS GFX_ScreenClear();
```

Description

This function clears the screen with the current color and sets the line cursor position to (0, 0).

If color is not set, before this function is called, the output is undefined.

If the function returns GFX_STATUS_FAILURE, clearing is not yet finished. Application must call the function again to continue the clearing.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#).

None.

Returns

The status of the screen clearing.

- GFX_STATUS_SUCCESS - screen was cleared.
- GFX_STATUS_FAILURE - screen is not yet cleared

Example

TBA

5.2.2.7.21.91 GFX_SetTransitionParameters Function

C

```
uint8_t GFX_SetTransitionParameters(
    short left,
    short top,
    GFX_TRANSITION_TYPE type,
    uint8_t sourcePage,
    uint8_t destinationPage,
    uint16_t delay_ms,
    uint16_t param1,
    uint16_t param2
);
```

Description

This sets up the transition effect using the [GFX_TRANSITION_TYPE](#) and the given parameters. The actual transition execution will occur when `GFXExecutePendingTransition()` is called. When `DOUBLE_BUFFERING` is enabled, `GFXExecutePendingTransition()` is executed after the current screen is fully rendered.

Parameters

Parameters	Description
left	left x coordinate
top	top y coordinate
right	right x coordinate
bottom	bottom y coordinate
type	Transition type
delay_ms	Delay in milli seconds between redraws in the screen while executing the transition
param1	Transition-type specific parameter

param2	Transition-type specific parameter

Returns

Returns success of the setup

- 0 : Parameters successfully saved for the new transition
- -1 : Parameters not saved, there is a pending transition

5.2.2.7.21.92 GFX_SolidLineDraw Function**C**

```
GFX_STATUS GFX_SolidLineDraw(  
    uint16_t x1,  
    uint16_t y1,  
    uint16_t x2,  
    uint16_t y2  
);
```

Description

This is function GFX_SolidLineDraw.

5.2.2.7.21.93 GFX_StyledLineDraw Function**C**

```
GFX_STATUS GFX_StyledLineDraw(  
    uint16_t x1,  
    uint16_t y1,  
    uint16_t x2,  
    uint16_t y2  
);
```

Description

This is function GFX_StyledLineDraw.

5.2.2.7.21.94 GFX_TextAreaBottomGet Function**C**

```
uint16_t GFX_TextAreaBottomGet();
```

Description

This is function GFX_TextAreaBottomGet.

5.2.2.7.21.95 GFX_TextAreaBottomSet Function**C**

```
void GFX_TextAreaBottomSet(  
    uint16_t bottom  
);
```

Description

This is function GFX_TextAreaBottomSet.

5.2.2.7.21.96 GFX_TextAreaLeftGet Function

C

```
uint16_t GFX_TextAreaLeftGet();
```

Description

This is function GFX_TextAreaLeftGet.

5.2.2.7.21.97 GFX_TextAreaLeftSet Function

C

```
void GFX_TextAreaLeftSet(  
    uint16_t left  
);
```

Description

This is function GFX_TextAreaLeftSet.

5.2.2.7.21.98 GFX_TextAreaRightGet Function

C

```
uint16_t GFX_TextAreaRightGet();
```

Description

This is function GFX_TextAreaRightGet.

5.2.2.7.21.99 GFX_TextAreaRightSet Function

C

```
void GFX_TextAreaRightSet(  
    uint16_t right  
);
```

Description

This is function GFX_TextAreaRightSet.

5.2.2.7.21.100 GFX_TextAreaTopGet Function

C

```
uint16_t GFX_TextAreaTopGet();
```

Description

This is function GFX_TextAreaTopGet.

5.2.2.7.21.101 GFX_TextAreaTopSet Function

C

```
void GFX_TextAreaTopSet(  
    uint16_t top  
);
```

Description

This is function GFX_TextAreaTopSet.

5.2.2.7.21.102 GFX_TextCharDraw Function

C

```
GFX_STATUS GFX_TextCharDraw(
    GFX_XCHAR ch
);
```

Description

This function renders the given character using the currently set font, and color to the location defined by the text cursor position. The color is set by [GFX_ColorSet\(\)](#) while the font is set by [GFX_FontSet\(\)](#). The text cursor position is set by [GFX_TextCursorPositionSet\(\)](#)

The rendering of the character becomes undefined when any one of the following is true:

- Text cursor position is set to locations outside the frame buffer.
- Color is not set, before this function is called.
- Font is not set, before this function is called.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Font must be set by [GFX_FontSet\(\)](#). Text cursor position must be set by [GFX_TextCursorPositionSet\(\)](#).

Parameters

Parameters	Description
ch	character ID that of the character that will be rendered.

Returns

The status of the character rendering.

- GFX_STATUS_SUCCESS - the character was rendered
- GFX_STATUS_FAILURE - the character was not rendered, function must be called again to render the character.
- GFX_STATUS_ERROR - the character ID passed is not a valid or points to the character glyph that does not exist on the font table.

Example

```
// assume textString is a string of characters
static uint16_t counter = 0;
GFX_XCHAR ch;
GFX_STATUS status;

// render characters until null character
while((GFX_XCHAR)(ch = *(textString + counter)) != 0)
{
    status = GFX_TextCharDraw(ch);
    if (status != GFX_STATUS_SUCCESS)
        return (GFX_STATUS_FAILURE);
    counter++;
}
```

5.2.2.7.21.103 GFX_TextCursorPositionSet Function

C

```
GFX_STATUS GFX_TextCursorPositionSet(
    int16_t x,
    int16_t y
);
```

Description

This function sets the text cursor to a new (x,y) position. Text cursor is used as a starting point of the character rendered by the [GFX_TextCharDraw\(\)](#) function.

If x and/or y does not lie on the frame buffer, then the behavior of [GFX_TextCharDraw\(\)](#) function is undefined.

Preconditions

None.

Parameters

Parameters	Description
x	new x coordinate position of the text cursor.
y	new y coordinate position of the text cursor.

Returns

TBA (see [GFX_STATUS](#)).

Example

TBA

5.2.2.7.21.104 GFX_TextCursorPositionXGet Function**C**

```
int16_t GFX_TextCursorPositionXGet();
```

Description

This function returns the current x position of the text cursor. Text cursor is used as a starting point of the line rendered by the [GFX_TextCharDraw\(\)](#) function.

Preconditions

None.

Returns

The current text cursor x position.

Example

TBA

5.2.2.7.21.105 GFX_TextCursorPositionYGet Function**C**

```
int16_t GFX_TextCursorPositionYGet();
```

Description

This function returns the current y position of the text cursor. Text cursor is used as a starting point of the line rendered by the [GFX_TextCharDraw\(\)](#) function.

Preconditions

None.

Returns

The current text cursor y position.

Example

TBA

5.2.2.7.21.106 GFX_TextStringBoxDraw Function

C

```
GFX_STATUS GFX_TextStringBoxDraw(  
    uint16_t x,  
    uint16_t y,  
    uint16_t width,  
    uint16_t height,  
    GFX_XCHAR * pString,  
    GFX_ALIGNMENT align  
);
```

Description

This function renders the given string using the currently color and font into the rectangular area defined by the given parameters x,y, width and height. The x,y parameters defines the left, top pixel position of the rectangular area. The width and height defines the size of the rectangular area. The rectangular area will define the area where the text will be rendered. Meaning all pixels EXCLUSIVE of the defined rectangle will be the rendering area for the text. If the given string exceeds the rectangular area, any pixels falling outside and along the defined rectangle (INCLUSIVE of the rectangle) will not be drawn.

The color is set by [GFX_ColorSet\(\)](#) while the font is set by [GFX_FontSet\(\)](#).

Multiple lines of strings can be rendered. To define string composed of multiple lines, end each line with a new line character (character ID 0x0A). Each line will be rendered according to the text alignment (See GFX_TEXT_ALIGNMENT).

The rendering of the character becomes undefined when any one of the following is true:

- x, y, width and height defines an area partially or fully outside outside the frame buffer.
- Color is not set, before this function is called.
- Font is not set, before this function is called.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Font must be set by [GFX_FontSet\(\)](#).

Parameters

Parameters	Description
x	Horizontal starting position of the rectangular area.
y	Vertical position position of the rectangular area.
width	Defines the width of the rectangular area.
height	Defines the height of the rectangular area.
pString	Pointer to the location of the string that will be rendered. String can have multiple lines where each line is terminated by a new line character.

Returns

The status of the string rendering.

- GFX_STATUS_SUCCESS - the string was rendered
- GFX_STATUS_FAILURE - the string was not rendered, or is not yet finished. The function must be called again to render the remaining characters or lines.

Example

TODO :need to add example code

5.2.2.7.21.107 GFX_TextStringDraw Function

C

```
GFX_STATUS GFX_TextStringDraw(
    uint16_t x,
    uint16_t y,
    GFX_XCHAR * textString
);
```

Description

This function renders the given string of character using the currently set font, and color to the location defined by the given x,y position. The color is set by [GFX_ColorSet\(\)](#) while the font is set by [GFX_FontSet\(\)](#).

The rendering of the character becomes undefined when any one of the following is true:

- x,y position is set to locations outside the frame buffer.
- Color is not set, before this function is called.
- Font is not set, before this function is called.

Preconditions

Color must be set by [GFX_ColorSet\(\)](#). Font must be set by [GFX_FontSet\(\)](#).

Parameters

Parameters	Description
x	Horizontal starting position of the string.
y	Vertical position position of the string.
pString	Pointer to the location of the string that will be rendered.

Returns

The status of the string rendering.

- GFX_STATUS_SUCCESS - the string was rendered
- GFX_STATUS_FAILURE - the string was not rendered, or is not yet finished. The function must be called again to render the remaining characters.

Example

```
// assume textString is a string of characters
static uint16_t counter = 0;
GFX_XCHAR ch;
GFX_STATUS status;

// render characters until null character
while((GFX_XCHAR)(ch = *(textString + counter)) != 0)
{
    status = GFX_TextCharDraw(ch);
    if (status != GFX_STATUS_SUCCESS)
        return (GFX_STATUS_FAILURE);
    counter++;
}
```

5.2.2.7.21.108 GFX_TextStringHeightGet Function

C

```
inline uint16_t GFX_TextStringHeightGet(
    GFX_RESOURCE_HDR * pFont
);
```

Description

This function returns the height of the given font in pixels. The given font must be present in the system.

This function return value is undefined if the given pointer does not point to a valid font.

Preconditions

None.

Parameters

Parameters	Description
pFont	Pointer to the specified font.

Returns

The height of the specified font in pixels.

Example

```
// assume pMyFont is a pointer initialized to a valid font
uint16_t height, x, y;

// center the text on the screen

height = GFX_TextStringHeightGet(pMyFont);
width  = GFX_TextStringWidthGet ( (GFX_XCHAR *)"Hello World",
                                  pMyFont
                                );
y = (GFX_MaxYGet() - height) >> 1;
x = (GFX_MaxXGet() - width) >> 1;
GFX_TextStringDraw(x, y, (GFX_XCHAR *)"Hello World");

GFX_TextStringDraw(x, y, (GFX_XCHAR *)"Hello World");
```

5.2.2.7.21.109 GFX_TextStringWidthGet Function**C**

```
uint16_t GFX_TextStringWidthGet(
    GFX_XCHAR * textString,
    GFX_RESOURCE_HDR * pFont
);
```

Description

This function returns the width of the given string using the given font in pixels. The given font must be present in the system.

This function return value is undefined if the given pointer does not point to a valid font or one or more characters in the given string does not exist on the given font.

Preconditions

None.

Parameters

Parameters	Description
pString	Pointer to the specified string.
pFont	Pointer to the specified font.

Returns

The width of the specified string using the specified font in pixels.

Example

```
// assume pMyFont is a pointer initialized to a valid font
uint16_t height, x, y;

// center the text on the screen

height = GFX_TextStringHeightGet(pMyFont);
width  = GFX_TextStringWidthGet (    (GFX_XCHAR *)"Hello World",
                                     pMyFont
                                   );
y = (GFX_MaxYGet() - height) >> 1;
x = (GFX_MaxXGet() - width) >> 1;
GFX_TextStringDraw(x, y, (GFX_XCHAR *)"Hello World");
```

5.2.2.7.21.110 GFX_ThickBevelDraw Function**C**

```
uint16_t GFX_ThickBevelDraw(
    uint16_t xL,
    uint16_t yT,
    uint16_t xR,
    uint16_t yB,
    uint16_t r1,
    uint16_t r2,
    uint8_t octant
);
```

Description

This is function `GFX_ThickBevelDraw`.

5.2.2.7.21.111 GFX_Transition Function**C**

```
GFX_STATUS GFX_Transition(
    GFX_TRANSITION_PARAMS* transitionParams,
    short width,
    short height
);
```

Description

This immediately executes the transition effect using the [GFX_TRANSITION_TYPE](#) and the given parameters.

Returns

Returns status of transition

- 0 : Transition executed successfully
- -1 : Transition not executed

5.2.2.7.21.112 GFX_TransparentColorDisable Function**C**

```
GFX_STATUS GFX_TransparentColorDisable();
```

Description

This function disables the transparent color feature used in [GFX_ImageDraw\(\)](#) and [GFX_ImagePartialDraw\(\)](#) functions.

The transparent color feature can only be enabled when the color depth used is 24 or 16 bpp.

Preconditions

Transparent color feature must be enabled at build time. This is enabled by default and can be disabled by defining the macro `GFX_CONFIG_TRANSPARENT_COLOR_DISABLE` in the system.

Returns

The status of the transparent color disable action.

Example

```
// assume ScreenBackground and RibbonIcon are valid
// image resources
// assume BLACK is a valid GFX_COLOR value

GFX_TransparentColorEnable(BLACK);
GFX_ImageDraw(0,0, (void*)&ScreenBackground);
GFX_ImageDraw(50,50, (void*)&RibbonIcon);

// disable the transparent color feature since the
// next image to render contains black pixels that
// we want to render
GFX_TransparentColorDisable();
GFX_ImageDraw(50,50, (void*)&OverlayImage);
```

5.2.2.7.21.113 GFX_TransparentColorEnable Function

C

```
GFX_STATUS GFX_TransparentColorEnable(
    GFX_COLOR color
);
```

Description

This function sets the transparent color used in [GFX_ImageDraw\(\)](#) functions and enables the transparent color feature.

When [GFX_ImageDraw\(\)](#) or [GFX_ImagePartialDraw\(\)](#) is called, any pixels in the image that matches the color value will not be rendered to the frame buffer.

The transparent color feature can only be enabled when the color depth used is 24 or 16 bpp.

Preconditions

Transparent color feature must be enabled at build time. This is enabled by default and can be disabled by defining the macro `GFX_CONFIG_TRANSPARENT_COLOR_DISABLE` in the system.

Parameters

Parameters	Description
color	the color value selected as the transparent color.

Returns

The status of the transparent color set action.

Example

```
// assume ScreenBackground and RibbonIcon are valid
// image resources
// assume BLACK is a valid GFX_COLOR value

GFX_TransparentColorEnable(BLACK);
GFX_ImageDraw(0,0, (void*)&ScreenBackground);
GFX_ImageDraw(0,0, (void*)&RibbonIcon);
```


5.2.2.7.21.114 GFX_TransparentColorGet Function

C

```
GFX_COLOR GFX_TransparentColorGet();
```

Description

This returns the current transparent color set for the transparent color feature used in [GFX_ImageDraw\(\)](#) and [GFX_ImagePartialDraw\(\)](#) functions.

The transparent color feature can only be enabled when the color depth used is 24 or 16 bpp.

Preconditions

Transparent color feature must be enabled at build time. This is enabled by default and can be disabled by defining the macro `GFX_CONFIG_TRANSPARENT_COLOR_DISABLE` in the system.

Returns

The current transparent color used.

Example

```
// assume ScreenBackground and RibbonIcon are valid
// image resources
// assume BLACK is a valid GFX_COLOR value

GFX_TransparentColorEnable(BLACK);
GFX_ImageDraw(0,0, (void*)&ScreenBackground);
GFX_ImageDraw(50,50, (void*)&RibbonIcon);

// disable the transparent color feature since the
// next image to render contains black pixels that
// we want to render
if (GFX_TransparentColorGet == BLACK)
    GFX_TransparentColorDisable();
GFX_ImageDraw(50,50, (void*)&OverlayImage);
```

5.2.2.7.21.115 GFX_TransparentColorStatusGet Function

C

```
GFX_FEATURE_STATUS GFX_TransparentColorStatusGet();
```

Description

This returns the current transparent color set for the transparent color feature used in [GFX_ImageDraw\(\)](#) and [GFX_ImagePartialDraw\(\)](#) functions.

The transparent color feature can only be enabled when the color depth used is 24 or 16 bpp.

Preconditions

Transparent color feature must be enabled at build time. This is enabled by default and can be disabled by defining the macro `GFX_CONFIG_TRANSPARENT_COLOR_DISABLE` in the system.

Returns

The current transparent feature status (see [GFX_FEATURE_STATUS](#) for details).

Example

```
// assume ScreenBackground and RibbonIcon are valid
// image resources
// assume BLACK is a valid GFX_COLOR value

if (GFX_TransparentColorStatusGet == GFX_FEATURE_DISABLED)
```

```
GFX_TransparentColorEnable(BLACK);
GFX_ImageDraw(0,0, (void*)&ScreenBackground);
GFX_ImageDraw(50,50, (void*)&RibbonIcon);

// disable the transparent color feature since the
// next image to render contains black pixels that
// we want to render
if (GFX_TransparentColorGet == BLACK)
    GFX_TransparentColorDisable();
GFX_ImageDraw(50,50, (void*)&OverlayImage);
```

5.2.2.7.22 Image Functions

5.2.2.7.22.1 ImageDecoderInit Function

C

```
void ImageDecoderInit();
```

Description

This function initializes the global variables to 0 and then initializes the driver. This must be called once before any other function of the library is called

Returns

None

Side Effects

The graphics driver will be reset

Example

```
void main(void)
{
    ImageInit();
    ...
}
```

5.2.2.7.22.2 ImageDecodeTask Function

C

```
uint8_t ImageDecodeTask();
```

Description

This function completes one small part of the image decode function

Returns

Status code - '1' means decoding is completed

- '0' means decoding is not yet completed, call this function again

Side Effects

None

Example

```
IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
while(!ImageDecodeTask());
```

5.2.2.7.22.3 ImageLoopCallbackRegister Function

C

```
void ImageLoopCallbackRegister(  
    IMG_LOOP_CALLBACK pFn  
);
```

Description

This function registers the loop callback function so that the decoder calls this function in every decoding loop. This can be used by the application program to do maintenance activities such as fetching data, updating the display, etc...

Returns

None

Side Effects

The graphics driver will be reset

Example

```
void Mainantance(void)  
{  
    ...  
}  
  
void main(void)  
{  
    ImageInit();  
    ImageLoopCallbackRegister(Mainantance);  
    ...  
}
```

5.2.2.7.22.4 IMG_vSetboundaries Function

C

```
void IMG_vSetboundaries();
```

Description

***** This is not for the user *****

This is used by the individual decoders

5.2.2.7.23 Palette Functions

5.2.2.7.23.1 IsPaletteEnabled Function

C

```
BYTE IsPaletteEnabled();
```

Description

Returns if the Palette mode is enabled or not.

Preconditions

none

Returns

Returns the palette mode status. 1 - If the palette mode is enabled 0 - If the palette mode is disabled

5.2.2.7.23.2 PaletteInit Function

C

```
void PaletteInit();
```

Description

Initializes the color look up table (CLUT).

Preconditions

none

Returns

none

Side Effects

All rendering will use the new palette entries.

5.2.2.7.23.3 RequestPaletteChange Function

C

```
void RequestPaletteChange(
    void * pPalette,
    uint16_t startEntry,
    uint16_t length
);
```

Description

Loads the palettes from the flash during vertical blanking period if possible, otherwise loads immediately.

Preconditions

Palette must be initialized by [PaletteInit\(\)](#).

Parameters

Parameters	Description
pPalette	Pointer to the palette structure
startEntry	Start entry to load (inclusive)
length	Number of entries

Returns

none

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

5.2.2.7.23.4 SetPalette Function

C

```
BYTE SetPalette(
    void * pPalette,
    uint16_t startEntry,
```

```
    uint16_t length
);
```

Description

Programs a block of palette entries starting from startEntry and until startEntry + length from the flash immediately.

Preconditions

Palette must be initialized by [PaletteInit\(\)](#).

Parameters

Parameters	Description
pPalette	Pointer to the palette structure
startEntry	Start entry to load (inclusive)
length	Number of entries

Returns

Returns the status of the palette set. 0 - Set was successful 1 - Set was not successful

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

5.2.2.7.23.5 SetPaletteBpp Function

C

```
BYTE SetPaletteBpp(
    uint8_t bpp
);
```

Description

Sets the color look up table (CLUT) number of valid entries.

Preconditions

Palette must be initialized by [PaletteInit\(\)](#).

Parameters

Parameters	Description
bpp	Bits per pixel

Returns

Returns the status of the change. 0 - Change was successful 1 - Change was not successful

Side Effects

none

5.2.2.7.23.6 SetPaletteFlash Function

C

```
BYTE SetPaletteFlash(
    PALETTE_ENTRY * pPaletteEntry,
    uint16_t startEntry,
    uint16_t length
);
```

Description

Loads the palettes from the flash immediately.

Preconditions

Palette must be initialized by [PaletteInit\(\)](#).

Parameters

Parameters	Description
pPaletteEntry	Pointer to the palette table in ROM
startEntry	Start entry to load (inclusive)
length	Number of entries

Returns

Returns the status of the palette set. 0 - Set was successful 1 - Set was not successful

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

5.2.2.7.24 Style Scheme Functions

5.2.2.7.24.1 GFX_GOL_SchemeAlphaPrecentSet Function

C

```
inline void GFX_GOL_SchemeAlphaPrecentSet(
    GFX_GOL_OBJ_SCHEME * pScheme,
    uint16_t percent
);
```

Description

This is function GFX_GOL_SchemeAlphaPrecentSet.

5.2.2.7.24.2 GFX_GOL_SchemeBackgroundColorGet Function

C

```
inline GFX_COLOR GFX_GOL_SchemeBackgroundColorGet(
    GFX_GOL_OBJ_SCHEME * pScheme
);
```

Description

This is function GFX_GOL_SchemeBackgroundColorGet.

5.2.2.7.24.3 GFX_GOL_SchemeBackgroundColorSet Function

C

```
inline void GFX_GOL_SchemeBackgroundColorSet(
    GFX_GOL_OBJ_SCHEME * pScheme,
    GFX_COLOR color
);
```

Description

This is function GFX_GOL_SchemeBackgroundColorSet.

5.2.2.7.24.4 GFX_GOL_SchemeBackgroundImageSet Function

C

```
inline void GFX_GOL_SchemeBackgroundImageSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    uint16_t left,  
    uint16_t top,  
    GFX_RESOURCE_HDR * image  
);
```

Description

This is function GFX_GOL_SchemeBackgroundImageSet.

5.2.2.7.24.5 GFX_GOL_SchemeBackgroundTypeGet Function

C

```
inline GFX_BACKGROUND_TYPE GFX_GOL_SchemeBackgroundTypeGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
);
```

Description

This is function GFX_GOL_SchemeBackgroundTypeGet.

5.2.2.7.24.6 GFX_GOL_SchemeBackgroundTypeSet Function

C

```
inline void GFX_GOL_SchemeBackgroundTypeSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_BACKGROUND_TYPE type  
);
```

Description

This is function GFX_GOL_SchemeBackgroundTypeSet.

5.2.2.7.24.7 GFX_GOL_SchemeColor0Get Function

C

```
inline GFX_COLOR GFX_GOL_SchemeColor0Get(  
    GFX_GOL_OBJ_SCHEME * pScheme  
);
```

Description

This is function GFX_GOL_SchemeColor0Get.

5.2.2.7.24.8 GFX_GOL_SchemeColor0Set Function

C

```
inline void GFX_GOL_SchemeColor0Set(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR color  
);
```

Description

This is function GFX_GOL_SchemeColor0Set.

5.2.2.7.24.9 GFX_GOL_SchemeColor1Get Function

C

```
inline GFX_COLOR GFX_GOL_SchemeColor1Get(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeColor1Get.

5.2.2.7.24.10 GFX_GOL_SchemeColor1Set Function

C

```
inline void GFX_GOL_SchemeColor1Set(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR color  
)  
;
```

Description

This is function GFX_GOL_SchemeColor1Set.

5.2.2.7.24.11 GFX_GOL_SchemeColorDisabledGet Function

C

```
inline GFX_COLOR GFX_GOL_SchemeColorDisabledGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeColorDisabledGet.

5.2.2.7.24.12 GFX_GOL_SchemeColorDisabledSet Function

C

```
inline void GFX_GOL_SchemeColorDisabledSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR disableColor  
)  
;
```

Description

This is function GFX_GOL_SchemeColorDisabledSet.

5.2.2.7.24.13 GFX_GOL_SchemeColorSet Function

C

```
inline void GFX_GOL_SchemeColorSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR color0,  
    GFX_COLOR color1,  
    GFX_COLOR disableColor  
)  
;
```

Description

This is function GFX_GOL_SchemeColorSet.

5.2.2.7.24.14 GFX_GOL_SchemeEmbossDarkColorGet Function

C

```
inline GFX_COLOR GFX_GOL_SchemeEmbossDarkColorGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeEmbossDarkColorGet.

5.2.2.7.24.15 GFX_GOL_SchemeEmbossDarkColorSet Function

C

```
inline void GFX_GOL_SchemeEmbossDarkColorSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR darkColor  
)  
;
```

Description

This is function GFX_GOL_SchemeEmbossDarkColorSet.

5.2.2.7.24.16 GFX_GOL_SchemeEmbossLightColorGet Function

C

```
inline GFX_COLOR GFX_GOL_SchemeEmbossLightColorGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeEmbossLightColorGet.

5.2.2.7.24.17 GFX_GOL_SchemeEmbossLightColorSet Function

C

```
inline void GFX_GOL_SchemeEmbossLightColorSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR lightColor  
)  
;
```

Description

This is function GFX_GOL_SchemeEmbossLightColorSet.

5.2.2.7.24.18 GFX_GOL_SchemeEmbossSet Function

C

```
inline void GFX_GOL_SchemeEmbossSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR darkColor,  
    GFX_COLOR lightColor,  
    uint16_t size  
)  
;
```

Description

This is function GFX_GOL_SchemeEmbossSet.

5.2.2.7.24.19 GFX_GOL_SchemeEmbossSizeGet Function

C

```
inline uint16_t GFX_GOL_SchemeEmbossSizeGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeEmbossSizeGet.

5.2.2.7.24.20 GFX_GOL_SchemeEmbossSizeSet Function

C

```
inline void GFX_GOL_SchemeEmbossSizeSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    uint16_t size  
)  
;
```

Description

This is function GFX_GOL_SchemeEmbossSizeSet.

5.2.2.7.24.21 GFX_GOL_SchemeFillStyleGet Function

C

```
inline GFX_FILL_STYLE GFX_GOL_SchemeFillStyleGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeFillStyleGet.

5.2.2.7.24.22 GFX_GOL_SchemeFillStyleSet Function

C

```
inline void GFX_GOL_SchemeFillStyleSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_FILL_STYLE style  
)  
;
```

Description

This is function GFX_GOL_SchemeFillStyleSet.

5.2.2.7.24.23 GFX_GOL_SchemeFontGet Function

C

```
inline GFX_RESOURCE_HDR* GFX_GOL_SchemeFontGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeFontGet.

5.2.2.7.24.24 GFX_GOL_SchemeFontSet Function

C

```
inline void GFX_GOL_SchemeFontSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_RESOURCE_HDR * font  
)  
;
```

Description

This is function GFX_GOL_SchemeFontSet.

5.2.2.7.24.25 GFX_GOL_SchemeGradientColorSet Function

C

```
inline void GFX_GOL_SchemeGradientColorSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR startColor,  
    GFX_COLOR endColor  
)  
;
```

Description

This is function GFX_GOL_SchemeGradientColorSet.

5.2.2.7.24.26 GFX_GOL_SchemeGradientEndColorGet Function

C

```
inline GFX_COLOR GFX_GOL_SchemeGradientEndColorGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeGradientEndColorGet.

5.2.2.7.24.27 GFX_GOL_SchemeGradientEndColorSet Function

C

```
inline void GFX_GOL_SchemeGradientEndColorSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR endColor  
)  
;
```

Description

This is function GFX_GOL_SchemeGradientEndColorSet.

5.2.2.7.24.28 GFX_GOL_SchemeGradientStartColorGet Function

C

```
inline GFX_COLOR GFX_GOL_SchemeGradientStartColorGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeGradientStartColorGet.

5.2.2.7.24.29 GFX_GOL_SchemeGradientStartColorSet Function

C

```
inline void GFX_GOL_SchemeGradientStartColorSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR startColor  
)  
;
```

Description

This is function GFX_GOL_SchemeGradientStartColorSet.

5.2.2.7.24.30 GFX_GOL_SchemeTextColor0Get Function

C

```
inline GFX_COLOR GFX_GOL_SchemeTextColor0Get(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeTextColor0Get.

5.2.2.7.24.31 GFX_GOL_SchemeTextColor0Set Function

C

```
inline void GFX_GOL_SchemeTextColor0Set(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR textColor  
)  
;
```

Description

This is function GFX_GOL_SchemeTextColor0Set.

5.2.2.7.24.32 GFX_GOL_SchemeTextColor1Get Function

C

```
inline GFX_COLOR GFX_GOL_SchemeTextColor1Get(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeTextColor1Get.

5.2.2.7.24.33 GFX_GOL_SchemeTextColor1Set Function

C

```
inline void GFX_GOL_SchemeTextColor1Set(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR textColor  
)  
;
```

Description

This is function GFX_GOL_SchemeTextColor1Set.

5.2.2.7.24.34 GFX_GOL_SchemeTextColorDisableGet Function

C

```
inline GFX_COLOR GFX_GOL_SchemeTextColorDisableGet(  
    GFX_GOL_OBJ_SCHEME * pScheme  
)  
;
```

Description

This is function GFX_GOL_SchemeTextColorDisableGet.

5.2.2.7.24.35 GFX_GOL_SchemeTextColorDisableSet Function

C

```
inline void GFX_GOL_SchemeTextColorDisableSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR disableColor  
)  
;
```

Description

This is function GFX_GOL_SchemeTextColorDisableSet.

5.2.2.7.24.36 GFX_GOL_SchemeTextColorSet Function

C

```
inline void GFX_GOL_SchemeTextColorSet(  
    GFX_GOL_OBJ_SCHEME * pScheme,  
    GFX_COLOR textColor0,  
    GFX_COLOR textColor1,  
    GFX_COLOR disableColor  
)  
;
```

Description

This is function GFX_GOL_SchemeTextColorSet.

5.2.2.7.25 Global Variables

5.2.2.7.25.1 IMG_bAlignCenter Variable

C

```
uint8_t IMG_bAlignCenter;
```

Description

This is variable IMG_bAlignCenter.

5.2.2.7.25.2 IMG_bDownScalingFactor Variable

C

```
uint8_t IMG_bDownScalingFactor;
```

Description

This is variable IMG_bDownScalingFactor.

5.2.2.7.25.3 IMG_bIAbortImageDecoding Variable

C

```
uint8_t IMG_bIAbortImageDecoding;
```

Description

The global variables which define the image position and size

5.2.2.7.25.4 IMG_pCurrentFile Variable

C

```
IMG_FILE * IMG_pCurrentFile;
```

Description

This is variable IMG_pCurrentFile.

5.2.2.7.25.5 IMG_pFileAPIs Variable

C

```
IMG_FILE_SYSTEM_API * IMG_pFileAPIs;
```

Description

This is variable IMG_pFileAPIs.

5.2.2.7.25.6 IMG_PixelXYColor Variable

C

```
IMG_PIXEL_XY_RGB_888 IMG_PixelXYColor;
```

Description

This is variable IMG_PixelXYColor.

5.2.2.7.25.7 IMG_pLoopCallbackFn Variable

C

```
IMG_LOOP_CALLBACK IMG_pLoopCallbackFn;
```

Description

This is variable IMG_pLoopCallbackFn.

5.2.2.7.25.8 IMG_pPixelOutput Variable

C

```
IMG_PIXEL_OUTPUT IMG_pPixelOutput;
```

Description

This is variable IMG_pPixelOutput.

5.2.2.7.25.9 IMG_wHeight Variable

C

```
uint16_t IMG_wHeight;
```

Description

This is variable IMG_wHeight.

5.2.2.7.25.10 IMG_wImageHeight Variable**C**

```
uint16_t IMG_wImageHeight;
```

Description

This is variable IMG_wImageHeight.

5.2.2.7.25.11 IMG_wImageWidth Variable**C**

```
uint16_t IMG_wImageWidth;
```

Description

This is variable IMG_wImageWidth.

5.2.2.7.25.12 IMG_wStartX Variable**C**

```
uint16_t IMG_wStartX;
```

Description

This is variable IMG_wStartX.

5.2.2.7.25.13 IMG_wStartY Variable**C**

```
uint16_t IMG_wStartY;
```

Description

This is variable IMG_wStartY.

5.2.2.7.25.14 IMG_wWidth Variable**C**

```
uint16_t IMG_wWidth;
```

Description

This is variable IMG_wWidth.

5.2.2.7.25.15 DisplayUpdatePending Variable**C**

```
volatile uint8_t DisplayUpdatePending;
```

Description

This is variable DisplayUpdatePending.

5.2.2.7.25.16 doubleBuffEnabled Variable

C

```
uint8_t doubleBuffEnabled;
```

Description

This is variable doubleBuffEnabled.

5.2.2.7.25.17 GFX_Primitive_instance Variable

C

```
GFX_Primitive_DATA GFX_Primitive_instance;
```

Description

structure to store the primitive driver structures

5.2.2.7.25.18 primitiveTaskImage Variable

C

```
PUTIMAGE_PARAM primitiveTaskImage;
```

Description

This is variable primitiveTaskImage.

5.2.2.7.26 Data Types and Constants

5.2.2.7.26.1 CHARTPARAM Structure

C

```
typedef struct {
    XCHAR * pTitle;
    XCHAR * pSmplLabel;
    XCHAR * pValLabel;
    short seriesCount;
    uint16_t smplStart;
    uint16_t smplEnd;
    uint16_t valMax;
    uint16_t valMin;
    uint16_t perMax;
    uint16_t perMin;
    GFX_COLOR * pColor;
    void * pTitleFont;
    void * pAxisLabelsFont;
    void * pGridLabelsFont;
} CHARTPARAM;
```

Description

Defines the parameters for the CHART object.

Members

Members	Description
XCHAR * pTitle;	Pointer to the Title of the chart.
XCHAR * pSmplLabel;	Pointer to the bar chart sample axis label. Depending

XCHAR * pValLabel;	Pointer to the bar chart value axis label. Depending
short seriesCount;	Number of data series that will be displayed when chart is drawn.
uint16_t smplStart;	Start point of data sample range to be displayed (minimum/default value = 1)
uint16_t smplEnd;	End point of data sample range to be displayed.
uint16_t valMax;	Maximum value of a sample that can be displayed.
uint16_t valMin;	Minimum value of a sample that can be displayed.
uint16_t perMax;	Maximum value of the percentage range that can be displayed.
uint16_t perMin;	Minimum value of the percentage range that can be displayed.
GFX_COLOR * pColor;	Pointer to the color table used to draw the chart data.
void * pTitleFont;	Pointer to the font used for the title label of the chart.
void * pAxisLabelsFont;	Pointer to the font used for X and Y axis labels.
void * pGridLabelsFont;	Pointer to the font used for X and Y axis grid labels.

5.2.2.7.26.2 DATASERIES Structure

C

```
typedef struct {
    XCHAR * pSData;
    uint16_t samples;
    BYTE show;
    uint16_t * pData;
    void * pNextData;
} DATASERIES;
```

Description

Defines a variable for the CHART object. It specifies the number of samples, pointer to the array of samples for the data series and pointer to the next data series. A member of this structure (show) is used as a flag to determine if the series is to be drawn or not. Together with the smplStart and smplEnd it will determine what kind of chart will be drawn.

Members

Members	Description
XCHAR * pSData;	Pointer to the data series name.
uint16_t samples;	Indicates the number of data samples (or data points) contained in the array specified by pData.
BYTE show;	The flag to indicate if the data series will be shown or not. If this flag is set to SHOW_DATA , the data series will be shown. If HIDE_DATA , the data series will not be shown.
uint16_t * pData;	Pointer to the array of data samples.
void * pNextData;	Pointer to the next data series. NULL if no other data series follows.

5.2.2.7.26.3 FileFeof Type

C

```
typedef bool (* FileFeof)(void *stream);
```

Description

This is type FileFeof.

5.2.2.7.26.4 FileRead Type

C

```
typedef size_t (* FileRead)(void *ptr, size_t size, size_t n, void *stream);
```

Description

Typedefs of the used File System APIs

5.2.2.7.26.5 FileSeek Type

C

```
typedef bool (* FileSeek)(void *stream, long offset, int whence);
```

Description

This is type FileSeek.

5.2.2.7.26.6 FileTell Type

C

```
typedef long (* FileTell)(void *fo);
```

Description

This is type FileTell.

5.2.2.7.26.7 FONT_ORIENTATION Enumeration

C

```
typedef enum {  
    ORIENT_HOR = 0,  
    ORIENT_VER = 1  
} FONT_ORIENTATION;
```

Description

GFX font orientation

Remarks

None.

5.2.2.7.26.8 GFX_ALIGNMENT Enumeration

C

```
typedef enum {  
    GFX_ALIGN_LEFT = 0x0C01,  
    GFX_ALIGN_HCENTER = 0x0C02,  
    GFX_ALIGN_RIGHT = 0x0C04,  
    GFX_ALIGN_TOP = 0x0C10,  
    GFX_ALIGN_VCENTER = 0x0C20,  
    GFX_ALIGN_BOTTOM = 0x0C40,  
    GFX_ALIGN_CENTER = 0x0C22,  
    GFX_ALIGN_HORIZONTAL_MASK = 0x0C07,  
    GFX_ALIGN_VERTICAL_MASK = 0x0C70  
} GFX_ALIGNMENT;
```

Description

Defines the types of text alignment. The text alignment is used in [GFX_TextStringBoxDraw\(\)](#).

Members

Members	Description
GFX_ALIGN_LEFT = 0x0C01	left aligned text
GFX_ALIGN_HCENTER = 0x0C02	horizontal center aligned text
GFX_ALIGN_RIGHT = 0x0C04	right aligned text
GFX_ALIGN_TOP = 0x0C10	top aligned text
GFX_ALIGN_VCENTER = 0x0C20	vertical center aligned text
GFX_ALIGN_BOTTOM = 0x0C40	bottom aligned text
GFX_ALIGN_CENTER = 0x0C22	vertical & horizontal center aligned text
GFX_ALIGN_HORIZONTAL_MASK = 0x0C07	horizontal mask, DO NOT document this item
GFX_ALIGN_VERTICAL_MASK = 0x0C70	vertical mask, DO NOT document this item DOM-IGNORE-END

5.2.2.7.26.9 GFX_BACKGROUND Structure**C**

```
typedef struct {
    uint16_t left;
    uint16_t top;
    GFX_BACKGROUND_TYPE type;
    GFX_RESOURCE_HDR * pImage;
    GFX_COLOR color;
} GFX_BACKGROUND;
```

Description

Structure describing the background information. Useful in refreshing the whole screen or an area of the screen. The background information, when used, can be one of the enumerated types in [GFX_BACKGROUND_TYPE](#).

- pBackGroundImage - pointer to a GFX_RESOURCE_HEADER with the type GFX_RESOURCE_TYPE_MCHP_MBITMAP.
- backGroundColor - value of the background color used when pBackGroundImage is NULL.

The pBackGroundImage has the high priority and will assume an image is used as a background when the pointer is initialized. If this is NULL the backGroundColor is assumed to be the background.

Members

Members	Description
uint16_t left;	Horizontal starting position of the background.
uint16_t top;	Horizontal starting position of the background.
GFX_BACKGROUND_TYPE type;	Specifies the type of background to use.
GFX_RESOURCE_HDR * pImage;	Pointer to the background image used. Set this to NULL if not used.
GFX_COLOR color;	If the background image is NULL, background is assumed to be this color.

5.2.2.7.26.10 GFX_BACKGROUND_TYPE Enumeration**C**

```
typedef enum {
    GFX_BACKGROUND_NONE = 0x00,
    GFX_BACKGROUND_COLOR = 0x01,
    GFX_BACKGROUND_IMAGE = 0x02,
    GFX_BACKGROUND_DISPLAY_BUFFER = 0x03
} GFX_BACKGROUND_TYPE;
```

Description

Defines the types of background information. Knowing the background information allows easier refresh of an area on the display buffer.

Members

Members	Description
GFX_BACKGROUND_NONE = 0x00	No background information set.
GFX_BACKGROUND_COLOR = 0x01	Background is set to a color.
GFX_BACKGROUND_IMAGE = 0x02	Background is set to an image.
GFX_BACKGROUND_DISPLAY_BUFFER = 0x03	Background is set to the current content of the display buffer. This requires support of GFX_PixelArrayGet().

5.2.2.7.26.11 GFX_BEVEL_RENDER_TYPE Enumeration**C**

```
typedef enum {
    GFX_DRAW_FULLBEVEL = 0xFF,
    GFX_DRAW_TOPBEVEL = 0xF0,
    GFX_DRAW_BOTTOMBEVEL = 0x0F
} GFX_BEVEL_RENDER_TYPE;
```

Description

Bevel shapes can be rendered in full or half size. The following types defines the full, top and bottom halves.

Members

Members	Description
GFX_DRAW_FULLBEVEL = 0xFF	render full bevel shape
GFX_DRAW_TOPBEVEL = 0xF0	render top bevel shape
GFX_DRAW_BOTTOMBEVEL = 0x0F	render bottom half bevel shape

5.2.2.7.26.12 GFX_COLOR Type**C**

```
typedef uint32_t GFX_COLOR;
```

Description

Data type that defines the color data. This type is dependent on the [COLOR_DEPTH](#) setting. See [COLOR_DEPTH](#).

- GFX_COLOR is type uint8_t if [COLOR_DEPTH](#) <= 8
- GFX_COLOR is type uint16_t if [COLOR_DEPTH](#) = 16
- GFX_COLOR is type uint32_t if [COLOR_DEPTH](#) = 24

5.2.2.7.26.13 GFX_DOUBLE_BUFFERING_MODE Structure**C**

```
typedef struct {
    GFX_FEATURE_STATUS gfxDoubleBufferFeature;
    GFX_FEATURE_STATUS gfxDoubleBufferRequestSync;
    GFX_FEATURE_STATUS gfxDoubleBufferFullSync;
    GFX_RECTANGULAR_AREA gfxDoubleBufferAreas[GFX_MAX_INVALIDATE_AREAS];
    uint16_t gfxUnsyncedAreaCount;
} GFX_DOUBLE_BUFFERING_MODE;
```

Description

Structure used to manage double buffering feature.

5.2.2.7.26.14 GFX_FEATURE_STATUS Enumeration

C

```
typedef enum {
    GFX_FEATURE_ENABLED = 1,
    GFX_FEATURE_DISABLED = 0
} GFX_FEATURE_STATUS;
```

Description

Defines the states of a Graphics Library feature that can nbe enabled/disabled at run time.

5.2.2.7.26.15 GFX_FILL_STYLE Enumeration

C

```
typedef enum {
    GFX_FILL_STYLE_NONE = 0x0B00,
    GFX_FILL_STYLE_COLOR,
    GFX_FILL_STYLE_ALPHA_COLOR,
    GFX_FILL_STYLE_GRADIENT_DOWN,
    GFX_FILL_STYLE_GRADIENT_UP,
    GFX_FILL_STYLE_GRADIENT_RIGHT,
    GFX_FILL_STYLE_GRADIENT_LEFT,
    GFX_FILL_STYLE_GRADIENT_DOUBLE_VER,
    GFX_FILL_STYLE_GRADIENT_DOUBLE_HOR
} GFX_FILL_STYLE;
```

Description

Defines the types of fill styles.

Members

Members	Description
GFX_FILL_STYLE_NONE = 0x0B00	no fill
GFX_FILL_STYLE_COLOR	color fill
GFX_FILL_STYLE_ALPHA_COLOR	color fill with alpha blending
GFX_FILL_STYLE_GRADIENT_DOWN	gradient, vertical-down direction
GFX_FILL_STYLE_GRADIENT_UP	gradient, vertical-up direction
GFX_FILL_STYLE_GRADIENT_RIGHT	gradient, horizontal-right direction
GFX_FILL_STYLE_GRADIENT_LEFT	gradient, horizontal-left direction
GFX_FILL_STYLE_GRADIENT_DOUBLE_VER	gradient, vertical-up/down direction
GFX_FILL_STYLE_GRADIENT_DOUBLE_HOR	gradient, horizontal-left/right direction

5.2.2.7.26.16 GFX_FONT_ANTIALIAS_TYPE Enumeration

C

```
typedef enum {
    GFX_FONT_ANTIALIAS_OPAQUE = 0,
    GFX_FONT_ANTIALIAS_TRANSLUCENT = 1
} GFX_FONT_ANTIALIAS_TYPE;
```

Description

Transparency type when rendering characters with anti-aliasing.

Members

Members	Description
GFX_FONT_ANTIALIAS_OPAQUE = 0	Mid colors are calculated only once while rendering each character. This is ideal for rendering text over a constant background.
GFX_FONT_ANTIALIAS_TRANSLUCENT = 1	Mid colors are calculated for every necessary pixel. This feature is useful when rendering text over an image or when the background is not one flat color.

5.2.2.7.26.17 GFX_FONT_CURRENT Structure**C**

```
typedef struct {
    GFX_RESOURCE_HDR * pFont;
    GFX_FONT_HEADER header;
    GFX_ALIGNMENT alignment;
    FONT_ORIENTATION orientation;
    uint8_t antiAliasType;
} GFX_FONT_CURRENT;
```

Description

GFX current font

Members

Members	Description
GFX_RESOURCE_HDR * pFont;	pointer to the currently set font
GFX_FONT_HEADER header;	copy of the currently set font header
GFX_ALIGNMENT alignment;	alignment of the font
FONT_ORIENTATION orientation;	orientation of the font
uint8_t antiAliasType;	anti-alias type set

Remarks

None.

5.2.2.7.26.18 GFX_FONT_GLYPH_ENTRY Structure**C**

```
typedef struct {
    uint8_t width;
    uint8_t offsetLSB;
    uint16_t offsetMSB;
} GFX_FONT_GLYPH_ENTRY;
```

Description

Structures describing the glyph entry.

Members

Members	Description
uint8_t width;	width of the glyph
uint8_t offsetLSB;	Least Significant uint8_t of the glyph location offset
uint16_t offsetMSB;	Most Significant (2) uint8_ts of the glyph location offset

5.2.2.7.26.19 GFX_FONT_GLYPH_ENTRY_EXTENDED Structure

C

```
typedef struct {
    uint32_t offset;
    uint16_t cursorAdvance;
    uint16_t glyphWidth;
    int16_t xAdjust;
    int16_t yAdjust;
} GFX_FONT_GLYPH_ENTRY_EXTENDED;
```

Description

This is type GFX_FONT_GLYPH_ENTRY_EXTENDED.

Members

Members	Description
uint32_t offset;	Offset Order: LSW_LSB LSW_MSB MSW_MSB MSW_MSB
uint16_t cursorAdvance;	x-value by which cursor has to advance after rendering the glyph
uint16_t glyphWidth;	width of the glyph
int16_t xAdjust;	x-position is adjusted as per this signed number
int16_t yAdjust;	y-position is adjusted as per this signed number

5.2.2.7.26.20 GFX_FONT_HEADER Structure

C

```
typedef struct {
    uint8_t fontID;
    uint8_t extendedGlyphEntry : 1;
    uint8_t res1 : 1;
    uint8_t bpp : 2;
    uint8_t orientation : 2;
    uint8_t res2 : 2;
    uint16_t firstChar;
    uint16_t lastChar;
    uint16_t height;
} GFX_FONT_HEADER;
```

Description

Structure describing the font header.

Members

Members	Description
uint8_t fontID;	User assigned value
uint8_t extendedGlyphEntry : 1;	Extended Glyph entry flag. When set font has extended glyph feature enabled.
uint8_t res1 : 1;	Reserved for future use (must be set to 0)

5.2.2.7.26.21 GFX_FONT_OUTCHAR Structure

C

```
typedef struct {
    GFX_FONT_SPACE GFX_FONT_GLYPH_ENTRY * pChTable;
    GFX_FONT_SPACE GFX_FONT_GLYPH_ENTRY_EXTENDED * pChTableExtended;
    uint8_t chImage[GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE];
    int16_t chEffectiveGlyphWidth;
    uint8_t bpp;
    int16_t chGlyphWidth;
```

```

    GFX_FONT_SPACE uint8_t * pChImage;
    int16_t xAdjust;
    int16_t yAdjust;
    int16_t xWidthAdjust;
    int16_t heightOvershoot;
} GFX_FONT_OUTCHAR;

```

Description

Structure for character information when rendering the character.

5.2.2.7.26.22 GFX_GOL_ANALOGCLOCK Structure

C

```

typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    short radius;
    short centerx;
    short centery;
    short valueS;
    short prev_valueS;
    short valueM;
    short prev_valueM;
    short valueH;
    short prev_valueH;
    void * pBitmap;
} GFX_GOL_ANALOGCLOCK;

```

Description

Defines the parameters required for a clock Object. The following relationships of the parameters determines the general shape of the clock:

1. centerx and centery determine the middle of the clock.
2. radius defines the radius of the clock.
4. *pBitmap points to the background image for the analog clock.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
short radius;	Radius of the clock.
short centerx;	center location in X-axis.
short centery;	center location in Y-axis.
short valueS;	time in Second
short prev_valueS;	previous time in Second
short valueM;	time in Minute
short prev_valueM;	previous time in Minute
short valueH;	time in Hour
short prev_valueH;	previous time in Hour
void * pBitmap;	Pointer to bitmap used.

5.2.2.7.26.23 GFX_GOL_BUTTON Structure

C

```

typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint16_t radius;

```



```

uint16_t textWidth;
uint16_t textHeight;
GFX_XCHAR * pText;
GFX_ALIGNMENT alignment;
GFX_RESOURCE_HDR * pPressImage;
GFX_RESOURCE_HDR * pReleaseImage;
GFX_COLOR previousAlphaColor;
} GFX_GOL_BUTTON;

```

Description

Defines the parameters required for a GFX_GOL_BUTTON Object. The following relationships of the parameters determines the general shape of the GFX_GOL_BUTTON:

1. Width is determined by right - left.
2. Height is determined by top - bottom.
3. Radius - specifies if the GFX_GOL_BUTTON will have a rounded edge. If zero then the GFX_GOL_BUTTON will have sharp (cornered) edge.
4. If $2 * \text{radius} = \text{height} = \text{width}$, the GFX_GOL_BUTTON is a circular GFX_GOL_BUTTON.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t radius;	Radius for rounded buttons.
uint16_t textWidth;	Computed text width, done at creation.
uint16_t textHeight;	Computed text height, done at creation.
GFX_XCHAR * pText;	Pointer to the text used.
GFX_ALIGNMENT alignment;	text alignment
GFX_RESOURCE_HDR * pPressImage;	Pointer to bitmap used.
GFX_RESOURCE_HDR * pReleaseImage;	Pointer to bitmap used.

5.2.2.7.26.24 GFX_GOL_BUTTON_STATE Enumeration

C

```

typedef enum {
    GFX_GOL_BUTTON_FOCUSED_STATE = 0x0001,
    GFX_GOL_BUTTON_DISABLED_STATE = 0x0002,
    GFX_GOL_BUTTON_PRESSED_STATE = 0x0004,
    GFX_GOL_BUTTON_TOGGLE_STATE = 0x0008,
    GFX_GOL_BUTTON_TWOTONE_STATE = 0x0100,
    GFX_GOL_BUTTON_NOPANEL_STATE = 0x0200,
    GFX_GOL_BUTTON_DRAW_FOCUS_STATE = 0x2000,
    GFX_GOL_BUTTON_DRAW_STATE = 0x4000,
    GFX_GOL_BUTTON_HIDE_STATE = 0x8000
} GFX_GOL_BUTTON_STATE;

```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_BUTTON_FOCUSED_STATE = 0x0001	Bit for focus state.
GFX_GOL_BUTTON_DISABLED_STATE = 0x0002	Bit for disabled state.

GFX_GOL_BUTTON_PRESSED_STATE = 0x0004	Bit for press state.
GFX_GOL_BUTTON_TOGGLE_STATE = 0x0008	Bit to indicate object will have a toggle behavior.
GFX_GOL_BUTTON_TWOTONE_STATE = 0x0100	Bit to indicate the object is a two tone type.
GFX_GOL_BUTTON_NOPANEL_STATE = 0x0200	Bit to indicate the object will be drawn without a panel (for faster drawing when the object image used is larger than the object's panel).
GFX_GOL_BUTTON_DRAW_FOCUS_STATE = 0x2000	Bit to indicate focus must be redrawn.
GFX_GOL_BUTTON_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_BUTTON_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.25 GFX_GOL_CHART Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    CHARTPARAM prm;
    DATASERIES * pChData;
} GFX_GOL_CHART;
```

Description

Defines the parameters required for a chart Object.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
CHARTPARAM prm;	Structure for the parameters of the chart.
DATASERIES * pChData;	Pointer to the first chart data series in the link list of data series.

5.2.2.7.26.26 GFX_GOL_CHECKBOX Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint16_t textHeight;
    GFX_XCHAR * pText;
    GFX_ALIGNMENT alignment;
} GFX_GOL_CHECKBOX;
```

Description

The structure contains check box data

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t textHeight;	Pre-computed text height
GFX_XCHAR * pText;	Pointer to text
GFX_ALIGNMENT alignment;	text alignment

5.2.2.7.26.27 GFX_GOL_CHECKBOX_STATE Enumeration

C

```
typedef enum {
    GFX_GOL_CHECKBOX_FOCUSED_STATE = 0x0001,
    GFX_GOL_CHECKBOX_DISABLED_STATE = 0x0002,
    GFX_GOL_CHECKBOX_CHECKED_STATE = 0x0004,
    GFX_GOL_CHECKBOX_DRAW_CHECK_STATE = 0x1000,
    GFX_GOL_CHECKBOX_DRAW_FOCUS_STATE = 0x2000,
    GFX_GOL_CHECKBOX_DRAW_STATE = 0x4000,
    GFX_GOL_CHECKBOX_HIDE_STATE = 0x8000
} GFX_GOL_CHECKBOX_STATE;
```

Description

- Object States Definition:

```
#define      GFX_GOL_CHECKBOX_FOCUSED_STATE      0x0001      //      Focus      state      #define
GFX_GOL_CHECKBOX_DISABLED_STATE 0x0002 // Disabled state #define GFX_GOL_CHECKBOX_CHECKED_STATE
0x0004 // Checked state #define GFX_GOL_CHECKBOX_HIDE_STATE 0x8000 // Check box must be removed from screen
#define      GFX_GOL_CHECKBOX_DRAW_FOCUS_STATE  0x2000      //      Focus      must      be      redrawn      #define
GFX_GOL_CHECKBOX_DRAW_STATE    0x4000      //      Whole      check      box      must      be      redrawn      #define
GFX_GOL_CHECKBOX_DRAW_CHECK_STATE 0x1000 // Check box mark should be redrawn
```

Members

Members	Description
GFX_GOL_CHECKBOX_FOCUSED_STATE = 0x0001	Bit for focus state.
GFX_GOL_CHECKBOX_DISABLED_STATE = 0x0002	Bit to indicate object is disabled
GFX_GOL_CHECKBOX_CHECKED_STATE = 0x0004	Bit to indicate object is checked
GFX_GOL_CHECKBOX_DRAW_CHECK_STATE = 0x1000	Bit to indicate the check must be redrawn.
GFX_GOL_CHECKBOX_DRAW_FOCUS_STATE = 0x2000	Bit to indicate focus must be redrawn.
GFX_GOL_CHECKBOX_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_CHECKBOX_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.28 GFX_GOL_CUSTOMCONTROL Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint8_t instance;
    uint16_t ID;
    void * pNxtObj;
    GFX_GOL_OBJ_TYPE type;
    uint16_t state;
    uint16_t left;
    uint16_t top;
    uint16_t right;
    uint16_t bottom;
    GFX_GOL_OBJ_SCHEME * pGolScheme;
```

```

DRAW_FUNC draw;
FREE_FUNC FreeObj;
ACTIONGET_FUNC actionGet;
ACTIONSET_FUNC actionSet;
uint16_t pos;
uint16_t prevPos;
} GFX_GOL_CUSTOMCONTROL;

```

Description

The structure contains data for the control

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
uint16_t ID;	instance unique ID
void * pNxtObj;	a pointer to the next object in the linked list
GFX_GOL_OBJ_TYPE type;	must be set to OBJ_CUSTOM
uint16_t state;	state
uint16_t left;	left border
uint16_t top;	top border
uint16_t right;	right border
uint16_t bottom;	bottom border
GFX_GOL_OBJ_SCHEME * pGolScheme;	the style scheme used
FREE_FUNC FreeObj;	function pointer to the object free function
ACTIONGET_FUNC actionGet;	function pointer to the object message function
ACTIONSET_FUNC actionSet;	function pointer to the object default message function
uint16_t pos;	current position
uint16_t prevPos;	previous position

5.2.2.7.26.29 GFX_GOL_DIGITALMETER Structure

C

```

typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint16_t textHeight;
    uint32_t Cvalue;
    uint32_t Pvalue;
    uint8_t NoOfDigits;
    uint8_t DotPos;
    GFX_ALIGNMENT alignment;
} GFX_GOL_DIGITALMETER;

```

Description

Digital meter parameters

DigitalMeter properties.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
uint16_t textHeight;	Pre-computed text height
uint32_t Cvalue;	Current value
uint32_t Pvalue;	Previous value

uint8_t NoOfDigits;	Number of digits to be displayed
uint8_t DotPos;	Position of decimal point
GFX_ALIGNMENT alignment;	text alignment

Remarks

None.

5.2.2.7.26.30 GFX_GOL_DIGITALMETER_STATE Enumeration

C

```
typedef enum {
    GFX_GOL_DIGITALMETER_DISABLED_STATE = 0x0002,
    GFX_GOL_DIGITALMETER_FRAME_STATE = 0x0010,
    GFX_GOL_DIGITALMETER_DRAW_STATE = 0x4000,
    GFX_GOL_DIGITALMETER_UPDATE_STATE = 0x2000,
    GFX_GOL_DIGITALMETER_HIDE_STATE = 0x8000
} GFX_GOL_DIGITALMETER_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_DIGITALMETER_DISABLED_STATE = 0x0002	Bit for disabled state.
GFX_GOL_DIGITALMETER_FRAME_STATE = 0x0010	Bit to indicate frame is displayed.
GFX_GOL_DIGITALMETER_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_DIGITALMETER_UPDATE_STATE = 0x2000	Bit to indicate that only text must be redrawn.
GFX_GOL_DIGITALMETER_HIDE_STATE = 0x8000	Bit to remove object from screen.

5.2.2.7.26.31 GFX_GOL_DRAW_CALLBACK_FUNC Type

C

```
typedef bool (* GFX_GOL_DRAW_CALLBACK_FUNC)(void);
```

Description

This callback function is implemented by the application. This is called inside the [GFX_GOL_ObjectListDraw\(\)](#) function when the drawing of objects in the active list is completed.

Any application specific rendering must be performed on this callback function so no object rendering will be affected by the application calls to primitive rendering functions. Application setting the drawing color, line style, fill style, text string cursor position and current font will not affect the object rendering. This is also the safe place to modify the active list.

When the application has performed its own primitive rendering calls, this function must return true to inform the [GFX_GOL_ObjectListDraw\(\)](#) that it is done rendering and checking for object drawing or redrawing can continue.

Preconditions

None.

Returns

true - is returned when application rendering is done. false - is returned when application rendering is not yet finished.

Example

None.

5.2.2.7.26.32 GFX_GOL_EDITBOX Structure**C**

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint16_t textHeight;
    GFX_XCHAR * pText;
    GFX_ALIGNMENT alignment;
    uint16_t charMax;
    uint16_t length;
} GFX_GOL_EDITBOX;
```

Description

Defines the parameters required for a Edit Box Object.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t textHeight;	Pre-computed text height.
GFX_XCHAR * pText;	Pointer to text buffer.
GFX_ALIGNMENT alignment;	text alignment
uint16_t charMax;	Maximum number of characters in the edit box.
uint16_t length;	Current text length.

5.2.2.7.26.33 GFX_GOL_GROUPBOX Structure**C**

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint16_t textWidth;
    uint16_t textHeight;
    GFX_XCHAR * pText;
    GFX_ALIGNMENT alignment;
} GFX_GOL_GROUPBOX;
```

Description

Defines the parameters required for a group box Object. The textwidth and textHeight is not checked with the actual dimension of the object. Clipping is not supported in group box object. It is possible for the text to exceed the dimension of the Object.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
uint16_t textWidth;	Pre-computed text width.
uint16_t textHeight;	Pre-computed text height.
GFX_XCHAR * pText;	Text string used.
GFX_ALIGNMENT alignment;	text alignment

5.2.2.7.26.34 GFX_GOL_GROUPBOX_STATE Enumeration

C

```
typedef enum {
    GFX_GOL_GROUPBOX_DISABLED_STATE = 0x0002,
    GFX_GOL_GROUPBOX_DRAW_STATE = 0x4000,
    GFX_GOL_GROUPBOX_HIDE_STATE = 0x8000
} GFX_GOL_GROUPBOX_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_GROUPBOX_DISABLED_STATE = 0x0002	Bit for disabled state
GFX_GOL_GROUPBOX_DRAW_STATE = 0x4000	Bit to indicate group box must be redrawn
GFX_GOL_GROUPBOX_HIDE_STATE = 0x8000	Bit to remove object from screen

5.2.2.7.26.35 GFX_GOL_LISTBOX Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    GFX_GOL_LISTITEM * pItemList;
    GFX_GOL_LISTITEM * pFocusItem;
    uint16_t itemsNumber;
    int16_t scrolly;
    int16_t textHeight;
    GFX_ALIGNMENT alignment;
} GFX_GOL_LISTBOX;
```

Description

Defines the parameters required for a list box Object.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
GFX_GOL_LISTITEM * pItemList;	Pointer to the list of items.
GFX_GOL_LISTITEM * pFocusItem;	Pointer to the focused item.
uint16_t itemsNumber;	Number of items in the list box.
int16_t scrolly;	Scroll displacement for the list.
int16_t textHeight;	Pre-computed text height.
GFX_ALIGNMENT alignment;	items alignment

5.2.2.7.26.36 GFX_GOL_LISTBOX_ITEM_STATUS Enumeration

C

```
typedef enum {
    GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED = 0x0001,
    GFX_GOL_LISTBOX_ITEM_STATUS_REDRAW = 0x0002
} GFX_GOL_LISTBOX_ITEM_STATUS;
```

Description

Bit definitions for the status of an item

Members

Members	Description
GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED = 0x0001	define LB_STS_SELECTED 0x0001 // Item is selected.
GFX_GOL_LISTBOX_ITEM_STATUS_REDRAW = 0x0002	define LB_STS_REDRAW 0x0002 // Item is to be redrawn.

5.2.2.7.26.37 GFX_GOL_LISTBOX_STATE Enumeration

C

```
typedef enum {
    GFX_GOL_LISTBOX_FOCUSED_STATE = 0x0001,
    GFX_GOL_LISTBOX_DISABLED_STATE = 0x0002,
    GFX_GOL_LISTBOX_SINGLE_SELECT_STATE = 0x0010,
    GFX_GOL_LISTBOX_DRAW_ITEMS_STATE = 0x1000,
    GFX_GOL_LISTBOX_DRAW_FOCUS_STATE = 0x2000,
    GFX_GOL_LISTBOX_DRAW_STATE = 0x4000,
    GFX_GOL_LISTBOX_HIDE_STATE = 0x8000
} GFX_GOL_LISTBOX_STATE;
```

Description

This is type GFX_GOL_LISTBOX_STATE.

Members

Members	Description
GFX_GOL_LISTBOX_FOCUSED_STATE = 0x0001	Bit for focus state.
GFX_GOL_LISTBOX_DISABLED_STATE = 0x0002	Bit for disabled state.
GFX_GOL_LISTBOX_SINGLE_SELECT_STATE = 0x0010	Bit to indicate only one item can be selected.
GFX_GOL_LISTBOX_DRAW_ITEMS_STATE = 0x1000	Bit to indicate selected items of the object must be redrawn.
GFX_GOL_LISTBOX_DRAW_FOCUS_STATE = 0x2000	Bit to indicate focus must be redrawn.
GFX_GOL_LISTBOX_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_LISTBOX_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.38 GFX_GOL_LISTITEM Structure

C

```
typedef struct {
    void * pPrevItem;
    void * pNextItem;
    GFX_GOL_LISTBOX_ITEM_STATUS status;
    GFX_XCHAR * pText;
    GFX_RESOURCE_HDR * pImage;
    uint16_t data;
} GFX_GOL_LISTITEM;
```

Description

Defines the parameters required for a list item used in list box.

Members

Members	Description
void * pPrevItem;	Pointer to the next item
void * pNextItem;	Pointer to the next item
GFX_GOL_LISTBOX_ITEM_STATUS status;	Specifies the status of the item.
GFX_XCHAR * pText;	Pointer to the text for the item
GFX_RESOURCE_HDR * pImage;	Pointer to the image used
uint16_t data;	Some data associated with the item

5.2.2.7.26.39 GFX_GOL_MESSAGE Structure**C**

```
typedef struct {
    uint8_t type;
    uint8_t uiEvent;
    int16_t param1;
    int16_t param2;
} GFX_GOL_MESSAGE;
```

Description

GFX_GOL_MESSAGE

Specifies message structure used in the library.

- The types must be one of the [INPUT_DEVICE_TYPE](#):
 - TYPE_UNKNOWN
 - TYPE_KEYBOARD
 - TYPE_TOUCHSCREEN
 - TYPE_MOUSE
- uiEvent must be one of the [INPUT_DEVICE_EVENT](#).
 - for touch screen:
 - EVENT_INVALID
 - EVENT_MOVE
 - EVENT_PRESS
 - EVENT_STILLPRESS
 - EVENT_RELEASE
 - for keyboard:
 - EVENT_KEYSCAN (param2 contains scan code)
 - EVENT_KEYCODE (param2 contains character code)
- param1:
 - for touch screen is the x position
 - for keyboard ID of object receiving the message
- param2
 - for touch screen y position
 - for keyboard scan or key code

Members

Members	Description
uint8_t type;	Type of input device.
uint8_t uiEvent;	The generic events for input device.
int16_t param1;	Parameter 1, definition and usage is dependent on the type of input device.
int16_t param2;	Parameter 2, definition and usage is dependent on the type of input device.

Remarks

None.

5.2.2.7.26.40 GFX_GOL_METER Structure**C**

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    GFX_XCHAR * pText;
    GFX_GOL_METER_DRAW_TYPE type;
    int16_t value;
    int16_t minValue;
    int16_t maxValue;
    int16_t xCenter;
    int16_t yCenter;
    int16_t radius;
    int16_t xPos;
    int16_t yPos;
    GFX_COLOR color1;
    GFX_COLOR color2;
    GFX_COLOR color3;
    GFX_COLOR color4;
    GFX_COLOR color5;
    GFX_COLOR color6;
    GFX_RESOURCE_HDR * pTitleFont;
    GFX_RESOURCE_HDR * pValueFont;
} GFX_GOL_METER;
```

Description

Defines the parameters required for a meter Object. Depending on the type selected the meter is drawn with the defined shape parameters and values set on the given fields.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER).
GFX_XCHAR * pText;	The text label of the meter.
GFX_GOL_METER_DRAW_TYPE type;	sets the type of the meter
int16_t value;	Current value of the meter.
int16_t minValue;	minimum value the meter can display
int16_t maxValue;	maximum value the meter can display (range is maxValue - minValue)
int16_t xCenter;	The x coordinate center position. This is computed automatically.
int16_t yCenter;	The y coordinate center position. This is computed automatically.
int16_t radius;	Radius of the meter, also defines the needle length.
int16_t xPos;	The current x position of the needle. This is computed automatically.
int16_t yPos;	The current y position of the needle. This is computed automatically.
GFX_COLOR color1;	Arc1 and scale1 color parameter.

GFX_COLOR color2;	Arc2 and scale2 color parameter
GFX_COLOR color3;	Arc3 and scale3 color parameter
GFX_COLOR color4;	Arc4 and scale4 color parameter
GFX_COLOR color5;	Arc5 and scale5 color parameter
GFX_COLOR color6;	Arc6 and scale6 color parameter
GFX_RESOURCE_HDR * pTitleFont;	Pointer to the font used in the title of the meter
GFX_RESOURCE_HDR * pValueFont;	Pointer to the font used in the current reading (if displayed) of the meter

5.2.2.7.26.41 GFX_GOL_METER_DRAW_TYPE Enumeration

C

```
typedef enum {
    GFX_GOL_METER_WHOLE_TYPE = 0,
    GFX_GOL_METER_HALF_TYPE = 1,
    GFX_GOL_METER_QUARTER_TYPE = 2
} GFX_GOL_METER_DRAW_TYPE;
```

Description

- Meter Types Definition: MTR_WHOLE_TYPE will use all the arc colors (color1 to color6) MTR_HALF_TYPE will use arc colors (color5, color4, color3, color2) MTR_QUARTER_TYPE will use arc colors (color3, color2)

Members

Members	Description
GFX_GOL_METER_WHOLE_TYPE = 0	draw circular meter
GFX_GOL_METER_HALF_TYPE = 1	draw semi circle meter
GFX_GOL_METER_QUARTER_TYPE = 2	draw quarter of a circle meter

5.2.2.7.26.42 GFX_GOL_METER_STATE Enumeration

C

```
typedef enum {
    GFX_GOL_METER_DISABLED_STATE = 0x0002,
    GFX_GOL_METER_RING_STATE = 0x0004,
    GFX_GOL_METER_ACCURACY_STATE = 0x0008,
    GFX_GOL_METER_UPDATE_DRAW_STATE = 0x1000,
    GFX_GOL_METER_DRAW_STATE = 0x4000,
    GFX_GOL_METER_HIDE_STATE = 0x8000
} GFX_GOL_METER_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_METER_DISABLED_STATE = 0x0002	Bit for disabled state.
GFX_GOL_METER_RING_STATE = 0x0004	Bit for press state.
GFX_GOL_METER_ACCURACY_STATE = 0x0008	Bit to indicate object will have a toggle behavior.
GFX_GOL_METER_UPDATE_DRAW_STATE = 0x1000	Bit to indicate hand update only.
GFX_GOL_METER_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.

GFX_GOL_METER_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.43 GFX_GOL_OBJ_HEADER Structure

C

```
typedef struct {
    uint16_t ID;
    void * pNxtObj;
    GFX_GOL_OBJ_TYPE type;
    uint16_t state;
    uint16_t left;
    uint16_t top;
    uint16_t right;
    uint16_t bottom;
    GFX_GOL_OBJ_SCHEME * pGolScheme;
    DRAW_FUNC DrawObj;
    FREE_FUNC FreeObj;
    ACTIONGET_FUNC actionGet;
    ACTIONSET_FUNC actionSet;
    bool imageStored;
} GFX_GOL_OBJ_HEADER;
```

Description

GFX_GOL_OBJ_HEADER

This structure defines the Graphics Object Layer header used in all objects in the Graphics Library.

Members

Members	Description
uint16_t ID;	Unique id assigned for referencing.
void * pNxtObj;	A pointer to the next object.
GFX_GOL_OBJ_TYPE type;	Identifies the type of GOL object.
uint16_t state;	State of object.
uint16_t left;	Left position of the Object.
uint16_t top;	Top position of the Object.
uint16_t right;	Right position of the Object.
uint16_t bottom;	Bottom position of the Object.
GFX_GOL_OBJ_SCHEME * pGolScheme;	Pointer to the scheme used.
DRAW_FUNC DrawObj;	function pointer to the object's draw function.
FREE_FUNC FreeObj;	function pointer to the object's free function.
ACTIONGET_FUNC actionGet;	function pointer to the object's action get function.
ACTIONSET_FUNC actionSet;	function pointer to the object's action set function.
bool imageStored;	Used for alpha blending

Remarks

None.

5.2.2.7.26.44 GFX_GOL_OBJ_SCHEME Structure

C

```
typedef struct {
    GFX_COLOR EmbossDkColor;
    GFX_COLOR EmbossLtColor;
    GFX_COLOR TextColor0;
    GFX_COLOR TextColor1;
```

```

GFX_COLOR TextColorDisabled;
GFX_COLOR Color0;
GFX_COLOR Color1;
GFX_COLOR ColorDisabled;
GFX_RESOURCE_HDR * pFont;
GFX_FILL_STYLE fillStyle;
GFX_COLOR CommonBkColor;
uint16_t CommonBkLeft;
uint16_t CommonBkTop;
GFX_BACKGROUND_TYPE CommonBkType;
GFX_RESOURCE_HDR * pCommonBkImage;
uint16_t AlphaValue;
GFX_COLOR gradientStartColor;
GFX_COLOR gradientEndColor;
uint16_t EmbossSize;
} GFX_GOL_OBJ_SCHEME;

```

Description

GFX_GOL_OBJ_SCHEME

This structure specifies the style scheme components of an object. All objects will use the style scheme when rendering. Refer to specific object documentation on how the style scheme colors are utilized by the object.

The style scheme allows objects to show 3D effects as well as feedback to users. For example, in Button objects, a press and release effect on the Buttons are easily shown by manipulating the colors when the object is drawn with a pressed state and released state. The style scheme also allows effects such as gradients and alpha blending. When using alpha blending, the style scheme requires objects to be associated with background information.

A background can be a flat color background or an image. The usage of a background requires the background dimension to be larger than the object. In other words, the object should be drawn within the background dimension. Multiple objects can share a common background. As long as all the objects are drawn within the dimension of the background they can share a common background.

The supported background types are (See [GFX_BACKGROUND_TYPE](#)):

- **GFX_BACKGROUND_COLOR** - this type will set the common background to be a flat color. The color used is specified by CommonBkType.
- **GFX_BACKGROUND_IMAGE** - this type will set the common background to be an image. The image used is specified by pCommonBkImage.

When an object is associated with a background, it can be easily hidden or redrawn with some effects (for example alpha blending with the background). The background information allows the redrawing of the background with the object without the need to manually refreshing the background using primitive calls by the application.

Members

Members	Description
GFX_COLOR EmbossDkColor;	Emboss dark color used for 3d effect.
GFX_COLOR EmbossLtColor;	Emboss light color used for 3d effect.
GFX_COLOR TextColor0;	Character color 0 used for objects that supports text.
GFX_COLOR TextColor1;	Character color 1 used for objects that supports text.
GFX_COLOR TextColorDisabled;	Character color used when object is in a disabled state.
GFX_COLOR Color0;	Color 0 usually assigned to an Object state.
GFX_COLOR Color1;	Color 1 usually assigned to an Object state.
GFX_COLOR ColorDisabled;	Color used when an Object is in a disabled state.
GFX_RESOURCE_HDR * pFont;	Font selected for the scheme.
GFX_FILL_STYLE fillStyle;	must be set to a gradient type when using gradient
GFX_COLOR CommonBkColor;	Background color used to hide Objects.

uint16_t CommonBkLeft;	Horizontal starting position of the background.
uint16_t CommonBkTop;	Horizontal starting position of the background.
GFX_BACKGROUND_TYPE CommonBkType;	Specifies the type of background to use.
GFX_RESOURCE_HDR * pCommonBkImage;	Pointer to the background image used. Set this
uint16_t AlphaValue;	Alpha value used for alpha blending
GFX_COLOR gradientStartColor;	start color of the gradient fill
GFX_COLOR gradientEndColor;	end color of the gradient fill
uint16_t EmbossSize;	Emboss size of the panel for 3-D effect. Set to zero if not used.

Remarks

None.

5.2.2.7.26.45 GFX_GOL_OBJ_TYPE Enumeration

C

```
typedef enum {
    GFX_GOL_ANALOGCLOCK_TYPE,
    GFX_GOL_BUTTON_TYPE,
    GFX_GOL_CHART_TYPE,
    GFX_GOL_CHECKBOX_TYPE,
    GFX_GOL_DIGITALMETER_TYPE,
    GFX_GOL_EDITBOX_TYPE,
    GFX_GOL_GRID_TYPE,
    GFX_GOL_GROUPBOX_TYPE,
    GFX_GOL_LISTBOX_TYPE,
    GFX_GOL_METER_TYPE,
    GFX_GOL_PICTURECONTROL_TYPE,
    GFX_GOL_PROGRESSBAR_TYPE,
    GFX_GOL_RADIOBUTTON_TYPE,
    GFX_GOL_SCROLLBAR_TYPE,
    GFX_GOL_STATICTEXT_TYPE,
    GFX_GOL_TEXTENTRY_TYPE,
    GFX_GOL_WINDOW_TYPE,
    GFX_GOL_CUSTOM_TYPE,
    GFX_GOL_UNKNOWN_TYPE
} GFX_GOL_OBJ_TYPE;
```

Description

GFX_GOL_OBJ_TYPE

This enumeration specifies the different object types used in the library.

Members

Members	Description
GFX_GOL_ANALOGCLOCK_TYPE	Type defined for Analog Clock Object.
GFX_GOL_BUTTON_TYPE	Type defined for Button Object.
GFX_GOL_CHART_TYPE	Type defined for Chart Object.
GFX_GOL_CHECKBOX_TYPE	Type defined for Check Box Object.
GFX_GOL_DIGITALMETER_TYPE	Type defined for Digital Meter Object.
GFX_GOL_EDITBOX_TYPE	Type defined for Edit Box Object.
GFX_GOL_GRID_TYPE	Type defined for Grid Object.
GFX_GOL_GROUPBOX_TYPE	Type defined for Group Box Object.
GFX_GOL_LISTBOX_TYPE	Type defined for List Box Object.

GFX_GOL_METER_TYPE	Type defined for Meter Object.
GFX_GOL_PICTURECONTROL_TYPE	Type defined for Picture Control Object.
GFX_GOL_PROGRESSBAR_TYPE	Type defined for Progress Bar Object.
GFX_GOL_RADIOBUTTON_TYPE	Type defined for Radio Button Object.
GFX_GOL_SCROLLBAR_TYPE	Type defined for Slider or Scroll Bar Object.
GFX_GOL_STATICTEXT_TYPE	Type defined for Static Text Object.
GFX_GOL_TEXTENTRY_TYPE	Type defined for Text-Entry Object.
GFX_GOL_WINDOW_TYPE	Type defined for Window Object.
GFX_GOL_CUSTOM_TYPE	Type defined for Custom Object.
GFX_GOL_UNKNOWN_TYPE	Type is undefined and not supported by the library.

Remarks

None.

5.2.2.7.26.46 GFX_GOL_PROGRESSBAR Structure**C**

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint16_t pos;
    uint16_t prevPos;
    uint16_t range;
} GFX_GOL_PROGRESSBAR;
```

Description

The structure contains data for the progress bar

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t pos;	Current progress position.
uint16_t prevPos;	Previous progress position.
uint16_t range;	Sets the range of the object.

5.2.2.7.26.47 GFX_GOL_PROGRESSBAR_STATE Enumeration**C**

```
typedef enum {
    GFX_GOL_PROGRESSBAR_DISABLED_STATE = 0x0002,
    GFX_GOL_PROGRESSBAR_VERTICAL_STATE = 0x0004,
    GFX_GOL_PROGRESSBAR_NOPROGRESS_STATE = 0x0008,
    GFX_GOL_PROGRESSBAR_DRAW_BAR_STATE = 0x2000,
    GFX_GOL_PROGRESSBAR_DRAW_STATE = 0x4000,
    GFX_GOL_PROGRESSBAR_HIDE_STATE = 0x8000
} GFX_GOL_PROGRESSBAR_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_PROGRESSBAR_DISABLED_STATE = 0x0002	Bit for disabled state.

GFX_GOL_PROGRESSBAR_VERTICAL_STATE = 0x0004	Bit for orientation (0- horizontal, 1 - vertical).
GFX_GOL_PROGRESSBAR_NOPROGRESS_STATE = 0x0008	Bit suppress rendering of progress in text.
GFX_GOL_PROGRESSBAR_DRAW_BAR_STATE = 0x2000	Bit to indicate object must be redrawn.
GFX_GOL_PROGRESSBAR_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_PROGRESSBAR_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.48 GFX_GOL_RADIOBUTTON Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    GFX_GOL_OBJ_HEADER * pHead;
    GFX_GOL_OBJ_HEADER * pNext;
    uint16_t textHeight;
    GFX_XCHAR * pText;
    GFX_ALIGNMENT alignment;
} GFX_GOL_RADIOBUTTON;
```

Description

the structure contains data for the Radio Button

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER).
GFX_GOL_OBJ_HEADER * pHead;	Pointer to the first Radio Button in the group
GFX_GOL_OBJ_HEADER * pNext;	Pointer to the next Radio Button in the group
uint16_t textHeight;	Pre-computed text height
GFX_XCHAR * pText;	Pointer to the text
GFX_ALIGNMENT alignment;	text alignment

5.2.2.7.26.49 GFX_GOL_RADIOBUTTON_STATE Enumeration

C

```
typedef enum {
    GFX_GOL_RADIOBUTTON_FOCUSED_STATE = 0x0001,
    GFX_GOL_RADIOBUTTON_DISABLED_STATE = 0x0002,
    GFX_GOL_RADIOBUTTON_CHECKED_STATE = 0x0004,
    GFX_GOL_RADIOBUTTON_GROUP_STATE = 0x0008,
    GFX_GOL_RADIOBUTTON_DRAW_CHECK_STATE = 0x1000,
    GFX_GOL_RADIOBUTTON_DRAW_FOCUS_STATE = 0x2000,
    GFX_GOL_RADIOBUTTON_DRAW_STATE = 0x4000,
    GFX_GOL_RADIOBUTTON_HIDE_STATE = 0x8000
} GFX_GOL_RADIOBUTTON_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_RADIOBUTTON_FOCUSED_STATE = 0x0001	Bit for focus state.

GFX_GOL_RADIOBUTTON_DISABLED_STATE = 0x0002	Bit for disabled state.
GFX_GOL_RADIOBUTTON_CHECKED_STATE = 0x0004	Bit to indicate Radio Button is checked.
GFX_GOL_RADIOBUTTON_GROUP_STATE = 0x0008	Bit to indicate the first Radio Button in the group. Each group MUST have this bit set for its first member even for a single member group. This means that any independent or stand alone Radio Button, the RB_GROUP bit must be always set. or alone in the group.
GFX_GOL_RADIOBUTTON_DRAW_CHECK_STATE = 0x1000	Bit to indicate check mark should be redrawn.
GFX_GOL_RADIOBUTTON_DRAW_FOCUS_STATE = 0x2000	Bit to indicate focus must be redrawn.
GFX_GOL_RADIOBUTTON_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_RADIOBUTTON_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.50 GFX_GOL_ROUNDIAL Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint16_t xCenter;
    uint16_t yCenter;
    uint16_t radius;
    uint16_t value;
    uint16_t max;
    uint16_t res;
    uint16_t curr_xPos;
    uint16_t curr_yPos;
    uint16_t new_xPos;
    uint16_t new_yPos;
    short vAngle;
} GFX_GOL_ROUNDIAL;
```

Description

The curr_xPos, curr_yPos, new_xPos and new_yPos parameters are internally generated to aid in the redrawing of the dial. User must avoid modifying these values.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
uint16_t xCenter;	x coordinate center position.
uint16_t yCenter;	y coordinate center position.
uint16_t radius;	Radius of the dial.
uint16_t value;	Initial value of the dial.
uint16_t max;	Maximum value of variable value (maximum = 65535).
uint16_t res;	Resolution of movement.
uint16_t curr_xPos;	Current x position.
uint16_t curr_yPos;	Current y position.
uint16_t new_xPos;	New x position.
uint16_t new_yPos;	New y position.

5.2.2.7.26.51 GFX_GOL_SCROLLBAR Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    int16_t currPos;
    uint16_t prevPos;
    uint16_t range;
    int16_t pos;
    uint16_t page;
    uint16_t thWidth;
    uint16_t thHeight;
} GFX_GOL_SCROLLBAR;
```

Description

Defines the parameters required for a slider/scrollbar Object. Depending on the GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE state bit slider or scrollbar mode is set. If GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE is set, mode is slider; if not set mode is scroll bar. For scrollbar mode, focus rectangle is not drawn.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
int16_t currPos;	Position of the slider relative to minimum.
uint16_t prevPos;	Previous position of the slider relative to minimum.
uint16_t range;	User defined range of the slider. Minimum value at 0 and maximum at 0x7FFF.
int16_t pos;	Position of the slider in range domain.
uint16_t page;	User specified resolution to incrementally change the position in range domain.
uint16_t thWidth;	Thumb width. This is computed internally. User must not change this value.
uint16_t thHeight;	Thumb width. This is computed internally. User must not change this value.

5.2.2.7.26.52 GFX_GOL_SCROLLBAR_STATE Enumeration

C

```
typedef enum {
    GFX_GOL_SCROLLBAR_FOCUSED_STATE = 0x0001,
    GFX_GOL_SCROLLBAR_DISABLED_STATE = 0x0002,
    GFX_GOL_SCROLLBAR_VERTICAL_STATE = 0x0004,
    GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE = 0x0010,
    GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE = 0x1000,
    GFX_GOL_SCROLLBAR_DRAW_FOCUS_STATE = 0x2000,
    GFX_GOL_SCROLLBAR_DRAW_STATE = 0x4000,
    GFX_GOL_SCROLLBAR_HIDE_STATE = 0x8000
} GFX_GOL_SCROLLBAR_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_SCROLLBAR_FOCUSED_STATE = 0x0001	Bit for focus state.
GFX_GOL_SCROLLBAR_DISABLED_STATE = 0x0002	Bit for disabled state.

GFX_GOL_SCROLLBAR_VERTICAL_STATE = 0x0004	Bit to indicate the scroll bar is drawn with vertical orientation
GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE = 0x0010	Bit to indicate the scroll bar is in slider mode.
GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE = 0x1000	Bit to indicate that only the thumb area will be redrawn
GFX_GOL_SCROLLBAR_DRAW_FOCUS_STATE = 0x2000	Bit to indicate focus must be redrawn.
GFX_GOL_SCROLLBAR_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_SCROLLBAR_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.53 GFX_GOL_STATICTEXT Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    GFX_XCHAR * pText;
    GFX_ALIGNMENT alignment;
} GFX_GOL_STATICTEXT;
```

Description

Defines the parameters required for a Static Text Object.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER).
GFX_XCHAR * pText;	The pointer to text used.
GFX_ALIGNMENT alignment;	text alignment

5.2.2.7.26.54 GFX_GOL_STATICTEXT_STATE Enumeration

C

```
typedef enum {
    GFX_GOL_STATICTEXT_DISABLED_STATE = 0x0002,
    GFX_GOL_STATICTEXT_FRAME_STATE = 0x0010,
    GFX_GOL_STATICTEXT_NOBACKGROUND_STATE = 0x0020,
    GFX_GOL_STATICTEXT_DRAW_UPDATE = 0x2000,
    GFX_GOL_STATICTEXT_DRAW_STATE = 0x4000,
    GFX_GOL_STATICTEXT_HIDE_STATE = 0x8000
} GFX_GOL_STATICTEXT_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_STATICTEXT_DISABLED_STATE = 0x0002	Bit for disabled state.
GFX_GOL_STATICTEXT_FRAME_STATE = 0x0010	Bit to indicate frame is enabled.
GFX_GOL_STATICTEXT_NOBACKGROUND_STATE = 0x0020	Bit to indicate background is enabled.
GFX_GOL_STATICTEXT_DRAW_UPDATE = 0x2000	Bit to indicate object must be redrawn.

GFX_GOL_STATICTEXT_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_STATICTEXT_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.55 GFX_GOL_TEXTENTRY Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint16_t horizontalKeys;
    uint16_t verticalKeys;
    GFX_XCHAR * pTeOutput;
    GFX_ALIGNMENT alignment;
    uint16_t CurrentLength;
    uint16_t outputLenMax;
    KEYMEMBER * pActiveKey;
    KEYMEMBER * pHeadOfList;
    GFX_RESOURCE_HDR * pDisplayFont;
} GFX_GOL_TEXTENTRY;
```

Description

Defines the parameters required for a TextEntry Object.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all objects (see GFX_GOL_OBJ_HEADER).
uint16_t horizontalKeys;	Number of horizontal keys
uint16_t verticalKeys;	Number of vertical keys
GFX_XCHAR * pTeOutput;	Pointer to the buffer assigned by the user which holds the text shown in the editbox.
GFX_ALIGNMENT alignment;	text alignment
uint16_t CurrentLength;	Current length of the string in the buffer. The maximum value of this is equal to outputLenMax.
uint16_t outputLenMax;	Maximum expected length of output buffer pTeOutput
KEYMEMBER * pActiveKey;	Pointer to the active key KEYMEMBER . This is only used by the Widget. User must not change the value of this parameter directly.
KEYMEMBER * pHeadOfList;	Pointer to head of the list
GFX_RESOURCE_HDR * pDisplayFont;	Pointer to the font used in displaying the text.

5.2.2.7.26.56 GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE Enumeration

C

```
typedef enum {
    GFX_GOL_TEXTENTRY_NONE_COM = 0x00,
    GFX_GOL_TEXTENTRY_DELETE_COM = 0x01,
    GFX_GOL_TEXTENTRY_SPACE_COM = 0x02,
    GFX_GOL_TEXTENTRY_ENTER_COM = 0x03
} GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE;
```

Description

Optional COMMANDS assigned to keys

Members

Members	Description
GFX_GOL_TEXTENTRY_NONE_COM = 0x00	This type is used when the key is not assigned to any command.
GFX_GOL_TEXTENTRY_DELETE_COM = 0x01	This type is used to assign a "delete" command on a key.
GFX_GOL_TEXTENTRY_SPACE_COM = 0x02	This type is used to assign an insert "space" command on a key.
GFX_GOL_TEXTENTRY_ENTER_COM = 0x03	This type is used to assign an "enter" (carriage return) command on a key.

5.2.2.7.26.57 GFX_GOL_TEXTENTRY_STATE Enumeration**C**

```
typedef enum {
    GFX_GOL_TEXTENTRY_DISABLED_STATE = 0x0002,
    GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE = 0x0004,
    GFX_GOL_TEXTENTRY_ECHO_HIDE_STATE = 0x0008,
    GFX_GOL_TEXTENTRY_DRAW_STATE = 0x4000,
    GFX_GOL_TEXTENTRY_HIDE_STATE = 0x8000,
    GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE = 0x2000,
    GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE = 0x1000
} GFX_GOL_TEXTENTRY_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_TEXTENTRY_DISABLED_STATE = 0x0002	Bit for disabled state.
GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE = 0x0004	Bit for press state.
GFX_GOL_TEXTENTRY_ECHO_HIDE_STATE = 0x0008	Bit to hide the entered characters and instead echo "*" characters to the display.
GFX_GOL_TEXTENTRY_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_TEXTENTRY_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE = 0x2000	Bit to indicate redraw of a key is needed
GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE = 0x1000	Bit to indicate redraw of the text displayed is needed.

5.2.2.7.26.58 GFX_GOL_TRANSLATED_ACTION Enumeration**C**

```
typedef enum {
    GFX_GOL_OBJECT_ACTION_INVALID = 0x3500,
    GFX_GOL_OBJECT_ACTION_PASSIVE,
    GFX_GOL_ANALOGCLOCK_ACTION_PRESSED,
    GFX_GOL_ANALOGCLOCK_ACTION_RELEASED,
    GFX_GOL_BUTTON_ACTION_PRESSED,
    GFX_GOL_BUTTON_ACTION_STILLPRESSED,
    GFX_GOL_BUTTON_ACTION_RELEASED,
    GFX_GOL_BUTTON_ACTION_CANCELPRESS,
    GFX_GOL_CHART_ACTION_SELECTED,
```

```

GFX_GOL_CHECKBOX_ACTION_CHECKED,
GFX_GOL_CHECKBOX_ACTION_UNCHECKED,
GFX_GOL_CUSTOMCONTROL_ACTION_SELECTED,
GFX_GOL_DIGITALMETER_ACTION_SELECTED,
GFX_GOL_EDITBOX_ACTION_ADD_CHAR,
GFX_GOL_EDITBOX_ACTION_DEL_CHAR,
GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN,
GFX_GOL_GRID_ACTION_TOUCHED,
GFX_GOL_GRID_ACTION_ITEM_SELECTED,
GFX_GOL_GRID_ACTION_UP,
GFX_GOL_GRID_ACTION_DOWN,
GFX_GOL_GRID_ACTION_LEFT,
GFX_GOL_GRID_ACTION_RIGHT,
GFX_GOL_GROUPBOX_ACTION_SELECTED,
GFX_GOL_LISTBOX_ACTION_SELECTED,
GFX_GOL_LISTBOX_ACTION_MOVE,
GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN,
GFX_GOL_METER_ACTION_SET,
GFX_GOL_PICTURECONTROL_ACTION_SELECTED,
GFX_GOL_PROGRESSBAR_ACTION_SELECTED,
GFX_GOL_RADIOBUTTON_ACTION_CHECKED,
GFX_GOL_SCROLLBAR_ACTION_INC,
GFX_GOL_SCROLLBAR_ACTION_DEC,
GFX_GOL_STATICTEXT_ACTION_SELECTED,
GFX_GOL_TEXTENTRY_ACTION_RELEASED,
GFX_GOL_TEXTENTRY_ACTION_PRESSED,
GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR,
GFX_GOL_TEXTENTRY_ACTION_DELETE,
GFX_GOL_TEXTENTRY_ACTION_SPACE,
GFX_GOL_TEXTENTRY_ACTION_ENTER,
GFX_GOL_WINDOW_ACTION_CLIENT,
GFX_GOL_WINDOW_ACTION_TITLE
} GFX_GOL_TRANSLATED_ACTION;

```

Description

GFX_GOL_TRANSLATED_ACTION

This enumeration specifies the different object actions supported in the library.

Members

Members	Description
GFX_GOL_OBJECT_ACTION_INVALID = 0x3500	Invalid message response.
GFX_GOL_OBJECT_ACTION_PASSIVE	Passive message response. No change in object needed.
GFX_GOL_ANALOGCLOCK_ACTION_PRESSED	Analog Clock Pressed Action
GFX_GOL_ANALOGCLOCK_ACTION_RELEASED	Analog Clock Released Action
GFX_GOL_BUTTON_ACTION_PRESSED	Button pressed action ID.
GFX_GOL_BUTTON_ACTION_STILLPRESSED	Button is continuously pressed ID.
GFX_GOL_BUTTON_ACTION_RELEASED	Button released action ID.
GFX_GOL_BUTTON_ACTION_CANCELPRESS	Button released action ID with button press canceled.
GFX_GOL_CHART_ACTION_SELECTED	Chart selected action ID
GFX_GOL_CHECKBOX_ACTION_CHECKED	Check Box check action ID.
GFX_GOL_CHECKBOX_ACTION_UNCHECKED	Check Box un-check action ID.
GFX_GOL_CUSTOMCONTROL_ACTION_SELECTED	Custom Control selected action ID.
GFX_GOL_DIGITALMETER_ACTION_SELECTED	Digital Meter selected action ID.
GFX_GOL_EDITBOX_ACTION_ADD_CHAR	Edit Box insert character action ID.
GFX_GOL_EDITBOX_ACTION_DEL_CHAR	Edit Box remove character action ID.
GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN	Edit Box touchscreen selected action ID.

GFX_GOL_GRID_ACTION_TOUCHED	Grid item touched action ID.
GFX_GOL_GRID_ACTION_ITEM_SELECTED	Grid item selected action ID.
GFX_GOL_GRID_ACTION_UP	Grid up action ID.
GFX_GOL_GRID_ACTION_DOWN	Grid down action ID.
GFX_GOL_GRID_ACTION_LEFT	Grid left action ID.
GFX_GOL_GRID_ACTION_RIGHT	Grid right action ID.
GFX_GOL_GROUPBOX_ACTION_SELECTED	Group Box selected action ID.
GFX_GOL_LISTBOX_ACTION_SELECTED	List Box item select action ID.
GFX_GOL_LISTBOX_ACTION_MOVE	List Box item move action ID.
GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN	List Box touchscreen selected action ID.
GFX_GOL_METER_ACTION_SET	Meter set value action ID.
GFX_GOL_PICTURECONTROL_ACTION_SELECTED	Picture selected action ID.
GFX_GOL_PROGRESSBAR_ACTION_SELECTED	Progress Bar selected action ID.
GFX_GOL_RADIOBUTTON_ACTION_CHECKED	Radio Button check action ID.
GFX_GOL_SCROLLBAR_ACTION_INC	Slider or Scroll Bar increment action ID.
GFX_GOL_SCROLLBAR_ACTION_DEC	Slider or Scroll Bar decrement action ID.
GFX_GOL_STATICTEXT_ACTION_SELECTED	Static Text selected action ID.
GFX_GOL_TEXTENTRY_ACTION_RELEASED	TextEntry released action ID
GFX_GOL_TEXTENTRY_ACTION_PRESSED	TextEntry pressed action ID
GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR	TextEntry add character action ID
GFX_GOL_TEXTENTRY_ACTION_DELETE	TextEntry delete character action ID
GFX_GOL_TEXTENTRY_ACTION_SPACE	TextEntry add space character action ID
GFX_GOL_TEXTENTRY_ACTION_ENTER	TextEntry enter action ID
GFX_GOL_WINDOW_ACTION_CLIENT	Window client area selected action ID.
GFX_GOL_WINDOW_ACTION_TITLE	Window title bar selected action ID.

Remarks

None.

5.2.2.7.26.59 GFX_GOL_TRANSLATED_ACTION, \ GFX_GOL_OBJ_HEADER *, \ GFX_GOL_MESSAGE * \) Type

C

```
typedef (bool)GFX_GOL_MESSAGE_CALLBACK_FUNC GFX_GOL_TRANSLATED_ACTION, \ GFX_GOL_OBJ_HEADER *,
\ GFX_GOL_MESSAGE * \ )();
```

Description

This application defined function is called by the [GFX_GOL_ObjectMessage\(\)](#) function allowing the application the opportunity to process the user messages and customize object behavior as well as application controlled functions.

[GFX_GOL_ObjectMessage\(\)](#) calls this function when a valid message for an object in the active list is received. Application implements any action for the message in this callback function. If this callback function returns true, the message for the object will be processed using the default action of the object. If false is returned, the default action will not be performed. In this case, it is assumed that this callback function has performed the appropriate changes to the states of the objects.

Preconditions

None.

Parameters

Parameters	Description
GFX_GOL_TRANSLATED_ACTION	Translated message for the object
GFX_GOL_OBJ_HEADER *	Pointer to the object that processed the message.
GFX_GOL_MESSAGE *	Pointer to the message from user.

Returns

true - When true is returned, the object will set its state depending on the translated messages. false - When false is returned, the object will not process the translated message and will assume the application has performed necessary action on the message.

Example

None.

5.2.2.7.26.60 GFX_GOL_WINDOW Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    uint16_t textHeight;
    GFX_XCHAR * pText;
    GFX_ALIGNMENT alignment;
    GFX_RESOURCE_HDR * pBitmap;
} GFX_GOL_WINDOW;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t textHeight;	Pre-computed text height
GFX_XCHAR * pText;	Pointer to the title text
GFX_ALIGNMENT alignment;	text alignment
GFX_RESOURCE_HDR * pBitmap;	Pointer to the bitmap for the title bar

5.2.2.7.26.61 GFX_GOL_WINDOW_STATE Enumeration

C

```
typedef enum {
    GFX_GOL_WINDOW_FOCUSED_STATE = 0x0001,
    GFX_GOL_WINDOW_DISABLED_STATE = 0x0002,
    GFX_GOL_WINDOW_TITLECENTER_STATE = 0x0004,
    GFX_GOL_WINDOW_HIDE_STATE = 0x8000,
    GFX_GOL_WINDOW_DRAW_CLIENT_STATE = 0x4000,
    GFX_GOL_WINDOW_DRAW_TITLE_STATE = 0x2000,
    GFX_GOL_WINDOW_DRAW_STATE = 0x6000
} GFX_GOL_WINDOW_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_WINDOW_FOCUSED_STATE = 0x0001	Bit for focus state

GFX_GOL_WINDOW_DISABLED_STATE = 0x0002	Bit for disabled state
GFX_GOL_WINDOW_TITLECENTER_STATE = 0x0004	Bit to center the text on the Title Area
GFX_GOL_WINDOW_HIDE_STATE = 0x8000	Bit to indicate window must be removed from screen
GFX_GOL_WINDOW_DRAW_CLIENT_STATE = 0x4000	Bit to indicate client area must be redrawn
GFX_GOL_WINDOW_DRAW_TITLE_STATE = 0x2000	Bit to indicate title area must be redrawn
GFX_GOL_WINDOW_DRAW_STATE = 0x6000	Bits to indicate whole window must be redrawn

5.2.2.7.26.62 GFX_GRADIENT_STYLE Structure

C

```
typedef struct {
    GFX_FILL_STYLE type;
    GFX_COLOR startColor;
    GFX_COLOR endColor;
    uint8_t length;
} GFX_GRADIENT_STYLE;
```

Description

This structure is used to describe the gradient style.

Members

Members	Description
GFX_FILL_STYLE type;	selected the gradient type
GFX_COLOR startColor;	sets the starting color of gradient transition
GFX_COLOR endColor;	sets the ending color of gradient transition
uint8_t length;	defines the length of the gradient transition in pixels

5.2.2.7.26.63 GFX_LINE_STYLE Enumeration

C

```
typedef enum {
    GFX_LINE_TYPE_MASK = 0x0007,
    GFX_LINE_THICKNESS_MASK = 0x0010,
    GFX_LINE_STYLE_THIN_SOLID = 0x0A00,
    GFX_LINE_STYLE_THIN_DOTTED = 0x0A01,
    GFX_LINE_STYLE_THIN_DASHED = 0x0A04,
    GFX_LINE_STYLE_THICK_SOLID = 0x0A10,
    GFX_LINE_STYLE_THICK_DOTTED = 0x0A11,
    GFX_LINE_STYLE_THICK_DASHED = 0x0A14,
    GFX_LINE_STYLE_ANTI_ALIASED = 0x0A20
} GFX_LINE_STYLE;
```

Description

Defines the types of line styles.

Members

Members	Description
GFX_LINE_TYPE_MASK = 0x0007	DO NOT document, mask for line type
GFX_LINE_THICKNESS_MASK = 0x0010	DO NOT document, mask for thickness

GFX_LINE_STYLE_THIN_SOLID = 0x0A00	solid line, 1 pixel wide (default)
GFX_LINE_STYLE_THIN_DOTTED = 0x0A01	dotted line, 1 pixel wide
GFX_LINE_STYLE_THIN_DASHED = 0x0A04	dashed line, , 1 pixel wide
GFX_LINE_STYLE_THICK_SOLID = 0x0A10	solid line, 3 pixel wide
GFX_LINE_STYLE_THICK_DOTTED = 0x0A11	dotted line, 3 pixel wide
GFX_LINE_STYLE_THICK_DASHED = 0x0A14	dashed line, 3 pixel wide
GFX_LINE_STYLE_ANTI_ALIASED = 0x0A20	anti-aliased line (future feature)

5.2.2.7.26.64 GFX_MCHP_BITMAP_HEADER Structure

C

```
typedef struct {
    uint8_t bitmapType;
    uint8_t colorDepth;
    int16_t height;
    int16_t width;
} GFX_MCHP_BITMAP_HEADER;
```

Description

Structure describing the bitmap header.

Members

Members	Description
uint8_t bitmapType;	type of image, information on how to render the image 0 - no compression, palette is present for color depth = 8, 4 and 1 BPP 1 - palette is provided as a separate object (see PALETTE_HEADER) for color depth = 8, 4, and 1 BPP, ID to the palette is embedded in the bitmap.
uint8_t colorDepth;	Color depth used
int16_t height;	Image height
int16_t width;	Image width

5.2.2.7.26.65 GFX_PALETTE_ENTRY Union

C

```
typedef union {
    uint16_t value;
    struct {
        uint16_t r : 5;
        uint16_t g : 6;
        uint16_t b : 5;
    } color;
    struct {
        uint16_t reserved : 12;
        uint16_t luma : 4;
    } monochrome;
} GFX_PALETTE_ENTRY;
```

Description

Structure used for the palette entries.

- For TFT: color is defined as 5-6-5 RGB format (5-bits for **RED**, 6-bits for **GREEN** and 5-bits for **BLUE**).
- For Monochrome: 4 bits are used to represent the luma.

Members

Members	Description
uint16_t value;	a 16-bit value representing a color or palette entry
struct { uint16_t r : 5; uint16_t g : 6; uint16_t b : 5; } color;	color value in 5-6-5 RGB format
uint16_t r : 5;	represents the RED component
uint16_t g : 6;	represents the GREEN component
uint16_t b : 5;	represents the BLUE component
struct { uint16_t reserved : 12; uint16_t luma : 4; } monochrome;	monochrome LUMA value
uint16_t reserved : 12;	reserved, used as a filler
uint16_t luma : 4;	monochrome LUMA value

5.2.2.7.26.66 GFX_PARTIAL_IMAGE_PARAM Structure**C**

```
typedef struct {
    uint16_t width;
    uint16_t height;
    uint16_t xoffset;
    uint16_t yoffset;
} GFX_PARTIAL_IMAGE_PARAM;
```

Description

Structure describing the partial image area to render.

Members

Members	Description
uint16_t width;	Partial Image width
uint16_t height;	Partial Image height
uint16_t xoffset;	xoffset of the partial image (with respect to the image horizontal origin)
uint16_t yoffset;	yoffset of the partial image (with respect to the image vertical origin)

5.2.2.7.26.67 GFX_Primitive_DATA Structure**C**

```
typedef struct {
    volatile uint8_t activePage;
    volatile uint8_t visualPage;
    uint8_t transitionPage;
    uint8_t foregroundPage;
    uint8_t backgroundPage;
    uint8_t destinationPage;
    GFX_LINE_STYLE lineType;
    uint16_t cursorX;
    uint16_t cursorY;
```

```
GFX_BEVEL_RENDER_TYPE bevelDrawType;
GFX_COLOR currentColor;
GFX_FONT_CURRENT currentFont;
GFX_RESOURCE_HDR * pResource;
GFX_ALPHA_PARAMS alphaArea;
GFX_TRANSITION_PARAMS transitionParams;
GFX_COLOR line[480];
GFX_COLOR alphaLine[480];
uint8_t InvalidatedRectangleCount;
GFX_GRADIENT_STYLE gradientStyle;
} GFX_Primitive_DATA;
```

Description

GFX primitive data

Members

Members	Description
uint8_t foregroundPage;	Initialize Alpha Values
GFX_COLOR currentColor;	Current Color for the instance
GFX_COLOR line[480];	Storage for line data
GFX_COLOR alphaLine[480];	Storage for alpha line data
uint8_t InvalidatedRectangleCount;	Double Buffer Variable

Remarks

None.

5.2.2.7.26.68 GFX_RECTANGULAR_AREA Structure

C

```
typedef struct {
    uint16_t left;
    uint16_t top;
    uint16_t right;
    uint16_t bottom;
} GFX_RECTANGULAR_AREA;
```

Description

Structure describing a rectangular area in the frame buffer.

5.2.2.7.26.69 GFX_RESOURCE Enumeration

C

```
typedef enum {
    GFX_RESOURCE_MEMORY_FLASH = 0x0000,
    GFX_RESOURCE_MEMORY_EXTERNAL = 0x0001,
    GFX_RESOURCE_MEMORY_RAM = 0x0004,
    GFX_RESOURCE_MEMORY_EDS_EPMP = 0x0005,
    GFX_RESOURCE_TYPE_MCHP_MBITMAP = 0x0000,
    GFX_RESOURCE_TYPE_JPEG = 0x0100,
    GFX_RESOURCE_TYPE_BINARY = 0x0200,
    GFX_RESOURCE_TYPE_FONT = 0x0300,
    GFX_RESOURCE_TYPE_PALETTE = 0x0400,
    GFX_RESOURCE_COMP_NONE = 0x0000,
    GFX_RESOURCE_COMP_RLE = 0x1000,
    GFX_RESOURCE_COMP_IPU = 0x2000,
    GFX_RESOURCE_MCHP_MBITMAP_FLASH_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP |
    GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_MCHP_MBITMAP_FLASH_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP |
```

```

GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_RLE),
    GFX_RESOURCE_MCHP_MBITMAP_FLASH_IPU = (GFX_RESOURCE_TYPE_MCHP_MBITMAP |
GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_IPU),
    GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP |
GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP |
GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_RLE),
    GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_IPU = (GFX_RESOURCE_TYPE_MCHP_MBITMAP |
GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_IPU),
    GFX_RESOURCE_MCHP_MBITMAP_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP |
GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_MCHP_MBITMAP_EDS_EPMP_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP |
GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_RLE),
    GFX_RESOURCE_MCHP_MBITMAP_EDS_EPMP_IPU = (GFX_RESOURCE_TYPE_MCHP_MBITMAP |
GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_IPU),
    GFX_RESOURCE_JPEG_FLASH_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_JPEG_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_JPEG_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_BINARY_FLASH_NONE = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_BINARY_FLASH_IPU = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_IPU),
    GFX_RESOURCE_BINARY_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EXTERNAL
| GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_BINARY_EXTERNAL_IPU = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EXTERNAL
| GFX_RESOURCE_COMP_IPU),
    GFX_RESOURCE_BINARY_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EDS_EPMP
| GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_BINARY_EDS_EPMP_IPU = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EDS_EPMP
| GFX_RESOURCE_COMP_IPU),
    GFX_RESOURCE_FONT_FLASH_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_FONT_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_FONT_RAM_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_RAM |
GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_FONT_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_PALETTE_FLASH_NONE = (GFX_RESOURCE_TYPE_PALETTE | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_PALETTE_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_PALETTE |
GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_NONE),
    GFX_RESOURCE_PALETTE_RAM_NONE = (GFX_RESOURCE_TYPE_PALETTE | GFX_RESOURCE_MEMORY_RAM |
GFX_RESOURCE_COMP_NONE)
} GFX_RESOURCE;

```

Description

Memory type enumeration to determine the source of data. Used in interpreting images, fonts and palette from different memory sources.

Members

Members	Description
GFX_RESOURCE_MEMORY_FLASH = 0x0000	internal flash
GFX_RESOURCE_MEMORY_EXTERNAL = 0x0001	external memory
GFX_RESOURCE_MEMORY_RAM = 0x0004	RAM

GFX_RESOURCE_MEMORY_EDS_EPMP = 0x0005	memory in EPMP, base addresses are set in the hardware profile
GFX_RESOURCE_TYPE_MCHP_MBITMAP = 0x0000	data resource is type Microchip bitmap
GFX_RESOURCE_TYPE_JPEG = 0x0100	data resource is type JPEG
GFX_RESOURCE_TYPE_BINARY = 0x0200	data resource is type binary
GFX_RESOURCE_TYPE_FONT = 0x0300	data resource is type font
GFX_RESOURCE_TYPE_PALETTE = 0x0400	data resource is type palette
GFX_RESOURCE_COMP_NONE = 0x0000	data resource has no compression
GFX_RESOURCE_COMP_RLE = 0x1000	data resource is compressed with RLE
GFX_RESOURCE_COMP_IPU = 0x2000	data resource compressed with DEFLATE (for IPU)
GFX_RESOURCE_MCHP_MBITMAP_FLASH_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP GFX_RESOURCE_MEMORY_FLASH GFX_RESOURCE_COMP_NONE)	These are common resource combinations used by the graphics library Bitmaps
GFX_RESOURCE_JPEG_FLASH_NONE = (GFX_RESOURCE_TYPE_JPEG GFX_RESOURCE_MEMORY_FLASH GFX_RESOURCE_COMP_NONE)	JPEGS
GFX_RESOURCE_BINARY_FLASH_NONE = (GFX_RESOURCE_TYPE_BINARY GFX_RESOURCE_MEMORY_FLASH GFX_RESOURCE_COMP_NONE)	Binary
GFX_RESOURCE_FONT_FLASH_NONE = (GFX_RESOURCE_TYPE_FONT GFX_RESOURCE_MEMORY_FLASH GFX_RESOURCE_COMP_NONE)	Fonts
GFX_RESOURCE_PALETTE_FLASH_NONE = (GFX_RESOURCE_TYPE_PALETTE GFX_RESOURCE_MEMORY_FLASH GFX_RESOURCE_COMP_NONE)	Palettes

5.2.2.7.26.70 GFX_RESOURCE_BINARY Structure

C

```
typedef struct {
    union {
        uint32_t extAddress;
        uint8_gfx_image_prog * progByteAddress;
        uint16_gfx_image_prog * progWordAddress;
        const char * constAddress;
        char * ramAddress;
        __eds__ char * edsAddress;
    } location;
    uint32_t size;
    uint32_t param1;
    uint32_t param2;
} GFX_RESOURCE_BINARY;
```

Description

Structure for Graphics Resource - binary data.

Members

Members	Description
uint32_t extAddress;	generic address
uint8_gfx_image_prog * progByteAddress;	for addresses in program section
uint16_gfx_image_prog * progWordAddress;	for addresses in program section
const char * constAddress;	for addresses in FLASH
char * ramAddress;	for addresses in RAM
__eds__ char * edsAddress;	for addresses in EDS
uint32_t param1;	Parameters used for the GFX_RESOURCE . Depending on the GFX_RESOURCE type definition of param1 can change. For IPU and RLE compressed images, param1 indicates the compressed size of the image.
uint32_t param2;	Parameters used for the GFX_RESOURCE . Depending on the GFX_RESOURCE type

5.2.2.7.26.71 GFX_RESOURCE_FONT Structure**C**

```
typedef struct {
    union {
        uint32_t extAddress;
        GFX_FONT_SPACE char * progByteAddress;
        char * ramAddress;
        __eds__ char * edsAddress;
    } location;
    GFX_FONT_HEADER header;
} GFX_RESOURCE_FONT;
```

Description

Structure for Graphics Resource - Fonts.

Members

Members	Description
__eds__ char * edsAddress;	for addresses in EDS

5.2.2.7.26.72 GFX_RESOURCE_HDR Structure**C**

```
typedef struct {
    GFX_RESOURCE type;
    uint16_t ID;
    union {
        GFX_RESOURCE_IMAGE image;
        GFX_RESOURCE_FONT font;
        GFX_RESOURCE_BINARY binary;
        GFX_RESOURCE_PALETTE palette;
    } resource;
} GFX_RESOURCE_HDR;
```

Description

Structure for Graphics Resource Header. This is the common header for all Graphics Resources.

Members

Members	Description
GFX_RESOURCE type;	Graphics resource type, determines the type and location of data
uint16_t ID;	memory ID, user defined value to differentiate between graphics resources of the same type When using EDS_EPMP the following ID values are reserved and used by the Microchip display driver 0 - reserved (do not use) 1 - reserved for base address of EPMP CS1 2 - reserved for base address of EPMP CS2

5.2.2.7.26.73 GFX_RESOURCE_IMAGE Structure**C**

```
typedef struct {
    union {
        uint32_t extAddress;
        uint8_gfx_image_prog * progByteAddress;
        uint16_gfx_image_prog * progWordAddress;
        const char * constAddress;
        char * ramAddress;
        __eds__ char * edsAddress;
    } location;
    uint16_t width;
    uint16_t height;
    union {
        uint32_t compressedSize;
        uint32_t reserved;
    } parameter1;
    union {
        uint32_t rawSize;
        uint32_t reserved;
    } parameter2;
    uint8_t colorDepth;
    uint8_t type;
    uint16_t paletteID;
} GFX_RESOURCE_IMAGE;
```

Description

Structure for Graphics Resource - image.

Members

Members	Description
uint32_t extAddress;	generic address
uint8_gfx_image_prog * progByteAddress;	for addresses in program section
uint16_gfx_image_prog * progWordAddress;	for addresses in program section
const char * constAddress;	for addresses in FLASH
char * ramAddress;	for addresses in RAM
__eds__ char * edsAddress;	for addresses in EDS
uint16_t width;	width of the image
uint16_t height;	height of the image
uint32_t compressedSize;	Parameters used for the GFX_RESOURCE . Depending on the GFX_RESOURCE type definition of param1 can change. For IPU and RLE compressed images, param1 indicates the compressed size of the image.
uint32_t rawSize;	Parameters used for the GFX_RESOURCE . Depending on the GFX_RESOURCE type definition of param2 can change. For IPU and RLE compressed images, param2 indicates the uncompressed size of the image.

uint8_t colorDepth;	color depth of the image
uint8_t type;	type of image, information on how to render the image 0x00 - no compression, palette is present for color depth = 8, 4 and 1 BPP 0x10 - palette is provided as a separate object (see PALETTE_HEADER) for color depth = 8, 4, and 1 BPP, ID to the palette is embedded color depth = 8, 4, and 1 BPP, in the bitmap. 0xYY - reserved
uint16_t paletteID;	if type == MCHP_BITMAP_PALETTE_STR (0x10), this represents the unique ID of the palette being used

5.2.2.7.26.74 GFX_RESOURCE_PALETTE Structure

C

```
typedef struct {
    union {
        uint32_t extAddress;
        const GFX_PALETTE_ENTRY * constByteAddress;
        GFX_PALETTE_ENTRY * ramAddress;
    } location;
    uint16_t numberOfEntries;
    uint16_t paletteID;
} GFX_RESOURCE_PALETTE;
```

Description

Structure for Graphics Resource - Palette.

Members

Members	Description
uint16_t paletteID;	Unique ID of the palette that will match the ID in the GFX_RESOURCE_HDR if the image has palette removed and supplied as a separate object.

5.2.2.7.26.75 GFX_STATUS Enumeration

C

```
typedef enum {
    GFX_STATUS_FAILURE = 0,
    GFX_STATUS_SUCCESS = 1
} GFX_STATUS;
```

Description

Status types for rendering.

5.2.2.7.26.76 GFX_STATUS_BIT Enumeration

C

```
typedef enum {
    GFX_STATUS_FAILURE_BIT = 0,
    GFX_STATUS_SUCCESS_BIT = 1,
    GFX_STATUS_ERROR_BIT = 2,
    GFX_STATUS_BUSY_BIT = 4,
    GFX_STATUS_READY_BIT = 8
} GFX_STATUS_BIT;
```

Description

Bit masks for additional status information.

5.2.2.7.26.77 GFX_TRANSITION_DIRECTION Enumeration

C

```
typedef enum {
    LEFT_TO_RIGHT = 0,
    RIGHT_TO_LEFT,
    TOP_TO_BOTTOM,
    BOTTOM_TO_TOP,
    HORIZONTAL,
    VERTICAL
} GFX_TRANSITION_DIRECTION;
```

Description

Direction enumeration to determine the direction of the selected [GFX_TRANSITION_TYPE](#).

Members

Members	Description
LEFT_TO_RIGHT = 0	option used in SLIDE, PUSH transition type (GFX_TRANSITION_TYPE)
RIGHT_TO_LEFT	option used in SLIDE, PUSH transition type (GFX_TRANSITION_TYPE)
TOP_TO_BOTTOM	option used in SLIDE, PUSH transition type (GFX_TRANSITION_TYPE)
BOTTOM_TO_TOP	option used in SLIDE, PUSH transition type (GFX_TRANSITION_TYPE)
HORIZONTAL	option used in EXPANDING_LINE and CONTRACTING_LINE transition type (GFX_TRANSITION_TYPE)
VERTICAL	option used in EXPANDING_LINE and CONTRACTING_LINE transition type (GFX_TRANSITION_TYPE)

5.2.2.7.26.78 GFX_TRANSITION_PARAMS Structure

C

```
typedef struct {
    uint16_t left;
    uint16_t top;
    uint16_t delay_ms;
    uint8_t sourcePage;
    uint8_t destinationPage;
    uint8_t type;
    uint8_t param1;
    uint8_t param2;
} GFX_TRANSITION_PARAMS;
```

Description

GFX transition parameters

Remarks

None.

5.2.2.7.26.79 GFX_TRANSITION_TYPE Enumeration

C

```
typedef enum {
    EXPANDING_RECTANGLE = 0,
    CONTRACTING_RECTANGLE,
    SLIDE,
    PUSH,
    EXPANDING_LINE,
    CONTRACTING_LINE
}
```

```
} GFX_TRANSITION_TYPE;
```

Description

Transition type enumeration to determine the type of the transition. Each type will require specific parameters when setting up the transition operation (GFXSetupTransition() or GFXTransition()).

GFX_TRANSITION_TYPE	param1	param2 (sets the direction of transition)
PLAIN	pixel block size	not used
EXPANDING_RECTANGLE	pixel block size	not used
CONTRACTING_RECTANGLE	pixel block size	not used
SLIDE	pixel block size	LEFT_TO_RIGHT, RIGHT_TO_LEFT, TOP_TO_BOTTOM, BOTTOM_TO_TOP
PUSH	pixel block size	LEFT_TO_RIGHT, RIGHT_TO_LEFT, TOP_TO_BOTTOM, BOTTOM_TO_TOP
EXPANDING_LINE	pixel block size	HORIZONTAL, VERTICAL
CONTRACTING_LINE	pixel block size	HORIZONTAL, VERTICAL

Members

Members	Description
EXPANDING_RECTANGLE = 0	param1 -> Pixel-block size
CONTRACTING_RECTANGLE	param1 -> Pixel-block size
SLIDE	param1 -> Pixel-block size, param2 -> Sliding direction LEFT_TO_RIGHT/RIGHT_TO_LEFT/TOP_TO_BOTTOM/BOTTOM_TO_TOP
PUSH	param1 -> Pixel-block size, param2 -> Sliding direction LEFT_TO_RIGHT/RIGHT_TO_LEFT/TOP_TO_BOTTOM/BOTTOM_TO_TOP
EXPANDING_LINE	param1 -> Pixel-block size, param2 -> direction HORIZONTAL/VERTICAL
CONTRACTING_LINE	param1 -> Pixel-block size, param2 -> direction HORIZONTAL/VERTICAL

5.2.2.7.26.80 GFX_TRIG_FUNCTION_TYPE Enumeration

C

```
typedef enum {
    GFX_TRIG_SINE_TYPE = 0,
    GFX_TRIG_COSINE_TYPE = 1
} GFX_TRIG_FUNCTION_TYPE;
```

Description

DOM-IGNORE-START

- Overview: Graphics Library helper function for trigonometric sine
- cosine angles are defined in a sine/cosine table.
- This type enumeration specifies if the value retrieved
- from that look up table is for a sine(angle) or
- cosine(angle) value.

5.2.2.7.26.81 GOL_PANEL_PARAM Structure

C

```
typedef struct {
    int16_t panelLeft;
    int16_t panelTop;
    int16_t panelRight;
    int16_t panelBottom;
    GFX_COLOR panelFaceColor;
    GFX_COLOR panelEmbossLtColor;
    GFX_COLOR panelEmbossDkColor;
    uint16_t panelEmbossSize;
    uint16_t panelRadius;
    GFX_RESOURCE_HDR * pPanelImage;
    GFX_FILL_STYLE panelFillStyle;
    GFX_COLOR panelGradientStartColor;
    GFX_COLOR panelGradientEndColor;
    uint16_t panelAlpha;
} GOL_PANEL_PARAM;
```

Description

GOL_PANEL_PARAM

This structure defines the panel parameters when rendering a panel.

Remarks

None.

5.2.2.7.26.82 GRID Structure

C

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    short numColumns;
    short numRows;
    short cellHeight;
    short cellWidth;
    short focusX;
    short focusY;
    GRIDITEM * gridObjects;
} GRID;
```

Description

Defines the parameters required for a grid Object. Clipping is not supported in grid object.

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
short numColumns;	Number of columns drawn for the Grid.
short numRows;	Number of rows drawn for the Grid.
short cellHeight;	The height of each cell in pixels.
short cellWidth;	The width of each cell in pixels.
short focusX;	The x position of the cell to be focused.
short focusY;	The y position of the cell to be focused.
GRIDITEM * gridObjects;	The pointer to grid items

5.2.2.7.26.83 GRIDITEM Structure

C

```
typedef struct {  
    void * data;  
    uint16_t status;  
} GRIDITEM;
```

Description

Defines the grid item.

Members

Members	Description
void * data;	Indicates if the Grid Item is type GRIDITEM_IS_TEXT or GRIDITEM_IS_BITMAP
uint16_t status;	indicates the status of the Grid Item

5.2.2.7.26.84 IMAGE_STRETCH Enumeration

C

```
typedef enum {  
    IMAGE_NORMAL = 1,  
    IMAGE_X2 = 2  
} IMAGE_STRETCH;
```

Description

GFX image stretch

Remarks

None.

5.2.2.7.26.85 IMG_FILE_FORMAT Type

C

```
typedef enum _IMG_FILE_FORMAT IMG_FILE_FORMAT;
```

Description

IMG_ImageFileFormat specifies all the supported image file formats

5.2.2.7.26.86 IMG_FILE_SYSTEM_API Type

C

```
typedef struct _IMG_FILE_SYSTEM_API IMG_FILE_SYSTEM_API;
```

Description

IMG_FileSystemAPI holds function pointers to the used File System APIs

5.2.2.7.26.87 IMG_LOOP_CALLBACK Type

C

```
typedef void (* IMG_LOOP_CALLBACK)(void);
```

Description

IMG_LoopCallback is a callback function which is called in every loop of the decoding cycle so that user application can do some

maintainance activities

5.2.2.7.26.88 IMG_PIXEL_OUTPUT Type

C

```
typedef void (* IMG_PIXEL_OUTPUT)(IMG_PIXEL_XY_RGB_888 *pPix);
```

Description

IMG_PixelOutput is a callback function which receives the color information of the output pixel

5.2.2.7.26.89 IMG_PIXEL_XY_RGB_888 Type

C

```
typedef struct _IMG_PIXEL_XY_RGB_888 IMG_PIXEL_XY_RGB_888;
```

Description

IMG_PixelRgb holds the RGB information of a pixel in BYTES

5.2.2.7.26.90 INPUT_DEVICE_EVENT Enumeration

C

```
typedef enum {
    EVENT_INVALID = 0,
    EVENT_MOVE,
    EVENT_PRESS,
    EVENT_STILLPRESS,
    EVENT_RELEASE,
    EVENT_KEYSCAN,
    EVENT_CHARCODE,
    EVENT_SET,
    EVENT_SET_STATE,
    EVENT_CLR_STATE
} INPUT_DEVICE_EVENT;
```

Description

INPUT_DEVICE_EVENT

This enumeration specifies the different user input device events supported in the library.

Members

Members	Description
EVENT_INVALID = 0	An invalid event.
EVENT_MOVE	A move event.
EVENT_PRESS	A press event.
EVENT_STILLPRESS	A continuous press event.
EVENT_RELEASE	A release event.
EVENT_KEYSCAN	A keyscan event, parameters has the object ID keyboard scan code.
EVENT_CHARCODE	Character code is presented at the parameters.
EVENT_SET	A generic set event.
EVENT_SET_STATE	A set state event.
EVENT_CLR_STATE	A clear state event.

Remarks

None.

5.2.2.7.26.91 INPUT_DEVICE_TYPE Enumeration

C

```
typedef enum {
    TYPE_UNKNOWN = 0,
    TYPE_KEYBOARD,
    TYPE_TOUCHSCREEN,
    TYPE_MOUSE,
    TYPE_TIMER,
    TYPE_SYSTEM
} INPUT_DEVICE_TYPE;
```

Description

INPUT_DEVICE_TYPE

This enumeration specifies the different user input devices supported in the library.

Members

Members	Description
TYPE_UNKNOWN = 0	Unknown device.
TYPE_KEYBOARD	Keyboard.
TYPE_TOUCHSCREEN	Touchscreen.
TYPE_MOUSE	Mouse.
TYPE_TIMER	Timer.
TYPE_SYSTEM	System Messages.

Remarks

None.

5.2.2.7.26.92 int16_gfx_image_prog Type

C

```
typedef int16_prog_pack int16_gfx_image_prog;
```

Description

This is type int16_gfx_image_prog.

5.2.2.7.26.93 int16_prog Type

C

```
typedef __prog__ int16_t int16_prog;
```

Description

This is type int16_prog.

5.2.2.7.26.94 int16_prog_pack Type

C

```
typedef __pack_upper_byte int16_t int16_prog_pack;
```

Description

This is type int16_prog_pack.

5.2.2.7.26.95 int32_gfx_image_prog Type

C

```
typedef int32_prog_pack int32_gfx_image_prog;
```

Description

This is type int32_gfx_image_prog.

5.2.2.7.26.96 int32_prog Type

C

```
typedef __prog__ int32_t int32_prog;
```

Description

This is type int32_prog.

5.2.2.7.26.97 int32_prog_pack Type

C

```
typedef __pack_upper_byte int32_t int32_prog_pack;
```

Description

This is type int32_prog_pack.

5.2.2.7.26.98 int8_gfx_image_prog Type

C

```
typedef int8_prog_pack int8_gfx_image_prog;
```

Description

This is type int8_gfx_image_prog.

5.2.2.7.26.99 int8_prog Type

C

```
typedef __prog__ int8_t int8_prog;
```

Description

This is type int8_prog.

5.2.2.7.26.100 int8_prog_pack Type

C

```
typedef __pack_upper_byte int8_t int8_prog_pack;
```

Description

This is type int8_prog_pack.

5.2.2.7.26.101 KEYMEMBER Structure

C

```
typedef struct {
```



```

uint16_t left;
uint16_t top;
uint16_t right;
uint16_t bottom;
uint16_t index;
uint16_t state;
bool update;
GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE command;
GFX_XCHAR * pKeyName;
int16_t textWidth;
int16_t textHeight;
void * pNextKey;
} KEYMEMBER;

```

Description

Defines the parameters and the strings assigned for each key.

Members

Members	Description
uint16_t left;	Left position of the key
uint16_t top;	Top position of the key
uint16_t right;	Right position of the key
uint16_t bottom;	Bottom position of the key
uint16_t index;	Index of the key in the list
uint16_t state;	State of the key. Either Pressed (GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE) or Released (0)
bool update;	flag to indicate key is to be redrawn with the current state
GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE command;	Command of the key. Either TE_DELETE_COM, TE_SPACE_COM or TE_ENTER_COM.
GFX_XCHAR * pKeyName;	Pointer to the custom text assigned to the key. This is displayed over the face of the key.
int16_t textWidth;	Computed text width, done at creation. Used to predict size and position of text on the key face.
int16_t textHeight;	Computed text height, done at creation. Used to predict size and position of text on the key face.
void * pNextKey;	Pointer to the next key parameters.

5.2.2.7.26.102 LAYER_ACTIONS Enumeration

C

```

typedef enum {
    ENABLE = 0x80,
    DISABLE = 0,
    UPDATE,
    SET_SIZE,
    SET_PAGE,
    SET_PAGE_START,
    SET_LAYER_START
} LAYER_ACTIONS;

```

Description

GFX layering actions

Remarks

None.

5.2.2.7.26.103 PALETTE_ENTRY Union

C

```
typedef union {
    uint16_t value;
    struct {
        uint16_t r : 5;
        uint16_t g : 6;
        uint16_t b : 5;
    } color;
    struct {
        uint16_t reserved : 12;
        uint16_t luma : 4;
    } monochrome;
} PALETTE_ENTRY;
```

Description

Structure used for the palette entries.

- For TFT: color is defined as 5-6-5 RGB format (5-bits for RED, 6-bits for GREEN and 5-bits for BLUE.
- For Monochrome: 4 bits are used to represent the luma.

Members

Members	Description
uint16_t value;	a 16-bit value representing a color or palette entry
struct { uint16_t r : 5; uint16_t g : 6; uint16_t b : 5; } color;	color value in 5-6-5 RGB format
uint16_t r : 5;	represents the RED component
uint16_t g : 6;	represents the GREEN component
uint16_t b : 5;	represents the BLUE component
struct { uint16_t reserved : 12; uint16_t luma : 4; } monochrome;	monochrome LUMA value
uint16_t reserved : 12;	reserved, used as a filler
uint16_t luma : 4;	monochrome LUMA value

5.2.2.7.26.104 PALETTE_EXTERNAL Type

C

```
typedef GFX_EXTDATA PALETTE_EXTERNAL;
```

Description

Structure for palette stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

5.2.2.7.26.105 PALETTE_FLASH Structure

C

```
typedef struct {
    short type;
    PALETTE_HEADER header;
```

```

    PALETTE_ENTRY * pPaletteEntry;
} PALETTE_FLASH;

```

Description

Structure for the palette stored in FLASH memory.

Members

Members	Description
short type;	Type must be FLASH
PALETTE_HEADER header;	Contains information on the palette
PALETTE_ENTRY * pPaletteEntry;	Pointer to the palette. Number of entries is determined by the header.

5.2.2.7.26.106 PALETTE_HEADER Structure**C**

```

typedef struct {
    uint16_t id;
    uint16_t length;
} PALETTE_HEADER;

```

Description

Structure for the palette header.

Members

Members	Description
uint16_t id;	User defined ID
uint16_t length;	number of palette entries (number of colors in the palette)

5.2.2.7.26.107 PUTIMAGE_PARAM Structure**C**

```

typedef struct {
    uint16_t width;
    uint16_t height;
    uint16_t xoffset;
    uint16_t yoffset;
    IMAGE_STRETCH stretch;
    uint8_t transparentFlag;
    GFX_COLOR transparentColor;
    uint8_gfx_image_prog* image;
    GFX_COLOR palette[256];
} PUTIMAGE_PARAM;

```

Description

GFX put image param

Members

Members	Description
uint16_t width;	Partial Image width
uint16_t height;	Partial Image height
uint16_t xoffset;	xoffset of image
uint16_t yoffset;	yoffset of image
IMAGE_STRETCH stretch;	stretch of the image
uint8_t transparentFlag;	transparent Color flag

GFX_COLOR transparentColor;	transparent Color
uint8_gfx_image_prog* image;	pointer to the image
GFX_COLOR palette[256];	palette used for <8BPP images

Remarks

None.

5.2.2.7.26.108 uint16_gfx_image_prog Type**C**

```
typedef uint16_prog_pack uint16_gfx_image_prog;
```

Description

This is type uint16_gfx_image_prog.

5.2.2.7.26.109 uint16_prog Type**C**

```
typedef __prog__ uint16_t uint16_prog;
```

Description

This is type uint16_prog.

5.2.2.7.26.110 uint16_prog_pack Type**C**

```
typedef __pack_upper_byte uint16_t uint16_prog_pack;
```

Description

This is type uint16_prog_pack.

5.2.2.7.26.111 uint32_gfx_image_prog Type**C**

```
typedef uint32_prog_pack uint32_gfx_image_prog;
```

Description

This is type uint32_gfx_image_prog.

5.2.2.7.26.112 uint32_prog Type**C**

```
typedef __prog__ uint32_t uint32_prog;
```

Description

This is type uint32_prog.

5.2.2.7.26.113 uint32_prog_pack Type**C**

```
typedef __pack_upper_byte uint32_t uint32_prog_pack;
```

Description

This is type uint32_prog_pack.

5.2.2.7.26.114 uint8_gfx_image_prog Type

C

```
typedef uint8_prog_pack uint8_gfx_image_prog;
```

Description

GFX Image data Types

5.2.2.7.26.115 uint8_prog Type

C

```
typedef __prog__ uint8_t uint8_prog;
```

Description

General program space data types

5.2.2.7.26.116 uint8_prog_pack Type

C

```
typedef __pack_upper_byte uint8_t uint8_prog_pack;
```

Description

This is type uint8_prog_pack.

5.2.2.7.26.117 BLACK Macro

C

```
#define BLACK GFX_RGBConvert(0, 0, 0)
```

Description

This is macro BLACK.

5.2.2.7.26.118 BLUE Macro

C

```
#define BLUE GFX_RGBConvert(0, 0, 128)
```

Description

This is macro BLUE.

5.2.2.7.26.119 BRIGHTBLUE Macro

C

```
#define BRIGHTBLUE GFX_RGBConvert(0, 0, 255)
```

Description

This is macro BRIGHTBLUE.

5.2.2.7.26.120 BRIGHTCYAN Macro

C

```
#define BRIGHTCYAN GFX_RGBConvert(0, 255, 255)
```

Description

This is macro BRIGHTCYAN.

5.2.2.7.26.121 BRIGHTGREEN Macro

C

```
#define BRIGHTGREEN GFX_RGBConvert(0, 255, 0)
```

Description

This is macro BRIGHTGREEN.

5.2.2.7.26.122 BRIGHTMAGENTA Macro

C

```
#define BRIGHTMAGENTA GFX_RGBConvert(255, 0, 255)
```

Description

This is macro BRIGHTMAGENTA.

5.2.2.7.26.123 BRIGHTRED Macro

C

```
#define BRIGHTRED GFX_RGBConvert(255, 0, 0)
```

Description

This is macro BRIGHTRED.

5.2.2.7.26.124 BRIGHTYELLOW Macro

C

```
#define BRIGHTYELLOW GFX_RGBConvert(255, 255, 0)
```

Description

This is macro BRIGHTYELLOW.

5.2.2.7.26.125 BROWN Macro

C

```
#define BROWN GFX_RGBConvert(255, 128, 0)
```

Description

This is macro BROWN.

5.2.2.7.26.126 BURLYWOOD Macro

C

```
#define BURLYWOOD GFX_RGBConvert(222, 184, 135)
```

Description

This is macro BURLYWOOD.

5.2.2.7.26.127 CH_CLR0 Macro**C**

```
#define CH_CLR0 WHITE( )
```

Description

Bright Blue

5.2.2.7.26.128 CH_CLR1 Macro**C**

```
#define CH_CLR1 BLACK( )
```

Description

Bright Red

5.2.2.7.26.129 CH_CLR10 Macro**C**

```
#define CH_CLR10 WHITE( )
```

Description

Light Blue

5.2.2.7.26.130 CH_CLR11 Macro**C**

```
#define CH_CLR11 BLACK( )
```

Description

Light Red

5.2.2.7.26.131 CH_CLR12 Macro**C**

```
#define CH_CLR12 WHITE( )
```

Description

Light Green

5.2.2.7.26.132 CH_CLR13 Macro**C**

```
#define CH_CLR13 BLACK( )
```

Description

Light Yellow

5.2.2.7.26.133 CH_CLR14 Macro

C

```
#define CH_CLR14 WHITE()
```

Description

Light Orange

5.2.2.7.26.134 CH_CLR15 Macro

C

```
#define CH_CLR15 BLACK()
```

Description

Gold

5.2.2.7.26.135 CH_CLR2 Macro

C

```
#define CH_CLR2 WHITE()
```

Description

Bright Green

5.2.2.7.26.136 CH_CLR3 Macro

C

```
#define CH_CLR3 BLACK()
```

Description

Bright Yellow

5.2.2.7.26.137 CH_CLR4 Macro

C

```
#define CH_CLR4 WHITE()
```

Description

Orange

5.2.2.7.26.138 CH_CLR5 Macro

C

```
#define CH_CLR5 BLACK()
```

Description

Blue

5.2.2.7.26.139 CH_CLR6 Macro

C

```
#define CH_CLR6 WHITE()
```


Description

Red

5.2.2.7.26.140 CH_CLR7 Macro

C

```
#define CH_CLR7 BLACK( )
```

Description

Green

5.2.2.7.26.141 CH_CLR8 Macro

C

```
#define CH_CLR8 WHITE( )
```

Description

Yellow

5.2.2.7.26.142 CH_CLR9 Macro

C

```
#define CH_CLR9 BLACK( )
```

Description

Dark Orange

5.2.2.7.26.143 CHART_MARGIN Macro

C

```
#define CHART_MARGIN 2           // Margin from the edges.
```

Description

Margin from the edges.

5.2.2.7.26.144 CHART_YGRIDCOUNT Macro

C

```
#define CHART_YGRIDCOUNT 6           // defines the number of grids to be drawn on the y-axis  
(includes the origin at y).
```

Description

defines the number of grids to be drawn on the y-axis (includes the origin at y).

5.2.2.7.26.145 ConvertColor25 Macro

C

```
#define ConvertColor25(color) (GFX_COLOR)((color & (0x00FCFCFCul))>>2)
```

Description

This is macro ConvertColor25.

5.2.2.7.26.146 ConvertColor50 Macro

C

```
#define ConvertColor50(color) (GFX_COLOR)((color & (0x00FEFEFEul))>>1)
```

Description

75, 50, 25 percentage color conversion macros. Used in converting colors to these percentages when performing alpha blending of colors using software alpha blending operation.

5.2.2.7.26.147 ConvertColor75 Macro

C

```
#define ConvertColor75(color) (GFX_COLOR)(ConvertColor50(color) + ConvertColor25(color))
```

Description

This is macro ConvertColor75.

5.2.2.7.26.148 CopyPageWindow Macro

C

```
#define CopyPageWindow(alphaArea, width, height) \
    GFX_AlphaBlendWindow(alphaArea,width,height,100)
```

Description

None

Preconditions

None

Parameters

Parameters	Description
None	None

Returns

None

Example

None

5.2.2.7.26.149 CYAN Macro

C

```
#define CYAN GFX_RGBConvert(0, 128, 128)
```

Description

This is macro CYAN.

5.2.2.7.26.150 DARKGRAY Macro

C

```
#define DARKGRAY GFX_RGBConvert(64, 64, 64)
```

Description

This is macro DARKGRAY.

5.2.2.7.26.151 DARKORANGE Macro**C**

```
#define DARKORANGE GFX_RGBConvert(255, 140, 0)
```

Description

This is macro DARKORANGE.

5.2.2.7.26.152 GFX_ActivePageGet Macro**C**

```
#define GFX_ActivePageGet GFX_Primitive_instance.activePage
```

Preconditions

None.

Returns

None.

Example

Remarks:

5.2.2.7.26.153 GFX_AlphaParamsSet Macro**C**

```
#define GFX_AlphaParamsSet(fArea, fLeft, fTop, bArea, bLeft, bTop, dArea, dLeft, dTop) \
                                                                    GFX_Primitive_instance.alphaArea.foregroundPage
= fArea;\                                                                    GFX_Primitive_instance.alphaArea.foregroundLeft
= fLeft;\                                                                    GFX_Primitive_instance.alphaArea.foregroundTop
= fTop;\                                                                    GFX_Primitive_instance.alphaArea.backgroundPage
= bArea;\                                                                    GFX_Primitive_instance.alphaArea.backgroundLeft
= bLeft;\                                                                    GFX_Primitive_instance.alphaArea.backgroundTop
= bTop;\                                                                    GFX_Primitive_instance.alphaArea.destinationPag
e                                                                    GFX_Primitive_instance.alphaArea.destinationLef
t                                                                    GFX_Primitive_instance.alphaArea.destinationTop
= dArea;\
= dLeft;\
= dTop
```

Description

None

Preconditions

None

Parameters

Parameters	Description
None	None

Returns

None

Example

None

5.2.2.7.26.154 GFX_BUFFER1 Macro**C**

```
#define GFX_BUFFER1 (0)
```

Description

This is macro GFX_BUFFER1.

5.2.2.7.26.155 GFX_BUFFER2 Macro**C**

```
#define GFX_BUFFER2 (1)
```

Description

This is macro GFX_BUFFER2.

5.2.2.7.26.156 GFX_COMP_MASK Macro**C**

```
#define GFX_COMP_MASK 0xF000
```

Description

This is macro GFX_COMP_MASK.

5.2.2.7.26.157 GFX_CONFIG_OBJECT_METER_DEGREECOUNT Macro**C**

```
#define GFX_CONFIG_OBJECT_METER_DEGREECOUNT 9
```

Description

This is a Meter compile time option to define how many degrees per scale, computed per octant

Example

for 5 division per octant $45/5 = 9$. So every 9 degrees a scale is drawn for a 5 scale division per octant.

5.2.2.7.26.158 GFX_CONFIG_OBJECT_METER_RESOLUTION Macro**C**

```
#define GFX_CONFIG_OBJECT_METER_RESOLUTION 10
```

Description

This is a Meter compile time option to define the factor that the meter widget will divide min**Value**, max**Value**

5.2.2.7.26.159 GFX_CONFIG_OBJECT_METER_SCALE_COUNT Macro**C**

```
#define GFX_CONFIG_OBJECT_METER_SCALE_COUNT 4
```

Description

This is a Meter compile time option to define how many characters will be allocated for the scale labels. Use this define in accordance to the max**Value**-min**Value**.

Example

if max**Value**-min**Value** = 500, SCALECHARCOUNT should be 3. if max**Value**-min**Value** = 90, SCALECHARCOUNT = 2 You must include the decimal point if this feature is enabled (see MTR_ACCURACY state bit).

5.2.2.7.26.160 GFX_FONT_SPACE Macro**C**

```
#define GFX_FONT_SPACE const
```

Description

Font space section. The fonts can be located in ps**v** (constant) or program space in PIC24/dsPIC MCUs. This define allows for switching of the pointer type used to access the font structure in memory See: GraphicsConfig.h for the application define.

5.2.2.7.26.161 GFX_FontAlignmentSet Macro**C**

```
#define GFX_FontAlignmentSet(align) GFX_Primitive_instance.currentFont.alignment = align;
```

Description

- 1 when font orientation is vertical
- 0 when font orientation is horizontal

Preconditions

None.

Returns

None.

Example**5.2.2.7.26.162 GFX_GOL_ANALOGCLOCK_DISABLED Macro****C**

```
#define GFX_GOL_ANALOGCLOCK_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

5.2.2.7.26.163 GFX_GOL_ANALOGCLOCK_DRAW Macro

C

```
#define GFX_GOL_ANALOGCLOCK_DRAW 0x4000 // Bit to indicate button must be redrawn.
```

Description

Bit to indicate button must be redrawn.

5.2.2.7.26.164 GFX_GOL_ANALOGCLOCK_HIDE Macro

C

```
#define GFX_GOL_ANALOGCLOCK_HIDE 0x8000 // Bit to indicate button must be removed from screen.
```

Description

Bit to indicate button must be removed from screen.

5.2.2.7.26.165 GFX_GOL_ANALOGCLOCK_PRESSED Macro

C

```
#define GFX_GOL_ANALOGCLOCK_PRESSED 0x0004 // Bit for press state.
```

Description

Bit for press state.

5.2.2.7.26.166 GFX_GOL_ANALOGCLOCK_TICK Macro

C

```
#define GFX_GOL_ANALOGCLOCK_TICK 0x1000 // Bit to tick second hand
```

Description

Bit to tick second hand

5.2.2.7.26.167 GFX_GOL_ANALOGCLOCK_UPDATE_HOUR Macro

C

```
#define GFX_GOL_ANALOGCLOCK_UPDATE_HOUR 0x2000 // Bit to indicate hour hand must be redrawn
```

Description

Bit to indicate hour hand must be redrawn

5.2.2.7.26.168 GFX_GOL_ANALOGCLOCK_UPDATE_MINUTE Macro

C

```
#define GFX_GOL_ANALOGCLOCK_UPDATE_MINUTE 0x0100 // Bit to indicate minute hand must be redrawn
```

Description

Bit to indicate minute hand must be redrawn

5.2.2.7.26.169 GFX_GOL_ANALOGCLOCK_UPDATE_SECOND Macro

C

```
#define GFX_GOL_ANALOGCLOCK_UPDATE_SECOND 0x0200 // Bit to indicate minute hand must be redrawn
```

Description

Bit to indicate minute hand must be redrawn

5.2.2.7.26.170 GFX_GOL_ButtonPressStateImageGet Macro

C

```
#define GFX_GOL_ButtonPressStateImageGet(pObject, pImage) \
(((GFX_GOL_BUTTON *)pObject)->pPressImage)
```

Description

This function gets the image used when in the pressed state.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to the image resource.

Example

None.

5.2.2.7.26.171 GFX_GOL_ButtonPressStateImageSet Macro

C

```
#define GFX_GOL_ButtonPressStateImageSet(pObject, pImage) \
(((GFX_GOL_BUTTON *)pObject)->pPressImage = pImage)
```

Description

This function sets the image to be used when in the pressed state.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pImage	pointer to the image resource.

Returns

None.

Example

```
// assume ImageIcon is a valid GFX_RESOURCE_HDR
```

```
GFX_RESOURCE_HDR *pMyIcon = &ImageIcon;  
GFX_GOL_BUTTON *pButton;  
  
GFX_GOL_ButtonPressStateImageSet(pButton , pMyIcon);
```

5.2.2.7.26.172 GFX_GOL_ButtonReleaseStateImageGet Macro

C

```
#define GFX_GOL_ButtonReleaseStateImageGet(pObject, pImage) \  
(((GFX_GOL_BUTTON *)pObject)->pReleaseImage)
```

Description

This function gets the image used when in the released state.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to the image resource.

Example

None.

5.2.2.7.26.173 GFX_GOL_ButtonReleaseStateImageSet Macro

C

```
#define GFX_GOL_ButtonReleaseStateImageSet(pObject, pImage) \  
(((GFX_GOL_BUTTON *)pObject)->pReleaseImage = pImage)
```

Description

This function sets the image to be used when in the released state.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pImage	pointer to the image resource.

Returns

None.

Example

```
// assume ImageIcon is a valid GFX_RESOURCE_HDR  
  
GFX_RESOURCE_HDR *pMyIcon = &ImageIcon;  
GFX_GOL_BUTTON *pButton;  
  
GFX_GOL_ButtonReleaseStateImageSet(pButton , pMyIcon);
```


5.2.2.7.26.174 GFX_GOL_ButtonTextGet Macro**C**

```
#define GFX_GOL_ButtonTextGet(pObject) (((GFX_GOL_BUTTON *)pObject)->pText)
```

Description

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to text string.

Example

```
GFX_XCHAR *pChar;
GFX_GOL_BUTTON GFX_GOL_BUTTON[2];

pChar = GFX_GOL_ButtonTextGet(GFX_GOL_BUTTON[0]);
```

5.2.2.7.26.175 GFX_GOL_CHART_3D_ENABLE Macro**C**

```
#define GFX_GOL_CHART_3D_ENABLE 0x0008 // Bit to indicate that bar charts are to be drawn
with 3-D effect
```

Description

Bit to indicate that bar charts are to be drawn with 3-D effect

5.2.2.7.26.176 GFX_GOL_CHART_BAR Macro**C**

```
#define GFX_GOL_CHART_BAR 0x0200 // Bit to indicate the chart is type bar. If both PIE
and BAR types are set BAR type has higher priority.
```

Description

Bit to indicate the chart is type bar. If both PIE and BAR types are set BAR type has higher priority.

5.2.2.7.26.177 GFX_GOL_CHART_BAR_HOR Macro**C**

```
#define GFX_GOL_CHART_BAR_HOR 0x0240 // These bits (with CH_BAR bit set), sets the bar
chart to be drawn horizontally.
```

Description

These bits (with CH_BAR bit set), sets the bar chart to be drawn horizontally.

5.2.2.7.26.178 GFX_GOL_CHART_DISABLED Macro

C

```
#define GFX_GOL_CHART_DISABLED 0x0002    // Bit for disabled state.
```

Description

Bit for disabled state.

5.2.2.7.26.179 GFX_GOL_CHART_DONUT Macro

C

```
#define GFX_GOL_CHART_DONUT 0x0140    // These bits (with CH_PIE bit set), sets the pie  
chart to be drawn in a donut shape.
```

Description

These bits (with CH_PIE bit set), sets the pie chart to be drawn in a donut shape.

5.2.2.7.26.180 GFX_GOL_CHART_DRAW Macro

C

```
#define GFX_GOL_CHART_DRAW 0x4000    // Bit to indicate chart must be redrawn.
```

Description

Bit to indicate chart must be redrawn.

5.2.2.7.26.181 GFX_GOL_CHART_DRAW_DATA Macro

C

```
#define GFX_GOL_CHART_DRAW_DATA 0x2000    // Bit to indicate data portion of the chart must  
be redrawn.
```

Description

Bit to indicate data portion of the chart must be redrawn.

5.2.2.7.26.182 GFX_GOL_CHART_HIDE Macro

C

```
#define GFX_GOL_CHART_HIDE 0x8000    // Bit to indicate chart must be removed from screen.
```

Description

Bit to indicate chart must be removed from screen.

5.2.2.7.26.183 GFX_GOL_CHART_LEGEND Macro

C

```
#define GFX_GOL_CHART_LEGEND 0x0001    // Bit to indicate that legend is to be shown. Usable  
only when seriesCount > 1.
```

Description

Bit to indicate that legend is to be shown. Usable only when seriesCount > 1.

5.2.2.7.26.184 GFX_GOL_CHART_NUMERIC Macro

C

```
#define GFX_GOL_CHART_NUMERIC 0x0080 // This bit is used only for bar charts. If this bit is set, it indicates that the
```

Description

This bit is used only for bar charts. If this bit is set, it indicates that the bar chart labels for variables are numeric. If this bit is not set, it indicates that the bar chart labels for variables are alphabets.

5.2.2.7.26.185 GFX_GOL_CHART_PERCENT Macro

C

```
#define GFX_GOL_CHART_PERCENT 0x0010 // Bit to indicate that the pie chart will be drawn with percentage values shown for the sample data.
```

Description

Bit to indicate that the pie chart will be drawn with percentage values shown for the sample data. For bar chart, if CH_VALUE is set, it toggles the value shown to percentage.

5.2.2.7.26.186 GFX_GOL_CHART_PIE Macro

C

```
#define GFX_GOL_CHART_PIE 0x0100 // Bit to indicate the chart is type pie. If both PIE and BAR types are set BAR type has higher priority.
```

Description

Bit to indicate the chart is type pie. If both PIE and BAR types are set BAR type has higher priority.

5.2.2.7.26.187 GFX_GOL_CHART_VALUE Macro

C

```
#define GFX_GOL_CHART_VALUE 0x0004 // Bit to indicate that the values of the bar chart data or pie chart data are to be shown
```

Description

Bit to indicate that the values of the bar chart data or pie chart data are to be shown

5.2.2.7.26.188 GFX_GOL_ChartAxisLabelFontGet Macro

C

```
#define GFX_GOL_ChartAxisLabelFontGet(pCh) (((GFX_GOL_CHART *)pCh)->prm.pAxisLabelsFont)
```

Description

Macros: GFX_GOL_ChartAxisLabelFontGet(pCh)

This macro returns the location of the font used for the X and Y axis labels of the chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the address of the current font used for the title text.

Side Effects

none

5.2.2.7.26.189 GFX_GOL_ChartAxisLabelFontSet Macro**C**

```
#define GFX_GOL_ChartAxisLabelFontSet(pCh, pNewFont) (((GFX_GOL_CHART *)pCh)->prm.pAxisLabelsFont = pNewFont)
```

Description

Macros: GFX_GOL_ChartAxisLabelFontSet(pCh, pNewFont)

This macro sets the location of the font used for the X and Y axis labels of the chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.
pNewFont	Pointer to the font used.

Returns

none.

Side Effects

none

Example

See ChCreate() example.

5.2.2.7.26.190 GFX_GOL_ChartColorTableGet Macro**C**

```
#define GFX_GOL_ChartColorTableGet(pCh) (((GFX_GOL_CHART *)pCh)->prm.pColor)
```

Description

Macros: GFX_GOL_ChartColorTableGet(pCh)

This macro returns the current color table used for the pie and bar charts.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the address of the color table used.

Side Effects

none

5.2.2.7.26.191 GFX_GOL_ChartColorTableSet Macro**C**

```
#define GFX_GOL_ChartColorTableSet(pCh, pNewTable) (((GFX_GOL_CHART *)pCh)->prm.pColor) = pNewTable)
```

Description

Macros: GFX_GOL_ChartColorTableSet(pCh, pNewTable)

This macro sets the color table used to draw the data in pie and bar charts.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.
pNewTable	Pointer to the color table that will be used.

Returns

none.

Side Effects

none

5.2.2.7.26.192 GFX_GOL_ChartGridLabelFontGet Macro**C**

```
#define GFX_GOL_ChartGridLabelFontGet(pCh) (((GFX_GOL_CHART *)pCh)->prm.pGridLabelsFont)
```

Description

Macros: GFX_GOL_ChartGridLabelFontGet(pCh)

This macro returns the location of the font used for the X and Y axis grid labels of the chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the address of the current font used for the title text.

Side Effects

none

5.2.2.7.26.193 GFX_GOL_ChartGridLabelFontSet Macro**C**

```
#define GFX_GOL_ChartGridLabelFontSet(pCh, pNewFont) (((GFX_GOL_CHART *)pCh)->prm.pGridLabelsFont = pNewFont)
```

Description

Macros: GFX_GOL_ChartGridLabelFontSet(pCh, pNewFont)

This macro sets the location of the font used for the X and Y axis grid labels of the chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.
pNewFont	Pointer to the font used.

Returns

none.

Side Effects

none

Example

See ChCreate() example.

5.2.2.7.26.194 GFX_GOL_ChartPercentMaxGet Macro**C**

```
#define GFX_GOL_ChartPercentMaxGet(pCh) (pCh->prm.perMax)
```

Description

Macros: ChGetPercentMax(pCh)

This macro returns the current maximum value of the percentage range that will be drawn for bar charts when CH_PERCENTAGE bit state is set.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the maximum percentage value set when bar charts are drawn.

Side Effects

none

5.2.2.7.26.195 GFX_GOL_ChartPercentMinGet Macro**C**

```
#define GFX_GOL_ChartPercentMinGet(pCh) (pCh->prm.perMin)
```

Description

Macros: ChGetPercentMin(pCh)

This macro returns the current minimum value of the percentage range that will be drawn for bar charts when CH_PERCENTAGE bit state is set.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the minimum percentage value when bar charts are drawn.

Side Effects

none

5.2.2.7.26.196 GFX_GOL_ChartPercentRangeGet Macro**C**

```
#define GFX_GOL_ChartPercentRangeGet(pCh) (pCh->prm.perMax - pCh->prm.perMin)
```

Description

This macro gets the percentage range for bar charts. The value returned is calculated from percentage max - min. To get the minimum use ChGetPercentMin() and to get the maximum use ChGetPercentMax().

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.

Returns

Percentage range computed from max-min.

Side Effects

none.

5.2.2.7.26.197 GFX_GOL_ChartSampleEndGet Macro

C

```
#define GFX_GOL_ChartSampleEndGet(pCh) (((GFX_GOL_CHART *)pCh)->prm.smpLEnd
```

Description

Macros: GFX_GOL_ChartSampleEndGet(pCh)

This macro returns the sampling end value.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the sample end point.

Side Effects

none

5.2.2.7.26.198 GFX_GOL_ChartSampleLabelGet Macro

C

```
#define GFX_GOL_ChartSampleLabelGet(pCh) (((GFX_GOL_CHART *)pCh)->prm.pSmpLLabel)
```

Description

Macros: GFX_GOL_ChartSampleLabelGet(pCh)

This macro returns the address of the current text string used for the sample axis label of the bar chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the pointer to the current sample axis label text of the bar chart.

Side Effects

none

5.2.2.7.26.199 GFX_GOL_ChartSampleLabelSet Macro

C

```
#define GFX_GOL_ChartSampleLabelSet(pCh, pNewXLabel) (((GFX_GOL_CHART *)pCh)->prm.pSmpLLabel =  
pNewXLabel)
```

Description

Macros: GFX_GOL_ChartSampleLabelSet(pCh, pNewXLabel)

This macro sets the address of the current text string used for the sample axis label of the bar chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.
pNewXLabel	pointer to the string to be used as an sample axis label of the bar chart.

Returns

none.

Side Effects

none

Example

See ChCreate() example.

5.2.2.7.26.200 GFX_GOL_ChartSampleRangeGet Macro**C**

```
#define GFX_GOL_ChartSampleRangeGet(pCh) (GFX_GOL_ChartSampleEndGet(pCh) -  
GFX_GOL_ChartSampleStartGet(pCh))
```

Description

This macro gets the sample range for pie or bar charts. The value returned is calculated from smplEnd - smplStart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.

Returns

Sample range computed from smplEnd - smplStart.

Side Effects

none.

5.2.2.7.26.201 GFX_GOL_ChartSampleStartGet Macro**C**

```
#define GFX_GOL_ChartSampleStartGet(pCh) (((GFX_GOL_CHART *)pCh)->prm.smplStart)
```

Description

Macros: GFX_GOL_ChartSampleStartGet(pCh)

This macro returns the sampling start value.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the sample start point.

Side Effects

none

5.2.2.7.26.202 GFX_GOL_ChartSeriesHide Macro**C**

```
#define GFX_GOL_ChartSeriesHide(pCh, seriesNum) (GFX_GOL_ChartDataSet(pCh, seriesNum, HIDE_DATA))
```

Description

Macros: GFX_GOL_ChartSeriesHide(CHART *pCh, uint16_t seriesNum)

This macro sets the specified data series number show flag to be set to **HIDE_DATA**.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.
seriesNum	The data series number that will be modified. If this number is zero, all the entries' flag in the list will be set to HIDE_DATA .

Returns

Returns the same passed number if successful otherwise -1 if unsuccessful.

Side Effects

none

Example

See ChShowSeries() example.

5.2.2.7.26.203 GFX_GOL_ChartShowSeriesCountGet Macro**C**

```
#define GFX_GOL_ChartShowSeriesCountGet(pCh) (pCh->prm.seriesCount)
```

Description

Macros: GFX_GOL_ChartShowSeriesCountGet(pCh)

This macro shows the number of data series that has its show flag set to **SHOW_DATA**

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the number of data series with its show flag set to [SHOW_DATA](#).

Side Effects

none

5.2.2.7.26.204 GFX_GOL_ChartShowSeriesSet Macro**C**

```
#define GFX_GOL_ChartShowSeriesSet(pCh, seriesNum) (GFX_GOL_ChartDataSeriesSet(pCh, seriesNum, SHOW_DATA))
```

Description

Macros: short GFX_GOL_ChartShowSeriesSet(CHART *pCh, uint16_t seriesNum)

This macro sets the specified data series number show flag to be set to [SHOW_DATA](#).

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.
seriesNum	The data series number that will be modified. If this number is zero, all the entries' flag in the list will be set to SHOW_DATA .

Returns

Returns the same passed number if successful otherwise -1 if unsuccessful.

Side Effects

none

Example

```
// from the example in ChCreate() we change the items to be shown when
// GFX_OBJ_Draw() is called.

// reset all data series to be HIDE_DATA
ChHideSeries(pMyChart, 0);
// set data series 1 (V1Data) to be shown
ChShowSeries(pMyChart, 1);
// draw the chart
GFX_OBJ_Draw();
.....
*
```

5.2.2.7.26.205 GFX_GOL_ChartShowSeriesStatusGet Macro**C**

```
#define GFX_GOL_ChartShowSeriesStatusGet(pDSeries) (pDSeries->show)
```

Description

Macros: GFX_GOL_ChartShowSeriesStatusGet(pDSeries)

This macro returns the show ID status of the [DATASERIES](#).

Preconditions

none

Parameters

Parameters	Description
pDSeries	Pointer to the data series(DATASERIES) that is being checked.

Returns

Returns the status of the show flag. 1 - ([SHOW_DATA](#)) means that the show status flag is set. 0 - ([HIDE_DATA](#)) means that the show status flag is not set.

Side Effects

none

5.2.2.7.26.206 GFX_GOL_ChartTitleFontGet Macro**C**

```
#define GFX_GOL_ChartTitleFontGet(pCh) (((GFX_GOL_CHART *)pCh)->prm.pTitleFont)
```

Description

Macros: GFX_GOL_ChartTitleFontGet(pCh)

This macro returns the location of the font used for the title of the chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the address of the current font used for the title text.

Side Effects

none

5.2.2.7.26.207 GFX_GOL_ChartTitleFontSet Macro**C**

```
#define GFX_GOL_ChartTitleFontSet(pCh, pNewFont) (((GFX_GOL_CHART *)pCh)->prm.pTitleFont = pNewFont)
```

Description

Macros: GFX_GOL_ChartTitleFontSet(pCh, pNewFont)

This macro sets the location of the font used for the title of the chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.
pNewFont	Pointer to the font used.

Returns

none.

Side Effects

none

Example

See ChCreate() example.

5.2.2.7.26.208 GFX_GOL_ChartTitleGet Macro**C**

```
#define GFX_GOL_ChartTitleGet(pCh) (((GFX_GOL_CHART *)pCh)->prm.pTitle)
```

Description

Macros: GFX_GOL_ChartTitleGet(pCh)

This macro returns the address of the current text string used for the title of the chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the pointer to the current title text used.

Side Effects

none

5.2.2.7.26.209 GFX_GOL_ChartTitleSet Macro**C**

```
#define GFX_GOL_ChartTitleSet(pCh, pNewTitle) (((GFX_GOL_CHART *)pCh)->prm.pTitle = pNewTitle)
```

Description

Macros: GFX_GOL_ChartTitleSet(pCh, pNewTitle)

This macro sets the address of the current text string used for the title of the chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.
pNewTitle	pointer to the string to be used as a title of the chart.

Returns

none.

Side Effects

none

Example

See ChCreate() example.

5.2.2.7.26.210 GFX_GOL_ChartValueLabelGet Macro**C**

```
#define GFX_GOL_ChartValueLabelGet(pCh) (((GFX_GOL_CHART *)pCh)->prm.pValLabel)
```

Description

Macros: GFX_GOL_ChartValueLabelGet(pCh)

This macro returns the address of the current text string used for the value axis label of the bar chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the pointer to the current value axis label text of the bar chart.

Side Effects

none

5.2.2.7.26.211 GFX_GOL_ChartValueLabelSet Macro**C**

```
#define GFX_GOL_ChartValueLabelSet(pCh, pNewValueLabel) (((GFX_GOL_CHART *)pCh)->prm.pValLabel = pNewValueLabel)
```

Description

Macros: GFX_GOL_ChartValueLabelSet(pCh, pNewYLabel)

This macro sets the address of the current text string used for the value axis label of the bar chart.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.
pNewYLabel	pointer to the string to be used as an value axis label of the bar chart.

Returns

none.

Side Effects

none

Example

See ChCreate() example.

5.2.2.7.26.212 GFX_GOL_ChartValueMaxGet Macro**C**

```
#define GFX_GOL_ChartValueMaxGet(pCh) (pCh->prm.valMax)
```

Description

Macros: GFX_GOL_ChartValueMaxGet(pCh)

This macro returns the current maximum value that will be drawn for bar charts.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the maximum value set when bar charts are drawn.

Side Effects

none

5.2.2.7.26.213 GFX_GOL_ChartValueMinGet Macro**C**

```
#define GFX_GOL_ChartValueMinGet(pCh) (pCh->prm.valMin)
```

Description

Macros: GFX_GOL_ChartValueMinGet(pCh)

This macro returns the current minimum value that will be drawn for bar charts.

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the object.

Returns

Returns the minimum value set when bar charts are drawn.

Side Effects

none

5.2.2.7.26.214 GFX_GOL_ChartValueRangeGet Macro**C**

```
#define GFX_GOL_ChartValueRangeGet(pCh) (pCh->prm.valMax - pCh->prm.valMin)
```

Description

This macro gets the current range for bar charts. The value returned is calculated from the current (valMax - valMin) set. To get the minimum use ChGetValueMin() and to get the maximum use ChGetValueMax().

Preconditions

none

Parameters

Parameters	Description
pCh	Pointer to the chart object.

Returns

Value range computed from valMax-valMin.

Side Effects

none.

5.2.2.7.26.215 GFX_GOL_CheckBoxTextGet Macro**C**

```
#define GFX_GOL_CheckBoxTextGet(pObject) (((GFX_GOL_CHECKBOX *)pObject)->pText)
```

Description

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to text string.

Example

```
// assume CHECK_BOX_OBJECT is a radio button that exists
GFX_XCHAR *pChar;
GFX_GOL_CHECKBOX *pObject = &CHECK_BOX_OBJECT;

pChar = GFX_GOL_CheckBoxTextGet(pObject);
```


5.2.2.7.26.216 GFX_GOL_CUSTOMCONTROL_DISABLED Macro

C

```
#define GFX_GOL_CUSTOMCONTROL_DISABLED 0x0002 // disabled
```

Description

disabled

5.2.2.7.26.217 GFX_GOL_CUSTOMCONTROL_DRAW Macro

C

```
#define GFX_GOL_CUSTOMCONTROL_DRAW 0x4000 // whole control must be redrawn
```

Description

whole control must be redrawn

5.2.2.7.26.218 GFX_GOL_CUSTOMCONTROL_DRAW_BAR Macro

C

```
#define GFX_GOL_CUSTOMCONTROL_DRAW_BAR 0x2000 // control bar should be redrawn
```

Description

control bar should be redrawn

5.2.2.7.26.219 GFX_GOL_CUSTOMCONTROL_HIDE Macro

C

```
#define GFX_GOL_CUSTOMCONTROL_HIDE 0x8000 // must be removed from screen
```

Description

must be removed from screen

5.2.2.7.26.220 GFX_GOL_CustomControlGetPos Macro

C

```
#define GFX_GOL_CustomControlGetPos(pCc) pCc->pos
```

Description

Macros: GFX_GOL_CustomControlGetPos(pCc)

returns position

Preconditions

none

Parameters

Parameters	Description
pCc	pointer to the object

Returns

current position

Side Effects

none

Remarks

none

5.2.2.7.26.221 GFX_GOL_CustomControlSetPos Macro

C

```
#define GFX_GOL_CustomControlSetPos(pCc, position) pCc->pos = position
```

Description

Macros: GFX_GOL_CustomControlSetPos(pCc,position)

sets the current position of the control

Preconditions

none

Parameters

Parameters	Description
pCc	the pointer to the object
position	new position

Returns

none

Remarks

none

5.2.2.7.26.222 GFX_GOL_DIGITALMETER_INDENT Macro

C

```
#define GFX_GOL_DIGITALMETER_INDENT 0x02
```

Description

This is a DigitalMeter compile time option to define the indent constant for the text used in the frame.

5.2.2.7.26.223 GFX_GOL_DIGITALMETER_WIDTH Macro

C

```
#define GFX_GOL_DIGITALMETER_WIDTH 16
```

Description

This is a DigitalMeter compile time option to define the user should change this value depending on the number of digits he wants to display.

5.2.2.7.26.224 GFX_GOL_DigitalMeterValueGet Macro

C

```
#define GFX_GOL_DigitalMeterValueGet(pObject) \
(((GFX_GOL_DIGITALMETER*)pObject)->value)
```

Description

This function returns the current value of the digital meter.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The current value of the object.

Example

```
#define MAXVALUE 100;

GFX_GOL_DIGITALMETER *pMeter;
uint32_t ctr = 0;

// create scroll bar here and initialize parameters
pMeter = GFX_GOL_DigitalMeterCreate(...)
GFX_GOL_ObjectStateSet(pMeter, GFX_GOL_DIGITALMETER_DRAW_STATE);

// draw the scroll bar
GFX_GOL_ObjectListDraw();

// a routine that updates the position of the thumb through some
// conditions
while("some condition")
{
    GFX_GOL_DigitalMeterValueSet(pMeter, ctr);
    GFX_GOL_ObjectStateSet( pMeter,
                           GFX_GOL_DIGITALMETER_UPDATE_STATE);

    // update the screen
    GFX_GOL_ObjectListDraw();

    // update ctr here
    ctr = "some source of value";
}

if (GFX_GOL_DigitalMeterValueGet(pScrollBar) > MAXVALUE)
    return 0;
else
    "do something else"
```

5.2.2.7.26.225 GFX_GOL_FONT_DEFAULT_H_FILE Macro

C

```
#define GFX_GOL_FONT_DEFAULT_H_FILE
```

5.2.2.7.26.226 GFX_GOL_GroupboxTextGet Macro

C

```
#define GFX_GOL_GroupboxTextGet(pObject) (((GFX_GOL_GROUPBOX *)pObject)->pText)
```

Description

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to text string.

Example

```
GFX_XCHAR *pChar;  
GFX_GOL_GROUPBOX pGroupbox;  
  
pChar = GFX_GOL_GroupboxTextGet(pGroupbox);
```

5.2.2.7.26.227 GFX_GOL_ListBoxItemCountGet Macro**C**

```
#define GFX_GOL_ListBoxItemCountGet(pObject) \  
(((GFX_GOL_LISTBOX *)pObject)->itemsNumber)
```

Description

This function returns the count of items in the list box.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object.

Returns

The number of items in the list box.

Example

None.

5.2.2.7.26.228 GFX_GOL_ListBoxItemImageGet Macro**C**

```
#define GFX_GOL_ListBoxItemImageGet(pItem) (((GFX_GOL_LISTITEM *)pItem)->pImage)
```

Description

This function gets the image set for a list box item.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pltem	pointer to the list box item.

Returns

Pointer to the image used for the item.

Example

See [GFX_GOL_ListBoxItemListGet\(\)](#) example.

5.2.2.7.26.229 GFX_GOL_ListBoxItemImageSet Macro**C**

```
#define GFX_GOL_ListBoxItemImageSet(pItem, pImage) (((GFX_GOL_LISTITEM *)pItem)->pImage = pImage)
```

Description

This function sets the image for a list box item.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pltem	pointer to the list box item.
pImage	pointer to the image resource.

Returns

None.

Example

See [GFX_GOL_ListBoxItemAdd\(\)](#) and [GFX_GOL_ListBoxItemListGet\(\)](#) example.

5.2.2.7.26.230 GFX_GOL_ListBoxItemListGet Macro**C**

```
#define GFX_GOL_ListBoxItemListGet(pObject) \
    (((GFX_GOL_LISTITEM *)((GFX_GOL_LISTBOX *)pObject)->pItemList)
```

Description

This function returns the pointer to the item list of the list box.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.

Returns

The pointer to the item list of the list box.

Example

```
// assume:  
// pLb is a initialized to a list box in memory  
// myIcon is a valid image in memory
```

```

GFX_GOL_LISTITEM *pList, *pItem;
GFX_RESOURCE_HDR *pMyIcon = &myIcon;

pList = GFX_GOL_ListBoxItemListGet(pLb);
pItem = pList;

// set the image for all the items to myIcon
// set the last item image to NULL

do
{
    GFX_GOL_ListBoxImageSet(pItem, &myIcon);
}
while(pItem->pNextItem != NULL);

// get the last item's image and set to null if not null
if (GFX_GOL_ListBoxItemImageGet(pItem) != NULL)
    GFX_GOL_ListBoxImageSet(pItem, NULL);

```

5.2.2.7.26.231 GFX_GOL_ListBoxItemSelectStatusClear Macro

C

```

#define GFX_GOL_ListBoxItemSelectStatusClear(pObject, pItem) \
    if(pItem->status & GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED) \
        GFX_GOL_ListBoxSelectionChange((LISTBOX *)pObject, pItem);

```

Description

This function clears the selection status of the item.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object.
pItem	The pointer the item that will have the selected status cleared.

Returns

None.

Example

None.

5.2.2.7.26.232 GFX_GOL_ListBoxItemSelectStatusSet Macro

C

```

#define GFX_GOL_ListBoxItemSelectStatusSet(pObject, pItem) \
    if(!(pItem->status & GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED)) \
        GFX_GOL_ListBoxSelectionChange((LISTBOX *)pObject, pItem);

```

Description

This function sets the selection status of the item to selected.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	The pointer to the object.
pltem	The pointer the item that will have the selected status set to selected.

Returns

None.

Example

None.

5.2.2.7.26.233 GFX_GOL_MeterMaximumValueGet Macro**C**

```
#define GFX_GOL_MeterMaximumValueGet(pObject) (((GFX_GOL_METER *)pObject)->maxValue)
```

Description

This function returns the maximum value set for the meter.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The current maximum value set for the object.

Example

None.

5.2.2.7.26.234 GFX_GOL_MeterMinimumValueGet Macro**C**

```
#define GFX_GOL_MeterMinimumValueGet(pObject) (((GFX_GOL_METER *)pObject)->minValue)
```

Description

This function returns the minimum value set for the meter.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The current minimum value set for the object.

Example

None.

5.2.2.7.26.235 GFX_GOL_MeterTitleFontSet Macro

C

```
#define GFX_GOL_MeterTitleFontSet(pObject, pNewFont) (((GFX_GOL_METER *)pObject)->pTitleFont = pNewFont)
```

Description

This function sets the meter title font used. Font pointer must be initialized to a valid [GFX_RESOURCE_HDR](#).

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pNewFont	pointer to the selected font.

Returns

None.

Example

None.

5.2.2.7.26.236 GFX_GOL_MeterTypeSet Macro

C

```
#define GFX_GOL_MeterTypeSet(pObject, type) (((GFX_GOL_METER *)pObject)->type = type)
```

Description

This function sets the meter draw type.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
type	the draw type selected (see GFX_GOL_METER_DRAW_TYPE for more information).

Returns

None.

Example

None.

5.2.2.7.26.237 GFX_GOL_MeterValueFontSet Macro

C

```
#define GFX_GOL_MeterValueFontSet(pObject, pNewFont) (((GFX_GOL_METER *)pObject)->pValueFont = pNewFont)
```


Description

This function sets the meter's displayed value font used. Font pointer must be initialized to a valid [GFX_RESOURCE_HDR](#).

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pNewFont	pointer to the selected font.

Returns

None.

Example

None.

5.2.2.7.26.238 GFX_GOL_MeterValueGet Macro**C**

```
#define GFX_GOL_MeterValueGet(pMtr) (((GFX_GOL_METER*)pMtr)->value)
```

Description

This function returns the current value of the meter.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The current value of the object.

Example

```
#define MAXVALUE 100;

GFX_GOL_METER *pMeter;
uint32_t ctr = 0;

// create scroll bar here and initialize parameters
pMeter = GFX_GOL_MeterCreate(...)
GFX_GOL_ObjectStateSet(pMeter, GFX_GOL_METER_DRAW_STATE);

// draw the scroll bar
GFX_GOL_ObjectListDraw();

// a routine that updates the position of the thumb through some
// conditions
while("some condition")
{
    GFX_GOL_MeterValueSet(pMeter, ctr);
    GFX_GOL_ObjectStateSet( pMeter,
                           GFX_GOL_METER_UPDATE_DRAW_STATE);
}
```

```

    // update the screen
    GFX_GOL_ObjectListDraw();

    // update ctr here
    ctr = "some source of value";
}

if (GFX_GOL_MeterValueGet(pScrollBar) > MAXVALUE)
    return 0;
else
    "do something else"

```

5.2.2.7.26.239 GFX_GOL_ObjectStateClear Macro

C

```

#define GFX_GOL_ObjectStateClear(pObject, stateBits) \
    (((GFX_GOL_OBJ_HEADER*)pObject)->state) &= (~(stateBits))

```

Description

This function clears the state bits of the given object. Object must be redrawn to display the changes. It is possible to set several state bits with this function.

Preconditions

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
stateBits	Defines which state bits are to be cleared. Please refer to specific objects for object state bits definition for details

Returns

GFX_STATUS_SUCCESS - is returned if the clear was successful. GFX_STATUS_FAILURE - is returned if the clear was not successful.

Example

See [GFX_GOL_ObjectStateSet\(\)](#) for code example.

5.2.2.7.26.240 GFX_GOL_ObjectStateGet Macro

C

```

#define GFX_GOL_ObjectStateGet(pObject, stateBits) \
    (((GFX_GOL_OBJ_HEADER*)pObject)->state & stateBits)

```

Description

This function retrieves the current value of the state bits of an object. It is possible to get several state bits.

Preconditions

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
stateBits	Defines which state bits are to be retrieved. Please refer to specific objects for object state bits definition for details

Returns

The current status of the specified state bits.

Example

```
#define BTN_HIDE 0x8000

GFX_GOL_BUTTON *pB;
// pB is created and initialized
// do something here to set state

// Hide the button (remove from screen)
if (GFX_GOL_ObjectStateGet(pB, GFX_GOL_BUTTON_HIDE_STATE))
{
    GFX_ColorSet(pB->pGolScheme->CommonBkColor);
    GFX_BarDraw(pB->left, pB->top, pB->right, pB->bottom);
}
```

5.2.2.7.26.241 GFX_GOL_ObjectStateSet Macro**C**

```
#define GFX_GOL_ObjectStateSet(pObject, stateBits) \
    (((GFX_GOL_OBJ_HEADER*)pObject)->state) |= stateBits
```

Description

This function sets the state bits of the given object. Object must be redrawn to display the changes. It is possible to set several state bits with this function.

Preconditions

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
stateBits	Defines which state bits are to be cleared. Please refer to specific objects for object state bits definition for details

Returns

GFX_STATUS_SUCCESS - is returned if the set was successful. GFX_STATUS_FAILURE - is returned if the set was not successful.

Example

```
void SetMessage(uint16_t msg, GFX_GOL_BUTTON* pB)
{
    switch(msg)
    {
        case GFX_GOL_BUTTON_ACTION_PRESSED:
            // set pressed and redraw
            GFX_GOL_ObjectStateSet(pB, BTN_PRESSED|BTN_DRAW);
            break;
        case GFX_GOL_BUTTON_ACTION_RELEASED:
            // reset pressed
            GFX_GOL_ObjectStateClear(pB, BTN_PRESSED);
            // redraw
            GFX_GOL_ObjectStateSet(pB, BTN_DRAW);
            break;
        default:
            break;
    }
}
```

5.2.2.7.26.242 GFX_GOL_ProgressBarPositionGet Macro

C

```
#define GFX_GOL_ProgressBarPositionGet(pObject) (((GFX_GOL_PROGRESSBAR*)pObject)->pos)
```

Description

This function returns the current position of the progress bar.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The current position of the scroll bar thumb.

Example

See [GFX_GOL_ProgressBarPositionSet\(\)](#) example.

5.2.2.7.26.243 GFX_GOL_ProgressBarRangeGet Macro

C

```
#define GFX_GOL_ProgressBarRangeGet(pObject) (pObject->range)
```

Description

This function returns the range of the progress bar.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The current range used by the object.

Example

None.

5.2.2.7.26.244 GFX_GOL_RadioButtonTextGet Macro

C

```
#define GFX_GOL_RadioButtonTextGet(pObject) (((GFX_GOL_RADIOBUTTON *)pObject)->pText)
```

Description

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to text string.

Example

```
// assume RADIO_BUTTON_OBJECT is a radio button that exists
GFX_XCHAR *pChar;
GFX_GOL_RADIOBUTTON *pRadioButton = &RADIO_BUTTON_OBJECT;

pChar = GFX_GOL_ButtonRadioTextGet(pRadioButton);
```

5.2.2.7.26.245 GFX_GOL_ROUNDIAL_DISABLED Macro

C

```
#define GFX_GOL_ROUNDIAL_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

5.2.2.7.26.246 GFX_GOL_ROUNDIAL_DRAW Macro

C

```
#define GFX_GOL_ROUNDIAL_DRAW 0x4000 // Bit to indicate object must be redrawn.
```

Description

Bit to indicate object must be redrawn.

5.2.2.7.26.247 GFX_GOL_ROUNDIAL_HIDE Macro

C

```
#define GFX_GOL_ROUNDIAL_HIDE 0x8000 // Bit to indicate object must be removed from screen.
```

Description

Bit to indicate object must be removed from screen.

5.2.2.7.26.248 GFX_GOL_ROUNDIAL_MAX_POSITIONS Macro

C

```
#define GFX_GOL_ROUNDIAL_MAX_POSITIONS 24
```

Description

This is macro GFX_GOL_ROUNDIAL_MAX_POSITIONS.

5.2.2.7.26.249 GFX_GOL_ROUNDIAL_QUADRANT_POSITIONS Macro

C

```
#define GFX_GOL_ROUNDIAL_QUADRANT_POSITIONS 6
```

Description

This is macro GFX_GOL_ROUNDIAL_QUADRANT_POSITIONS.

5.2.2.7.26.250 GFX_GOL_ROUNDIAL_ROT_CCW Macro

C

```
#define GFX_GOL_ROUNDIAL_ROT_CCW 0x0008 // Bit for rotate counter clockwise state.
```

Description

Bit for rotate counter clockwise state.

5.2.2.7.26.251 GFX_GOL_ROUNDIAL_ROT_CW Macro

C

```
#define GFX_GOL_ROUNDIAL_ROT_CW 0x0004 // Bit for rotate clockwise state.
```

Description

Bit for rotate clockwise state.

5.2.2.7.26.252 GFX_GOL_RoundDialValueDecrement Macro

C

```
#define GFX_GOL_RoundDialValueDecrement(pDia) GFX_GOL_RoundDial_ValSet(pDia, (pDia->pos - pDia->res))
```

Description

Macros: GFX_GOL_RoundDialValueDecrement(pDia)

Preconditions

none

Parameters

Parameters	Description
pDia	Pointer to the object.

Returns

none

Remarks

none

Example

Refer to [GFX_GOL_RoundDialValueIncrement\(\)](#) example.

5.2.2.7.26.253 GFX_GOL_RoundDialValueGet Macro

C

```
#define GFX_GOL_RoundDialValueGet(pDia) (pDia->value
```

Description

Macros: GFX_GOL_RoundDialValueGet(pDia)

Preconditions

none

Parameters

Parameters	Description
pDia	Pointer to the object.

Returns

Returns the current value of the dial.

Remarks

none

Example

```
uint16_t currVal;
GFX_GOL_ROUNDIAL *pDia;

// assuming pDia is initialized to an existing dial Object
currVal = GFX_GOL_RoundDial_ValGet(pDia);
```

5.2.2.7.26.254 GFX_GOL_RoundDialValueIncrement Macro**C**

```
#define GFX_GOL_RoundDialValueIncrement(pDia) GFX_GOL_RoundDial_ValSet(pDia, (pDia->val + pDia->res))
```

Description

Macros: GFX_GOL_RoundDialValueIncrement(pDia)

Preconditions

none

Parameters

Parameters	Description
pDia	Pointer to the object.

Returns

none

none

Example

```
uint16_t updatedVal, prevVal;
GFX_GOL_ROUNDIAL *pDia;

// assuming pDia is initialized to an existing dial Object
// assume GetInput() is a function that retrieves source data
prevVal = GFX_GOL_RoundDial_ValGet(pDia);
updatedVal = GetInput();
if (updatedVal > prevVal)
    GFX_GOL_RoundDial_ValInc(pDia);
if (updatedVal < prevVal)
    GFX_GOL_RoundDial_ValDec(pDia);
```

5.2.2.7.26.255 GFX_GOL_RoundDialValueSet Macro**C**

```
#define GFX_GOL_RoundDialValueSet(pDia, newVal) (pDia)->value = newVal
```

Description

Macros: GFX_GOL_RoundDialValueSet(pDia, newVal)

Preconditions

none

Parameters

Parameters	Description
pDia	Pointer to the object.
newVal	New dial value.

Returns

none

Remarks

none

Example

```
uint16_t updatedVal;
GFX_GOL_ROUNDIAL *pDia;

// assuming pDia is initialized to an existing dial Object
// assume GetInput() is a function that retrieves source data
updatedVal = GetInput();
GFX_GOL_RoundDialValueSet(pDia, updatedVal);
```

5.2.2.7.26.256 GFX_GOL_WindowImageGet Macro**C**

```
#define GFX_GOL_WindowImageGet(pObject, pImage) \
(((GFX_GOL_WINDOW *)pObject)->pImage)
```

Description

This function gets the image used.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to the image resource.

Example

None.

5.2.2.7.26.257 GFX_GOL_WindowTextGet Macro**C**

```
#define GFX_GOL_WindowTextGet(pObject) (((GFX_GOL_WINDOW *)pObject)->pText)
```


Description

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to text string.

Example

```
GFX_XCHAR *pChar;  
GFX_GOL_WINDOW pWindow;  
  
pChar = GFX_GOL_WindowTextGet(pWindow);
```

5.2.2.7.26.258 GFX_ImageStretchSet Macro**C**

```
#define GFX_ImageStretchSet(value) primitiveTaskImage.stretch = value
```

Description

This is macro GFX_ImageStretchSet.

5.2.2.7.26.259 GFX_MAX_INVALIDATE_AREAS Macro**C**

```
#define GFX_MAX_INVALIDATE_AREAS 5
```

Description

This is macro GFX_MAX_INVALIDATE_AREAS.

5.2.2.7.26.260 GFX_MaxXGet Macro**C**

```
#define GFX_MaxXGet (GFX_DRV_instance[0].horizontalResolution-1)
```

Example

Remarks:

5.2.2.7.26.261 GFX_MaxYGet Macro**C**

```
#define GFX_MaxYGet (GFX_DRV_instance[0].verticalResolution-1)
```

Example

Remarks:

5.2.2.7.26.262 GFX_MEM_MASK Macro

C

```
#define GFX_MEM_MASK 0x000F
```

Description

This is macro GFX_MEM_MASK.

5.2.2.7.26.263 GFX_PixelPut Macro

C

```
#define GFX_PixelPut(x, y) GFX_PixelsPut(x, y, 1, 1)
```

Description

This is macro GFX_PixelPut.

5.2.2.7.26.264 GFX_PixelsPut Macro

C

```
#define GFX_PixelsPut(x, y, count, lineCount) GFX_RectangleFillDraw(x, y, x+count, y+lineCount)
```

Description

None

Preconditions

None

Parameters

Parameters	Description
instance	primitive instance
x	x position of the left top corner.
y	y position of the left top corner.
count	number of pixels

Returns

Queue handle once the primitive has been place in queue

Example

None

5.2.2.7.26.265 GFX_PrevAlphaColorGet Macro

C

```
#define GFX_PrevAlphaColorGet GFX_Primitive_instance.alphaArea.prevAlphaColor
```

Example

Remarks:

5.2.2.7.26.266 GFX_PrevAlphaColorSet Macro

C

```
#define GFX_PrevAlphaColorSet(color) GFX_Primitive_instance.alphaArea.prevAlphaColor = color
```

Example

Remarks:

5.2.2.7.26.267 GFX_RGBConvert Macro**C**

```
#define GFX_RGBConvert(red, green, blue) (GFX_COLOR) (((GFX_COLOR)(red) << 16) |  
((GFX_COLOR)(green) << 8) | (GFX_COLOR)(blue))
```

Description

Macros: GFX_RGBConvert(red, green, blue)

This macro converts the color data to the selected **COLOR_DEPTH**. This macro is only valid when **COLOR_DEPTH** is 8, 16, or 24.

COLOR_DEPTH	Conversion
8	8-8-8 to 3-3-2 conversion
16	8-8-8 to 5-6-5 conversion
24	8-8-8 to 8-8-8 conversion or no conversion

Preconditions

none

Parameters

Parameters	Description
red	red component of the color.
green	green component of the color.
blue	blue component of the color.

Returns

none

Side Effects

none

5.2.2.7.26.268 GFX_SetFontOrientation Macro**C**

```
#define GFX_SetFontOrientation(orient) GFX_Primitive_instance.currentFont.orientation = orient;
```

Preconditions

None.

Parameters

Parameters	Description
orient	sets font orientation when rendering characters and strings.

Returns

None.

Example

5.2.2.7.26.269 GFX_TYPE_MASK Macro

C

```
#define GFX_TYPE_MASK 0x0F00
```

Description

This is macro GFX_TYPE_MASK.

5.2.2.7.26.270 GFX_TYPES_FONTS_H Macro

C

```
#define GFX_TYPES_FONTS_H
```

Description

This is macro GFX_TYPES_FONTS_H.

5.2.2.7.26.271 GFX_TYPES_IMAGE_H Macro

C

```
#define GFX_TYPES_IMAGE_H
```

Description

This is macro GFX_TYPES_IMAGE_H.

5.2.2.7.26.272 GFX_TYPES_PALETTE_H Macro

C

```
#define GFX_TYPES_PALETTE_H
```

Description

This is macro GFX_TYPES_PALETTE_H.

5.2.2.7.26.273 GFX_TYPES_RESOURCE_H Macro

C

```
#define GFX_TYPES_RESOURCE_H
```

Description

This is macro GFX_TYPES_RESOURCE_H.

5.2.2.7.26.274 GFX_UXCHAR Macro

C

```
#define GFX_UXCHAR uint8_t
```

Description

This is macro GFX_UXCHAR.

5.2.2.7.26.275 GFX_VisualPageGet Macro

C

```
#define GFX_VisualPageGet GFX_Primitive_instance.visualPage
```

Preconditions

None.

Returns

None.

Example

Remarks:

5.2.2.7.26.276 GFX_W3_ALICEBLUE Macro

C

```
#define GFX_W3_ALICEBLUE GFX_RGBConvert(240, 248, 255)
```

Description

This is macro GFX_W3_ALICEBLUE.

5.2.2.7.26.277 GFX_W3_ANTIQUWHITE Macro

C

```
#define GFX_W3_ANTIQUWHITE GFX_RGBConvert(250, 235, 215)
```

Description

This is macro GFX_W3_ANTIQUWHITE.

5.2.2.7.26.278 GFX_W3_AQUA Macro

C

```
#define GFX_W3_AQUA GFX_RGBConvert( 0, 255, 255)
```

Description

This is macro GFX_W3_AQUA.

5.2.2.7.26.279 GFX_W3_AQUAMARINE Macro

C

```
#define GFX_W3_AQUAMARINE GFX_RGBConvert(127, 255, 212)
```

Description

This is macro GFX_W3_AQUAMARINE.

5.2.2.7.26.280 GFX_W3_AZURE Macro

C

```
#define GFX_W3_AZURE GFX_RGBConvert(240, 255, 255)
```

Description

This is macro GFX_W3_AZURE.

5.2.2.7.26.281 GFX_W3_BEIGE Macro

C

```
#define GFX_W3_BEIGE GFX_RGBConvert(245, 245, 220)
```

Description

This is macro GFX_W3_BEIGE.

5.2.2.7.26.282 GFX_W3_BISQUE Macro

C

```
#define GFX_W3_BISQUE GFX_RGBConvert(255, 228, 196)
```

Description

This is macro GFX_W3_BISQUE.

5.2.2.7.26.283 GFX_W3_BLACK Macro

C

```
#define GFX_W3_BLACK GFX_RGBConvert( 0, 0, 0)
```

Description

This is macro GFX_W3_BLACK.

5.2.2.7.26.284 GFX_W3_BLANCHEDALMOND Macro

C

```
#define GFX_W3_BLANCHEDALMOND GFX_RGBConvert(255, 235, 205)
```

Description

This is macro GFX_W3_BLANCHEDALMOND.

5.2.2.7.26.285 GFX_W3_BLUE Macro

C

```
#define GFX_W3_BLUE GFX_RGBConvert( 0, 0, 255)
```

Description

This is macro GFX_W3_BLUE.

5.2.2.7.26.286 GFX_W3_BLUEVIOLET Macro

C

```
#define GFX_W3_BLUEVIOLET GFX_RGBConvert(138, 43, 226)
```

Description

This is macro GFX_W3_BLUEVIOLET.

5.2.2.7.26.287 GFX_W3_BROWN Macro

C

```
#define GFX_W3_BROWN GFX_RGBConvert(165, 42, 42)
```

Description

This is macro GFX_W3_BROWN.

5.2.2.7.26.288 GFX_W3_BURLYWOOD Macro

C

```
#define GFX_W3_BURLYWOOD GFX_RGBConvert(222, 184, 135)
```

Description

This is macro GFX_W3_BURLYWOOD.

5.2.2.7.26.289 GFX_W3_CADETBLUE Macro

C

```
#define GFX_W3_CADETBLUE GFX_RGBConvert( 95, 158, 160)
```

Description

This is macro GFX_W3_CADETBLUE.

5.2.2.7.26.290 GFX_W3_CHARTREUSE Macro

C

```
#define GFX_W3_CHARTREUSE GFX_RGBConvert(127, 255, 0)
```

Description

This is macro GFX_W3_CHARTREUSE.

5.2.2.7.26.291 GFX_W3_CHOCOLATE Macro

C

```
#define GFX_W3_CHOCOLATE GFX_RGBConvert(210, 105, 30)
```

Description

This is macro GFX_W3_CHOCOLATE.

5.2.2.7.26.292 GFX_W3_CORAL Macro

C

```
#define GFX_W3_CORAL GFX_RGBConvert(255, 127, 80)
```

Description

This is macro GFX_W3_CORAL.

5.2.2.7.26.293 GFX_W3_CORNFLOWERBLUE Macro

C

```
#define GFX_W3_CORNFLOWERBLUE GFX_RGBConvert(100, 149, 237)
```

Description

This is macro GFX_W3_CORNFLOWERBLUE.

5.2.2.7.26.294 GFX_W3_CORNSILK Macro

C

```
#define GFX_W3_CORNSILK GFX_RGBConvert(255, 248, 220)
```

Description

This is macro GFX_W3_CORNSILK.

5.2.2.7.26.295 GFX_W3_CRIMSON Macro

C

```
#define GFX_W3_CRIMSON GFX_RGBConvert(220, 20, 60)
```

Description

This is macro GFX_W3_CRIMSON.

5.2.2.7.26.296 GFX_W3_CYAN Macro

C

```
#define GFX_W3_CYAN GFX_RGBConvert( 0, 255, 255)
```

Description

This is macro GFX_W3_CYAN.

5.2.2.7.26.297 GFX_W3_DARKBLUE Macro

C

```
#define GFX_W3_DARKBLUE GFX_RGBConvert( 0, 0, 139)
```

Description

This is macro GFX_W3_DARKBLUE.

5.2.2.7.26.298 GFX_W3_DARKCYAN Macro

C

```
#define GFX_W3_DARKCYAN GFX_RGBConvert( 0, 139, 139)
```

Description

This is macro GFX_W3_DARKCYAN.

5.2.2.7.26.299 GFX_W3_darkgoldenrod Macro

C

```
#define GFX_W3_darkgoldenrod GFX_RGBConvert(184, 134, 11)
```

Description

This is macro GFX_W3_darkgoldenrod.

5.2.2.7.26.300 GFX_W3_DARKGRAY Macro

C

```
#define GFX_W3_DARKGRAY GFX_RGBConvert(169, 169, 169)
```


Description

This is macro GFX_W3_DARKGRAY.

5.2.2.7.26.301 GFX_W3_DARKGREEN Macro

C

```
#define GFX_W3_DARKGREEN GFX_RGBConvert( 0, 100, 0)
```

Description

This is macro GFX_W3_DARKGREEN.

5.2.2.7.26.302 GFX_W3_DARKGREY Macro

C

```
#define GFX_W3_DARKGREY GFX_RGBConvert(169, 169, 169)
```

Description

This is macro GFX_W3_DARKGREY.

5.2.2.7.26.303 GFX_W3_DARKHAKI Macro

C

```
#define GFX_W3_DARKHAKI GFX_RGBConvert(189, 183, 107)
```

Description

This is macro GFX_W3_DARKHAKI.

5.2.2.7.26.304 GFX_W3_DARKMAGENTA Macro

C

```
#define GFX_W3_DARKMAGENTA GFX_RGBConvert(139, 0, 139)
```

Description

This is macro GFX_W3_DARKMAGENTA.

5.2.2.7.26.305 GFX_W3_DARKOLIVEGREEN Macro

C

```
#define GFX_W3_DARKOLIVEGREEN GFX_RGBConvert( 85, 107, 47)
```

Description

This is macro GFX_W3_DARKOLIVEGREEN.

5.2.2.7.26.306 GFX_W3_DARKORANGE Macro

C

```
#define GFX_W3_DARKORANGE GFX_RGBConvert(255, 140, 0)
```

Description

This is macro GFX_W3_DARKORANGE.

5.2.2.7.26.307 GFX_W3_DARKORCHID Macro

C

```
#define GFX_W3_DARKORCHID GFX_RGBConvert(153, 50, 204)
```

Description

This is macro GFX_W3_DARKORCHID.

5.2.2.7.26.308 GFX_W3_DARKRED Macro

C

```
#define GFX_W3_DARKRED GFX_RGBConvert(139, 0, 0)
```

Description

This is macro GFX_W3_DARKRED.

5.2.2.7.26.309 GFX_W3_DARKSALMON Macro

C

```
#define GFX_W3_DARKSALMON GFX_RGBConvert(233, 150, 122)
```

Description

This is macro GFX_W3_DARKSALMON.

5.2.2.7.26.310 GFX_W3_DARKSEAGREEN Macro

C

```
#define GFX_W3_DARKSEAGREEN GFX_RGBConvert(143, 188, 143)
```

Description

This is macro GFX_W3_DARKSEAGREEN.

5.2.2.7.26.311 GFX_W3_DARKSLATEBLUE Macro

C

```
#define GFX_W3_DARKSLATEBLUE GFX_RGBConvert( 72, 61, 139)
```

Description

This is macro GFX_W3_DARKSLATEBLUE.

5.2.2.7.26.312 GFX_W3_DARKSLATEGRAY Macro

C

```
#define GFX_W3_DARKSLATEGRAY GFX_RGBConvert( 47, 79, 79)
```

Description

This is macro GFX_W3_DARKSLATEGRAY.

5.2.2.7.26.313 GFX_W3_DARKSLATEGREY Macro

C

```
#define GFX_W3_DARKSLATEGREY GFX_RGBConvert( 47, 79, 79)
```

Description

This is macro GFX_W3_DARKSLATEGREY.

5.2.2.7.26.314 GFX_W3_DARKTURQUOISE Macro

C

```
#define GFX_W3_DARKTURQUOISE GFX_RGBConvert( 0, 206, 209)
```

Description

This is macro GFX_W3_DARKTURQUOISE.

5.2.2.7.26.315 GFX_W3_DARKVIOLET Macro

C

```
#define GFX_W3_DARKVIOLET GFX_RGBConvert(148, 0, 211)
```

Description

This is macro GFX_W3_DARKVIOLET.

5.2.2.7.26.316 GFX_W3_DEEPPINK Macro

C

```
#define GFX_W3_DEEPPINK GFX_RGBConvert(255, 20, 147)
```

Description

This is macro GFX_W3_DEEPPINK.

5.2.2.7.26.317 GFX_W3_DEEPSKYBLUE Macro

C

```
#define GFX_W3_DEEPSKYBLUE GFX_RGBConvert( 0, 191, 255)
```

Description

This is macro GFX_W3_DEEPSKYBLUE.

5.2.2.7.26.318 GFX_W3_DIMGRAY Macro

C

```
#define GFX_W3_DIMGRAY GFX_RGBConvert(105, 105, 105)
```

Description

This is macro GFX_W3_DIMGRAY.

5.2.2.7.26.319 GFX_W3_DIMGREY Macro

C

```
#define GFX_W3_DIMGREY GFX_RGBConvert(105, 105, 105)
```

Description

This is macro GFX_W3_DIMGREY.

5.2.2.7.26.320 GFX_W3_DODGERBLUE Macro

C

```
#define GFX_W3_DODGERBLUE GFX_RGBConvert( 30, 144, 255)
```

Description

This is macro GFX_W3_DODGERBLUE.

5.2.2.7.26.321 GFX_W3_FIREBRICK Macro

C

```
#define GFX_W3_FIREBRICK GFX_RGBConvert(178, 34, 34)
```

Description

This is macro GFX_W3_FIREBRICK.

5.2.2.7.26.322 GFX_W3_FLORALWHITE Macro

C

```
#define GFX_W3_FLORALWHITE GFX_RGBConvert(255, 250, 240)
```

Description

This is macro GFX_W3_FLORALWHITE.

5.2.2.7.26.323 GFX_W3_FORESTGREEN Macro

C

```
#define GFX_W3_FORESTGREEN GFX_RGBConvert( 34, 139, 34)
```

Description

This is macro GFX_W3_FORESTGREEN.

5.2.2.7.26.324 GFX_W3_FUCHSIA Macro

C

```
#define GFX_W3_FUCHSIA GFX_RGBConvert(255, 0, 255)
```

Description

This is macro GFX_W3_FUCHSIA.

5.2.2.7.26.325 GFX_W3_GAINSBORO Macro

C

```
#define GFX_W3_GAINSBORO GFX_RGBConvert(220, 220, 220)
```

Description

This is macro GFX_W3_GAINSBORO.

5.2.2.7.26.326 GFX_W3_GHOSTWHITE Macro

C

```
#define GFX_W3_GHOSTWHITE GFX_RGBConvert(248, 248, 255)
```

Description

This is macro GFX_W3_GHOSTWHITE.

5.2.2.7.26.327 GFX_W3_GOLD Macro

C

```
#define GFX_W3_GOLD GFX_RGBConvert(255, 215, 0)
```

Description

This is macro GFX_W3_GOLD.

5.2.2.7.26.328 GFX_W3_GOLDENROD Macro

C

```
#define GFX_W3_GOLDENROD GFX_RGBConvert(218, 165, 32)
```

Description

This is macro GFX_W3_GOLDENROD.

5.2.2.7.26.329 GFX_W3_GRAY Macro

C

```
#define GFX_W3_GRAY GFX_RGBConvert(128, 128, 128)
```

Description

This is macro GFX_W3_GRAY.

5.2.2.7.26.330 GFX_W3_GREEN Macro

C

```
#define GFX_W3_GREEN GFX_RGBConvert( 0, 128, 0)
```

Description

This is macro GFX_W3_GREEN.

5.2.2.7.26.331 GFX_W3_GREENYELLOW Macro

C

```
#define GFX_W3_GREENYELLOW GFX_RGBConvert(173, 255, 47)
```

Description

This is macro GFX_W3_GREENYELLOW.

5.2.2.7.26.332 GFX_W3_GREY Macro

C

```
#define GFX_W3_GREY GFX_RGBConvert(128, 128, 128)
```

Description

This is macro GFX_W3_GREY.

5.2.2.7.26.333 GFX_W3_HONEYDEW Macro

C

```
#define GFX_W3_HONEYDEW GFX_RGBConvert(240, 255, 240)
```

Description

This is macro GFX_W3_HONEYDEW.

5.2.2.7.26.334 GFX_W3_HOTPINK Macro

C

```
#define GFX_W3_HOTPINK GFX_RGBConvert(255, 105, 180)
```

Description

This is macro GFX_W3_HOTPINK.

5.2.2.7.26.335 GFX_W3_INDIANRED Macro

C

```
#define GFX_W3_INDIANRED GFX_RGBConvert(205, 92, 92)
```

Description

This is macro GFX_W3_INDIANRED.

5.2.2.7.26.336 GFX_W3_INDIGO Macro

C

```
#define GFX_W3_INDIGO GFX_RGBConvert( 75, 0, 130)
```

Description

This is macro GFX_W3_INDIGO.

5.2.2.7.26.337 GFX_W3_IVORY Macro

C

```
#define GFX_W3_IVORY GFX_RGBConvert(255, 255, 240)
```

Description

This is macro GFX_W3_IVORY.

5.2.2.7.26.338 GFX_W3_LAVENDER Macro

C

```
#define GFX_W3_LAVENDER GFX_RGBConvert(230, 230, 250)
```

Description

This is macro GFX_W3_LAVENDER.

5.2.2.7.26.339 GFX_W3_LAVENDERBLUSH Macro

C

```
#define GFX_W3_LAVENDERBLUSH GFX_RGBConvert(255, 240, 245)
```

Description

This is macro GFX_W3_LAVENDERBLUSH.

5.2.2.7.26.340 GFX_W3_LAWNGREEN Macro

C

```
#define GFX_W3_LAWNGREEN GFX_RGBConvert(124, 252, 0)
```

Description

This is macro GFX_W3_LAWNGREEN.

5.2.2.7.26.341 GFX_W3_LEMONCHIFFON Macro

C

```
#define GFX_W3_LEMONCHIFFON GFX_RGBConvert(255, 250, 205)
```

Description

This is macro GFX_W3_LEMONCHIFFON.

5.2.2.7.26.342 GFX_W3_LIGHTBLUE Macro

C

```
#define GFX_W3_LIGHTBLUE GFX_RGBConvert(173, 216, 230)
```

Description

This is macro GFX_W3_LIGHTBLUE.

5.2.2.7.26.343 GFX_W3_LIGHTCORAL Macro

C

```
#define GFX_W3_LIGHTCORAL GFX_RGBConvert(240, 128, 128)
```

Description

This is macro GFX_W3_LIGHTCORAL.

5.2.2.7.26.344 GFX_W3_LIGHTCYAN Macro

C

```
#define GFX_W3_LIGHTCYAN GFX_RGBConvert(224, 255, 255)
```

Description

This is macro GFX_W3_LIGHTCYAN.

5.2.2.7.26.345 GFX_W3_LIGHTGOLDENRODYELLOW Macro

C

```
#define GFX_W3_LIGHTGOLDENRODYELLOW GFX_RGBConvert(250, 250, 210)
```

Description

This is macro GFX_W3_LIGHTGOLDENRODYELLOW.

5.2.2.7.26.346 GFX_W3_LIGHTGRAY Macro

C

```
#define GFX_W3_LIGHTGRAY GFX_RGBConvert(211, 211, 211)
```

Description

This is macro GFX_W3_LIGHTGRAY.

5.2.2.7.26.347 GFX_W3_LIGHTGREEN Macro

C

```
#define GFX_W3_LIGHTGREEN GFX_RGBConvert(144, 238, 144)
```

Description

This is macro GFX_W3_LIGHTGREEN.

5.2.2.7.26.348 GFX_W3_LIGHTGREY Macro

C

```
#define GFX_W3_LIGHTGREY GFX_RGBConvert(211, 211, 211)
```

Description

This is macro GFX_W3_LIGHTGREY.

5.2.2.7.26.349 GFX_W3_LIGHTPINK Macro

C

```
#define GFX_W3_LIGHTPINK GFX_RGBConvert(255, 182, 193)
```

Description

This is macro GFX_W3_LIGHTPINK.

5.2.2.7.26.350 GFX_W3_LIGHTSALMON Macro

C

```
#define GFX_W3_LIGHTSALMON GFX_RGBConvert(255, 160, 122)
```

Description

This is macro GFX_W3_LIGHTSALMON.

5.2.2.7.26.351 GFX_W3_LIGHTSKYBLUE Macro

C

```
#define GFX_W3_LIGHTSKYBLUE GFX_RGBConvert(135, 206, 250)
```

Description

This is macro GFX_W3_LIGHTSKYBLUE.

5.2.2.7.26.352 GFX_W3_LIGHTSLATEGRAY Macro

C

```
#define GFX_W3_LIGHTSLATEGRAY GFX_RGBConvert(119, 136, 153)
```


Description

This is macro GFX_W3_LIGHTSLATEGRAY.

5.2.2.7.26.353 GFX_W3_LIGHTSLATEGREY Macro**C**

```
#define GFX_W3_LIGHTSLATEGREY GFX_RGBConvert(119, 136, 153)
```

Description

This is macro GFX_W3_LIGHTSLATEGREY.

5.2.2.7.26.354 GFX_W3_LIGHTSTEELBLUE Macro**C**

```
#define GFX_W3_LIGHTSTEELBLUE GFX_RGBConvert(176, 196, 222)
```

Description

This is macro GFX_W3_LIGHTSTEELBLUE.

5.2.2.7.26.355 GFX_W3_LIGHTYELLOW Macro**C**

```
#define GFX_W3_LIGHTYELLOW GFX_RGBConvert(255, 255, 224)
```

Description

This is macro GFX_W3_LIGHTYELLOW.

5.2.2.7.26.356 GFX_W3_LIGTHSEAGREEN Macro**C**

```
#define GFX_W3_LIGTHSEAGREEN GFX_RGBConvert( 32, 178, 170)
```

Description

This is macro GFX_W3_LIGTHSEAGREEN.

5.2.2.7.26.357 GFX_W3_LIME Macro**C**

```
#define GFX_W3_LIME GFX_RGBConvert( 0, 255, 0)
```

Description

This is macro GFX_W3_LIME.

5.2.2.7.26.358 GFX_W3_LIMEGREEN Macro**C**

```
#define GFX_W3_LIMEGREEN GFX_RGBConvert( 50, 205, 50)
```

Description

This is macro GFX_W3_LIMEGREEN.

5.2.2.7.26.359 GFX_W3_LINEN Macro

C

```
#define GFX_W3_LINEN GFX_RGBConvert(250, 240, 230)
```

Description

This is macro GFX_W3_LINEN.

5.2.2.7.26.360 GFX_W3_MAGENTA Macro

C

```
#define GFX_W3_MAGENTA GFX_RGBConvert(255, 0, 255)
```

Description

This is macro GFX_W3_MAGENTA.

5.2.2.7.26.361 GFX_W3_MAROON Macro

C

```
#define GFX_W3_MAROON GFX_RGBConvert(128, 0, 0)
```

Description

This is macro GFX_W3_MAROON.

5.2.2.7.26.362 GFX_W3_MEDIUMAQUAMARINE Macro

C

```
#define GFX_W3_MEDIUMAQUAMARINE GFX_RGBConvert(102, 205, 170)
```

Description

This is macro GFX_W3_MEDIUMAQUAMARINE.

5.2.2.7.26.363 GFX_W3_MEDIUMBLUE Macro

C

```
#define GFX_W3_MEDIUMBLUE GFX_RGBConvert( 0, 0, 205)
```

Description

This is macro GFX_W3_MEDIUMBLUE.

5.2.2.7.26.364 GFX_W3_MEDIUMORCHID Macro

C

```
#define GFX_W3_MEDIUMORCHID GFX_RGBConvert(186, 85, 211)
```

Description

This is macro GFX_W3_MEDIUMORCHID.

5.2.2.7.26.365 GFX_W3_MEDIUMPURPLE Macro

C

```
#define GFX_W3_MEDIUMPURPLE GFX_RGBConvert(147, 112, 219)
```

Description

This is macro GFX_W3_MEDIUMPURPLE.

5.2.2.7.26.366 GFX_W3_MEDIUMSEAGREEN Macro**C**

```
#define GFX_W3_MEDIUMSEAGREEN GFX_RGBConvert( 60, 179, 113)
```

Description

This is macro GFX_W3_MEDIUMSEAGREEN.

5.2.2.7.26.367 GFX_W3_MEDIUMSLATEBLUE Macro**C**

```
#define GFX_W3_MEDIUMSLATEBLUE GFX_RGBConvert(123, 104, 238)
```

Description

This is macro GFX_W3_MEDIUMSLATEBLUE.

5.2.2.7.26.368 GFX_W3_MEDIUMSPRINGGREEN Macro**C**

```
#define GFX_W3_MEDIUMSPRINGGREEN GFX_RGBConvert( 0, 250, 154)
```

Description

This is macro GFX_W3_MEDIUMSPRINGGREEN.

5.2.2.7.26.369 GFX_W3_MEDIUMTURQUOISE Macro**C**

```
#define GFX_W3_MEDIUMTURQUOISE GFX_RGBConvert( 72, 209, 204)
```

Description

This is macro GFX_W3_MEDIUMTURQUOISE.

5.2.2.7.26.370 GFX_W3_MEDIUMVIOLETRED Macro**C**

```
#define GFX_W3_MEDIUMVIOLETRED GFX_RGBConvert(199, 21, 133)
```

Description

This is macro GFX_W3_MEDIUMVIOLETRED.

5.2.2.7.26.371 GFX_W3_MIDNIGHTBLUE Macro**C**

```
#define GFX_W3_MIDNIGHTBLUE GFX_RGBConvert( 25, 25, 112)
```

Description

This is macro GFX_W3_MIDNIGHTBLUE.

5.2.2.7.26.372 GFX_W3_MINTCREAM Macro

C

```
#define GFX_W3_MINTCREAM GFX_RGBConvert(245, 255, 250)
```

Description

This is macro GFX_W3_MINTCREAM.

5.2.2.7.26.373 GFX_W3_MISTYROSE Macro

C

```
#define GFX_W3_MISTYROSE GFX_RGBConvert(255, 228, 225)
```

Description

This is macro GFX_W3_MISTYROSE.

5.2.2.7.26.374 GFX_W3_MOCCASIN Macro

C

```
#define GFX_W3_MOCCASIN GFX_RGBConvert(255, 228, 181)
```

Description

This is macro GFX_W3_MOCCASIN.

5.2.2.7.26.375 GFX_W3_NAVAJOWHITE Macro

C

```
#define GFX_W3_NAVAJOWHITE GFX_RGBConvert(255, 222, 173)
```

Description

This is macro GFX_W3_NAVAJOWHITE.

5.2.2.7.26.376 GFX_W3_NAVY Macro

C

```
#define GFX_W3_NAVY GFX_RGBConvert( 0, 0, 128)
```

Description

This is macro GFX_W3_NAVY.

5.2.2.7.26.377 GFX_W3_OLDLACE Macro

C

```
#define GFX_W3_OLDLACE GFX_RGBConvert(253, 245, 230)
```

Description

This is macro GFX_W3_OLDLACE.

5.2.2.7.26.378 GFX_W3_OLIVE Macro

C

```
#define GFX_W3_OLIVE GFX_RGBConvert(128, 128, 0)
```

Description

This is macro GFX_W3_OLIVE.

5.2.2.7.26.379 GFX_W3_OLIVEDRAB Macro

C

```
#define GFX_W3_OLIVEDRAB GFX_RGBConvert(107, 142, 35)
```

Description

This is macro GFX_W3_OLIVEDRAB.

5.2.2.7.26.380 GFX_W3_ORANGE Macro

C

```
#define GFX_W3_ORANGE GFX_RGBConvert(255, 165, 0)
```

Description

This is macro GFX_W3_ORANGE.

5.2.2.7.26.381 GFX_W3_ORANGERED Macro

C

```
#define GFX_W3_ORANGERED GFX_RGBConvert(255, 69, 0)
```

Description

This is macro GFX_W3_ORANGERED.

5.2.2.7.26.382 GFX_W3_ORCHID Macro

C

```
#define GFX_W3_ORCHID GFX_RGBConvert(218, 112, 214)
```

Description

This is macro GFX_W3_ORCHID.

5.2.2.7.26.383 GFX_W3_PALEGOLDENROD Macro

C

```
#define GFX_W3_PALEGOLDENROD GFX_RGBConvert(238, 232, 170)
```

Description

This is macro GFX_W3_PALEGOLDENROD.

5.2.2.7.26.384 GFX_W3_PALEGREEN Macro

C

```
#define GFX_W3_PALEGREEN GFX_RGBConvert(152, 251, 152)
```

Description

This is macro GFX_W3_PALEGREEN.

5.2.2.7.26.385 GFX_W3_PALETURQUOISE Macro

C

```
#define GFX_W3_PALETURQUOISE GFX_RGBConvert(175, 238, 238)
```

Description

This is macro GFX_W3_PALETURQUOISE.

5.2.2.7.26.386 GFX_W3_PALEVIOLETRED Macro

C

```
#define GFX_W3_PALEVIOLETRED GFX_RGBConvert(219, 112, 147)
```

Description

This is macro GFX_W3_PALEVIOLETRED.

5.2.2.7.26.387 GFX_W3_PAPAYAWHIP Macro

C

```
#define GFX_W3_PAPAYAWHIP GFX_RGBConvert(255, 239, 213)
```

Description

This is macro GFX_W3_PAPAYAWHIP.

5.2.2.7.26.388 GFX_W3_PEACHPUFF Macro

C

```
#define GFX_W3_PEACHPUFF GFX_RGBConvert(255, 218, 185)
```

Description

This is macro GFX_W3_PEACHPUFF.

5.2.2.7.26.389 GFX_W3_PERU Macro

C

```
#define GFX_W3_PERU GFX_RGBConvert(205, 133, 63)
```

Description

This is macro GFX_W3_PERU.

5.2.2.7.26.390 GFX_W3_PINK Macro

C

```
#define GFX_W3_PINK GFX_RGBConvert(255, 192, 203)
```

Description

This is macro GFX_W3_PINK.

5.2.2.7.26.391 GFX_W3_PLUM Macro

C

```
#define GFX_W3_PLUM GFX_RGBConvert(221, 160, 221)
```

Description

This is macro GFX_W3_PLUM.

5.2.2.7.26.392 GFX_W3_POWDERBLUE Macro**C**

```
#define GFX_W3_POWDERBLUE GFX_RGBConvert(176, 224, 230)
```

Description

This is macro GFX_W3_POWDERBLUE.

5.2.2.7.26.393 GFX_W3_PURPLE Macro**C**

```
#define GFX_W3_PURPLE GFX_RGBConvert(128, 0, 128)
```

Description

This is macro GFX_W3_PURPLE.

5.2.2.7.26.394 GFX_W3_RED Macro**C**

```
#define GFX_W3_RED GFX_RGBConvert(255, 0, 0)
```

Description

This is macro GFX_W3_RED.

5.2.2.7.26.395 GFX_W3_ROSYBROWN Macro**C**

```
#define GFX_W3_ROSYBROWN GFX_RGBConvert(188, 143, 143)
```

Description

This is macro GFX_W3_ROSYBROWN.

5.2.2.7.26.396 GFX_W3_ROYALBLUE Macro**C**

```
#define GFX_W3_ROYALBLUE GFX_RGBConvert( 65, 105, 225)
```

Description

This is macro GFX_W3_ROYALBLUE.

5.2.2.7.26.397 GFX_W3_SADDLEBROWN Macro**C**

```
#define GFX_W3_SADDLEBROWN GFX_RGBConvert(139, 69, 19)
```

Description

This is macro GFX_W3_SADDLEBROWN.

5.2.2.7.26.398 GFX_W3_SALMON Macro

C

```
#define GFX_W3_SALMON GFX_RGBConvert(250, 128, 114)
```

Description

This is macro GFX_W3_SALMON.

5.2.2.7.26.399 GFX_W3_SANDYGREEN Macro

C

```
#define GFX_W3_SANDYGREEN GFX_RGBConvert(244, 164, 96)
```

Description

This is macro GFX_W3_SANDYGREEN.

5.2.2.7.26.400 GFX_W3_SEAGREEN Macro

C

```
#define GFX_W3_SEAGREEN GFX_RGBConvert( 46, 139, 87)
```

Description

This is macro GFX_W3_SEAGREEN.

5.2.2.7.26.401 GFX_W3_SEASHELL Macro

C

```
#define GFX_W3_SEASHELL GFX_RGBConvert(255, 245, 238)
```

Description

This is macro GFX_W3_SEASHELL.

5.2.2.7.26.402 GFX_W3_SIENNA Macro

C

```
#define GFX_W3_SIENNA GFX_RGBConvert(160, 82, 45)
```

Description

This is macro GFX_W3_SIENNA.

5.2.2.7.26.403 GFX_W3_SILVER Macro

C

```
#define GFX_W3_SILVER GFX_RGBConvert(192, 192, 192)
```

Description

This is macro GFX_W3_SILVER.

5.2.2.7.26.404 GFX_W3_SKYBLUE Macro

C

```
#define GFX_W3_SKYBLUE GFX_RGBConvert(135, 206, 235)
```


Description

This is macro GFX_W3_SKYBLUE.

5.2.2.7.26.405 GFX_W3_SLATEBLUE Macro

C

```
#define GFX_W3_SLATEBLUE GFX_RGBConvert(106, 90, 205)
```

Description

This is macro GFX_W3_SLATEBLUE.

5.2.2.7.26.406 GFX_W3_SLATEGRAY Macro

C

```
#define GFX_W3_SLATEGRAY GFX_RGBConvert(112, 128, 144)
```

Description

This is macro GFX_W3_SLATEGRAY.

5.2.2.7.26.407 GFX_W3_SLATEGREY Macro

C

```
#define GFX_W3_SLATEGREY GFX_RGBConvert(112, 128, 144)
```

Description

This is macro GFX_W3_SLATEGREY.

5.2.2.7.26.408 GFX_W3_SNOW Macro

C

```
#define GFX_W3_SNOW GFX_RGBConvert(255, 250, 250)
```

Description

This is macro GFX_W3_SNOW.

5.2.2.7.26.409 GFX_W3_SPRINGGREEN Macro

C

```
#define GFX_W3_SPRINGGREEN GFX_RGBConvert( 0, 255, 127)
```

Description

This is macro GFX_W3_SPRINGGREEN.

5.2.2.7.26.410 GFX_W3_STEELBLUE Macro

C

```
#define GFX_W3_STEELBLUE GFX_RGBConvert( 70, 130, 180)
```

Description

This is macro GFX_W3_STEELBLUE.

5.2.2.7.26.411 GFX_W3_TAN Macro

C

```
#define GFX_W3_TAN GFX_RGBConvert(210, 180, 140)
```

Description

This is macro GFX_W3_TAN.

5.2.2.7.26.412 GFX_W3_TEAL Macro

C

```
#define GFX_W3_TEAL GFX_RGBConvert( 0, 128, 128)
```

Description

This is macro GFX_W3_TEAL.

5.2.2.7.26.413 GFX_W3_THISTLE Macro

C

```
#define GFX_W3_THISTLE GFX_RGBConvert(216, 191, 216)
```

Description

This is macro GFX_W3_THISTLE.

5.2.2.7.26.414 GFX_W3_TOMATO Macro

C

```
#define GFX_W3_TOMATO GFX_RGBConvert(255, 99, 71)
```

Description

This is macro GFX_W3_TOMATO.

5.2.2.7.26.415 GFX_W3_TURQUOISE Macro

C

```
#define GFX_W3_TURQUOISE GFX_RGBConvert( 64, 224, 208)
```

Description

This is macro GFX_W3_TURQUOISE.

5.2.2.7.26.416 GFX_W3_VIOLET Macro

C

```
#define GFX_W3_VIOLET GFX_RGBConvert(238, 130, 238)
```

Description

This is macro GFX_W3_VIOLET.

5.2.2.7.26.417 GFX_W3_WHEAT Macro

C

```
#define GFX_W3_WHEAT GFX_RGBConvert(245, 222, 179)
```

Description

This is macro GFX_W3_WHEAT.

5.2.2.7.26.418 GFX_W3_WHITE Macro**C**

```
#define GFX_W3_WHITE GFX_RGBConvert(255, 255, 255)
```

Description

This is macro GFX_W3_WHITE.

5.2.2.7.26.419 GFX_W3_WHITESMOKE Macro**C**

```
#define GFX_W3_WHITESMOKE GFX_RGBConvert(245, 245, 245)
```

Description

This is macro GFX_W3_WHITESMOKE.

5.2.2.7.26.420 GFX_W3_YELLOW Macro**C**

```
#define GFX_W3_YELLOW GFX_RGBConvert(255, 255, 0)
```

Description

This is macro GFX_W3_YELLOW.

5.2.2.7.26.421 GFX_W3_YELLOWGREEN Macro**C**

```
#define GFX_W3_YELLOWGREEN GFX_RGBConvert(154, 205, 50)
```

Description

This is macro GFX_W3_YELLOWGREEN.

5.2.2.7.26.422 GFX_X11_ALICE_BLUE Macro**C**

```
#define GFX_X11_ALICE_BLUE GFX_RGBConvert(0xF0, 0xF8, 0xFF)
```

Description

This is macro GFX_X11_ALICE_BLUE.

5.2.2.7.26.423 GFX_X11_ANTIQUUE_WHITE Macro**C**

```
#define GFX_X11_ANTIQUUE_WHITE GFX_RGBConvert(0xFA, 0xEB, 0xD7)
```

Description

This is macro GFX_X11_ANTIQUUE_WHITE.

5.2.2.7.26.424 GFX_X11_AQUA Macro

C

```
#define GFX_X11_AQUA GFX_RGBConvert(0x00, 0xFF, 0xFF)
```

Description

This is macro GFX_X11_AQUA.

5.2.2.7.26.425 GFX_X11_AQUAMARINE Macro

C

```
#define GFX_X11_AQUAMARINE GFX_RGBConvert(0x7F, 0xFF, 0xD4)
```

Description

This is macro GFX_X11_AQUAMARINE.

5.2.2.7.26.426 GFX_X11_AZURE Macro

C

```
#define GFX_X11_AZURE GFX_RGBConvert(0xF0, 0xFF, 0xFF)
```

Description

This is macro GFX_X11_AZURE.

5.2.2.7.26.427 GFX_X11_BEIGE Macro

C

```
#define GFX_X11_BEIGE GFX_RGBConvert(0xF5, 0xF5, 0xDC)
```

Description

This is macro GFX_X11_BEIGE.

5.2.2.7.26.428 GFX_X11_BISQUE Macro

C

```
#define GFX_X11_BISQUE GFX_RGBConvert(0xFF, 0xE4, 0xC4)
```

Description

This is macro GFX_X11_BISQUE.

5.2.2.7.26.429 GFX_X11_BLACK Macro

C

```
#define GFX_X11_BLACK GFX_RGBConvert(0x00, 0x00, 0x00)
```

Description

This is macro GFX_X11_BLACK.

5.2.2.7.26.430 GFX_X11_BLANCHED_ALMOND Macro

C

```
#define GFX_X11_BLANCHED_ALMOND GFX_RGBConvert(0xFF, 0xEB, 0xCD)
```

Description

This is macro GFX_X11_BLANCHED_ALMOND.

5.2.2.7.26.431 GFX_X11_BLUE Macro**C**

```
#define GFX_X11_BLUE GFX_RGBConvert(0x00, 0x00, 0xFF)
```

Description

This is macro GFX_X11_BLUE.

5.2.2.7.26.432 GFX_X11_BLUE_VIOLET Macro**C**

```
#define GFX_X11_BLUE_VIOLET GFX_RGBConvert(0x8A, 0x2B, 0xE2)
```

Description

This is macro GFX_X11_BLUE_VIOLET.

5.2.2.7.26.433 GFX_X11_BROWN Macro**C**

```
#define GFX_X11_BROWN GFX_RGBConvert(0xA5, 0x2A, 0x2A)
```

Description

This is macro GFX_X11_BROWN.

5.2.2.7.26.434 GFX_X11_BURLY_WOOD Macro**C**

```
#define GFX_X11_BURLY_WOOD GFX_RGBConvert(0xDE, 0xB8, 0x87)
```

Description

This is macro GFX_X11_BURLY_WOOD.

5.2.2.7.26.435 GFX_X11_CADEL_BLUE Macro**C**

```
#define GFX_X11_CADEL_BLUE GFX_RGBConvert(0x5F, 0x9E, 0xA0)
```

Description

This is macro GFX_X11_CADEL_BLUE.

5.2.2.7.26.436 GFX_X11_CHARTEUSE Macro**C**

```
#define GFX_X11_CHARTEUSE GFX_RGBConvert(0x7F, 0xFF, 0x00)
```

Description

This is macro GFX_X11_CHARTEUSE.

5.2.2.7.26.437 GFX_X11_CHOCOLATE Macro

C

```
#define GFX_X11_CHOCOLATE GFX_RGBConvert(0xD2, 0x69, 0x1E)
```

Description

This is macro GFX_X11_CHOCOLATE.

5.2.2.7.26.438 GFX_X11_CORAL Macro

C

```
#define GFX_X11_CORAL GFX_RGBConvert(0xFF, 0x7F, 0x50)
```

Description

This is macro GFX_X11_CORAL.

5.2.2.7.26.439 GFX_X11_CORNFLOWER_BLUE Macro

C

```
#define GFX_X11_CORNFLOWER_BLUE GFX_RGBConvert(0x64, 0x95, 0xED)
```

Description

This is macro GFX_X11_CORNFLOWER_BLUE.

5.2.2.7.26.440 GFX_X11_CORNSILK Macro

C

```
#define GFX_X11_CORNSILK GFX_RGBConvert(0xFF, 0xF8, 0xDC)
```

Description

X11: Brown Colors

5.2.2.7.26.441 GFX_X11_CRIMSON Macro

C

```
#define GFX_X11_CRIMSON GFX_RGBConvert(0xDC, 0x14, 0x3C)
```

Description

This is macro GFX_X11_CRIMSON.

5.2.2.7.26.442 GFX_X11_CYAN Macro

C

```
#define GFX_X11_CYAN GFX_RGBConvert(0x00, 0xFF, 0xFF)
```

Description

This is macro GFX_X11_CYAN.

5.2.2.7.26.443 GFX_X11_DARK_BLUE Macro

C

```
#define GFX_X11_DARK_BLUE GFX_RGBConvert(0x00, 0x00, 0x8B)
```

Description

This is macro GFX_X11_DARK_BLUE.

5.2.2.7.26.444 GFX_X11_DARK_CYAN Macro

C

```
#define GFX_X11_DARK_CYAN GFX_RGBConvert(0x00, 0x8B, 0x8B)
```

Description

This is macro GFX_X11_DARK_CYAN.

5.2.2.7.26.445 GFX_X11_DARK_GOLDENROD Macro

C

```
#define GFX_X11_DARK_GOLDENROD GFX_RGBConvert(0xB8, 0x86, 0x0B)
```

Description

This is macro GFX_X11_DARK_GOLDENROD.

5.2.2.7.26.446 GFX_X11_DARK_GREEN Macro

C

```
#define GFX_X11_DARK_GREEN GFX_RGBConvert(0x00, 0x64, 0x00)
```

Description

This is macro GFX_X11_DARK_GREEN.

5.2.2.7.26.447 GFX_X11_DARK_GREY Macro

C

```
#define GFX_X11_DARK_GREY GFX_RGBConvert(0xA9, 0xA9, 0xA9)
```

Description

This is macro GFX_X11_DARK_GREY.

5.2.2.7.26.448 GFX_X11_DARK_KHAKI Macro

C

```
#define GFX_X11_DARK_KHAKI GFX_RGBConvert(0xBD, 0xB7, 0x6B)
```

Description

This is macro GFX_X11_DARK_KHAKI.

5.2.2.7.26.449 GFX_X11_DARK_MAGENTA Macro

C

```
#define GFX_X11_DARK_MAGENTA GFX_RGBConvert(0x8B, 0x00, 0x8B)
```

Description

This is macro GFX_X11_DARK_MAGENTA.

5.2.2.7.26.450 GFX_X11_DARK_OLIVE_GREEN Macro

C

```
#define GFX_X11_DARK_OLIVE_GREEN GFX_RGBConvert(0x55, 0x6B, 0x2F)
```

Description

X11: Green Colors

5.2.2.7.26.451 GFX_X11_DARK_ORANGE Macro

C

```
#define GFX_X11_DARK_ORANGE GFX_RGBConvert(0xFF, 0x8C, 0x00)
```

Description

This is macro GFX_X11_DARK_ORANGE.

5.2.2.7.26.452 GFX_X11_DARK_ORCHID Macro

C

```
#define GFX_X11_DARK_ORCHID GFX_RGBConvert(0x99, 0x32, 0xCC)
```

Description

This is macro GFX_X11_DARK_ORCHID.

5.2.2.7.26.453 GFX_X11_DARK_RED Macro

C

```
#define GFX_X11_DARK_RED GFX_RGBConvert(0x8B, 0x00, 0x00)
```

Description

This is macro GFX_X11_DARK_RED.

5.2.2.7.26.454 GFX_X11_DARK_SALMON Macro

C

```
#define GFX_X11_DARK_SALMON GFX_RGBConvert(0xEA, 0x96, 0x7A)
```

Description

This is macro GFX_X11_DARK_SALMON.

5.2.2.7.26.455 GFX_X11_DARK_SEA_GREEN Macro

C

```
#define GFX_X11_DARK_SEA_GREEN GFX_RGBConvert(0x8F, 0xBC, 0x8F)
```

Description

This is macro GFX_X11_DARK_SEA_GREEN.

5.2.2.7.26.456 GFX_X11_DARK_SLATE_BLUE Macro

C

```
#define GFX_X11_DARK_SLATE_BLUE GFX_RGBConvert(0x48, 0x3D, 0x8B)
```


Description

This is macro GFX_X11_DARK_SLATE_BLUE.

5.2.2.7.26.457 GFX_X11_DARK_SLATE_GREY Macro**C**

```
#define GFX_X11_DARK_SLATE_GREY GFX_RGBConvert(0x2F, 0x4F, 0x4F)
```

Description

This is macro GFX_X11_DARK_SLATE_GREY.

5.2.2.7.26.458 GFX_X11_DARK_TURQUOISE Macro**C**

```
#define GFX_X11_DARK_TURQUOISE GFX_RGBConvert(0x00, 0xCE, 0xD1)
```

Description

This is macro GFX_X11_DARK_TURQUOISE.

5.2.2.7.26.459 GFX_X11_DARK_VIOLET Macro**C**

```
#define GFX_X11_DARK_VIOLET GFX_RGBConvert(0x94, 0x00, 0xD3)
```

Description

This is macro GFX_X11_DARK_VIOLET.

5.2.2.7.26.460 GFX_X11_DEEP_PINK Macro**C**

```
#define GFX_X11_DEEP_PINK GFX_RGBConvert(0xFF, 0x14, 0x93)
```

Description

This is macro GFX_X11_DEEP_PINK.

5.2.2.7.26.461 GFX_X11_DEEP_SKY_BLUE Macro**C**

```
#define GFX_X11_DEEP_SKY_BLUE GFX_RGBConvert(0x00, 0xBF, 0xFF)
```

Description

This is macro GFX_X11_DEEP_SKY_BLUE.

5.2.2.7.26.462 GFX_X11_DIM_GREY Macro**C**

```
#define GFX_X11_DIM_GREY GFX_RGBConvert(0x69, 0x69, 0x69)
```

Description

This is macro GFX_X11_DIM_GREY.

5.2.2.7.26.463 GFX_X11_DODGER_BLUE Macro

C

```
#define GFX_X11_DODGER_BLUE GFX_RGBConvert(0x1E, 0x90, 0xFF)
```

Description

This is macro GFX_X11_DODGER_BLUE.

5.2.2.7.26.464 GFX_X11_FIRE_BRICK Macro

C

```
#define GFX_X11_FIRE_BRICK GFX_RGBConvert(0xB2, 0x22, 0x22)
```

Description

This is macro GFX_X11_FIRE_BRICK.

5.2.2.7.26.465 GFX_X11_FLORAL_WHITE Macro

C

```
#define GFX_X11_FLORAL_WHITE GFX_RGBConvert(0xFF, 0xFA, 0xF0)
```

Description

This is macro GFX_X11_FLORAL_WHITE.

5.2.2.7.26.466 GFX_X11_FOREST_GREEN Macro

C

```
#define GFX_X11_FOREST_GREEN GFX_RGBConvert(0x22, 0x8B, 0x22)
```

Description

This is macro GFX_X11_FOREST_GREEN.

5.2.2.7.26.467 GFX_X11_FUCHSIA Macro

C

```
#define GFX_X11_FUCHSIA GFX_RGBConvert(0xFF, 0x00, 0xFF)
```

Description

This is macro GFX_X11_FUCHSIA.

5.2.2.7.26.468 GFX_X11_GAINSBORO Macro

C

```
#define GFX_X11_GAINSBORO GFX_RGBConvert(0xDC, 0xDC, 0xDC)
```

Description

This is macro GFX_X11_GAINSBORO.

5.2.2.7.26.469 GFX_X11_GHOST_WHITE Macro

C

```
#define GFX_X11_GHOST_WHITE GFX_RGBConvert(0xF8, 0xF8, 0xFF)
```

Description

This is macro GFX_X11_GHOST_WHITE.

5.2.2.7.26.470 GFX_X11_GOLD Macro

C

```
#define GFX_X11_GOLD GFX_RGBConvert(0xFF, 0xD7, 0x00)
```

Description

This is macro GFX_X11_GOLD.

5.2.2.7.26.471 GFX_X11_GOLDENROD Macro

C

```
#define GFX_X11_GOLDENROD GFX_RGBConvert(0xDA, 0xA5, 0x20)
```

Description

This is macro GFX_X11_GOLDENROD.

5.2.2.7.26.472 GFX_X11_GREEN Macro

C

```
#define GFX_X11_GREEN GFX_RGBConvert(0x00, 0x80, 0x00)
```

Description

This is macro GFX_X11_GREEN.

5.2.2.7.26.473 GFX_X11_GREEN_YELLOW Macro

C

```
#define GFX_X11_GREEN_YELLOW GFX_RGBConvert(0xAD, 0xFF, 0x2F)
```

Description

This is macro GFX_X11_GREEN_YELLOW.

5.2.2.7.26.474 GFX_X11_GREY Macro

C

```
#define GFX_X11_GREY GFX_RGBConvert(0x80, 0x80, 0x80)
```

Description

This is macro GFX_X11_GREY.

5.2.2.7.26.475 GFX_X11_HONEYDEW Macro

C

```
#define GFX_X11_HONEYDEW GFX_RGBConvert(0xF0, 0xFF, 0xF0)
```

Description

This is macro GFX_X11_HONEYDEW.

5.2.2.7.26.476 GFX_X11_HOT_PINK Macro

C

```
#define GFX_X11_HOT_PINK GFX_RGBConvert(0xFF, 0x69, 0xB4)
```

Description

This is macro GFX_X11_HOT_PINK.

5.2.2.7.26.477 GFX_X11_INDIAN_RED Macro

C

```
#define GFX_X11_INDIAN_RED GFX_RGBConvert(0xCD, 0x5C, 0x5C)
```

Description

This is macro GFX_X11_INDIAN_RED.

5.2.2.7.26.478 GFX_X11_INDIGO Macro

C

```
#define GFX_X11_INDIGO GFX_RGBConvert(0x4B, 0x00, 0x82)
```

Description

This is macro GFX_X11_INDIGO.

5.2.2.7.26.479 GFX_X11_IVORY Macro

C

```
#define GFX_X11_IVORY GFX_RGBConvert(0xFF, 0xFF, 0xF0)
```

Description

This is macro GFX_X11_IVORY.

5.2.2.7.26.480 GFX_X11_KHAKI Macro

C

```
#define GFX_X11_KHAKI GFX_RGBConvert(0xF0, 0xE6, 0x8C)
```

Description

This is macro GFX_X11_KHAKI.

5.2.2.7.26.481 GFX_X11_LAVENDER Macro

C

```
#define GFX_X11_LAVENDER GFX_RGBConvert(0xE6, 0xE6, 0xFA)
```

Description

X11: [BLUE](#) COLORS

5.2.2.7.26.482 GFX_X11_LAVENDOR_BLUSH Macro

C

```
#define GFX_X11_LAVENDOR_BLUSH GFX_RGBConvert(0xFF, 0xF0, 0xF5)
```

Description

This is macro GFX_X11_LAVENDOR_BLUSH.

5.2.2.7.26.483 GFX_X11_LAWN_GREEN Macro

C

```
#define GFX_X11_LAWN_GREEN GFX_RGBConvert(0x7C, 0xFC, 0x00)
```

Description

This is macro GFX_X11_LAWN_GREEN.

5.2.2.7.26.484 GFX_X11_LEMON_CHIFFON Macro

C

```
#define GFX_X11_LEMON_CHIFFON GFX_RGBConvert(0xFF, 0xFA, 0xCD)
```

Description

This is macro GFX_X11_LEMON_CHIFFON.

5.2.2.7.26.485 GFX_X11_LIGHT_BLUE Macro

C

```
#define GFX_X11_LIGHT_BLUE GFX_RGBConvert(0xAD, 0xD8, 0xE6)
```

Description

This is macro GFX_X11_LIGHT_BLUE.

5.2.2.7.26.486 GFX_X11_LIGHT_CAROL Macro

C

```
#define GFX_X11_LIGHT_CAROL GFX_RGBConvert(0xF0, 0x80, 0x80)
```

Description

This is macro GFX_X11_LIGHT_CAROL.

5.2.2.7.26.487 GFX_X11_LIGHT_CYAN Macro

C

```
#define GFX_X11_LIGHT_CYAN GFX_RGBConvert(0xE0, 0xFF, 0xFF)
```

Description

This is macro GFX_X11_LIGHT_CYAN.

5.2.2.7.26.488 GFX_X11_LIGHT_GOLDENROD_YELLOW Macro

C

```
#define GFX_X11_LIGHT_GOLDENROD_YELLOW GFX_RGBConvert(0xFA, 0xFA, 0xD2)
```

Description

This is macro GFX_X11_LIGHT_GOLDENROD_YELLOW.

5.2.2.7.26.489 GFX_X11_LIGHT_GRAY Macro

C

```
#define GFX_X11_LIGHT_GRAY GFX_RGBConvert(0xD3, 0xD3, 0xD3)
```

Description

This is macro GFX_X11_LIGHT_GRAY.

5.2.2.7.26.490 GFX_X11_LIGHT_GREEN Macro

C

```
#define GFX_X11_LIGHT_GREEN GFX_RGBConvert(0x90, 0xEE, 0x90)
```

Description

This is macro GFX_X11_LIGHT_GREEN.

5.2.2.7.26.491 GFX_X11_LIGHT_PINK Macro

C

```
#define GFX_X11_LIGHT_PINK GFX_RGBConvert(0xFF, 0xB6, 0xC1)
```

Description

This is macro GFX_X11_LIGHT_PINK.

5.2.2.7.26.492 GFX_X11_LIGHT_SALMON Macro

C

```
#define GFX_X11_LIGHT_SALMON GFX_RGBConvert(0xFF, 0xA0, 0x7A)
```

Description

X11: Red Colors

5.2.2.7.26.493 GFX_X11_LIGHT_SEA_GREEN Macro

C

```
#define GFX_X11_LIGHT_SEA_GREEN GFX_RGBConvert(0x20, 0xB2, 0xAA)
```

Description

This is macro GFX_X11_LIGHT_SEA_GREEN.

5.2.2.7.26.494 GFX_X11_LIGHT_SKY_BLUE Macro

C

```
#define GFX_X11_LIGHT_SKY_BLUE GFX_RGBConvert(0x87, 0xCE, 0xFA)
```

Description

This is macro GFX_X11_LIGHT_SKY_BLUE.

5.2.2.7.26.495 GFX_X11_LIGHT_SLATE_GREY Macro

C

```
#define GFX_X11_LIGHT_SLATE_GREY GFX_RGBConvert(0x77, 0x88, 0x99)
```

Description

This is macro GFX_X11_LIGHT_SLATE_GREY.

5.2.2.7.26.496 GFX_X11_LIGHT_STEEL_BLUE Macro**C**

```
#define GFX_X11_LIGHT_STEEL_BLUE GFX_RGBConvert(0xB0, 0xC4, 0xDE)
```

Description

X11: [BLUE](#) COLORS

5.2.2.7.26.497 GFX_X11_LIGHT_YELLOW Macro**C**

```
#define GFX_X11_LIGHT_YELLOW GFX_RGBConvert(0xFF, 0xFF, 0xE0)
```

Description

This is macro GFX_X11_LIGHT_YELLOW.

5.2.2.7.26.498 GFX_X11_LIME Macro**C**

```
#define GFX_X11_LIME GFX_RGBConvert(0x00, 0xFF, 0x00)
```

Description

This is macro GFX_X11_LIME.

5.2.2.7.26.499 GFX_X11_LIME_GREEN Macro**C**

```
#define GFX_X11_LIME_GREEN GFX_RGBConvert(0x32, 0xCD, 0x32)
```

Description

This is macro GFX_X11_LIME_GREEN.

5.2.2.7.26.500 GFX_X11_LINEN Macro**C**

```
#define GFX_X11_LINEN GFX_RGBConvert(0xFA, 0xF0, 0xE6)
```

Description

This is macro GFX_X11_LINEN.

5.2.2.7.26.501 GFX_X11_MAGENTA Macro**C**

```
#define GFX_X11_MAGENTA GFX_RGBConvert(0xFF, 0x00, 0xFF)
```

Description

This is macro GFX_X11_MAGENTA.

5.2.2.7.26.502 GFX_X11_MARRON Macro

C

```
#define GFX_X11_MARRON GFX_RGBConvert(0x80, 0x00, 0x00)
```

Description

This is macro GFX_X11_MARRON.

5.2.2.7.26.503 GFX_X11_MEDIUM_AQUAMARINE Macro

C

```
#define GFX_X11_MEDIUM_AQUAMARINE GFX_RGBConvert(0x66, 0xCD, 0xAA)
```

Description

X11: [CYAN](#) COLORS

5.2.2.7.26.504 GFX_X11_MEDIUM_BLUE Macro

C

```
#define GFX_X11_MEDIUM_BLUE GFX_RGBConvert(0x00, 0x00, 0xCD)
```

Description

This is macro GFX_X11_MEDIUM_BLUE.

5.2.2.7.26.505 GFX_X11_MEDIUM_ORCHID Macro

C

```
#define GFX_X11_MEDIUM_ORCHID GFX_RGBConvert(0xBA, 0x55, 0xD3)
```

Description

This is macro GFX_X11_MEDIUM_ORCHID.

5.2.2.7.26.506 GFX_X11_MEDIUM_PURPLE Macro

C

```
#define GFX_X11_MEDIUM_PURPLE GFX_RGBConvert(0x93, 0x70, 0xDB)
```

Description

This is macro GFX_X11_MEDIUM_PURPLE.

5.2.2.7.26.507 GFX_X11_MEDIUM_SEA_GREEN Macro

C

```
#define GFX_X11_MEDIUM_SEA_GREEN GFX_RGBConvert(0x3C, 0xB3, 0x71)
```

Description

This is macro GFX_X11_MEDIUM_SEA_GREEN.

5.2.2.7.26.508 GFX_X11_MEDIUM_SLATE_BLUE Macro

C

```
#define GFX_X11_MEDIUM_SLATE_BLUE GFX_RGBConvert(0x7B, 0x68, 0xEE)
```


Description

This is macro GFX_X11_MEDIUM_SLATE_BLUE.

5.2.2.7.26.509 GFX_X11_MEDIUM_SPRING_GREEN Macro**C**

```
#define GFX_X11_MEDIUM_SPRING_GREEN GFX_RGBConvert(0x00, 0xFA, 0x9A)
```

Description

This is macro GFX_X11_MEDIUM_SPRING_GREEN.

5.2.2.7.26.510 GFX_X11_MEDIUM_TURQUOISE Macro**C**

```
#define GFX_X11_MEDIUM_TURQUOISE GFX_RGBConvert(0x48, 0xD1, 0xCC)
```

Description

This is macro GFX_X11_MEDIUM_TURQUOISE.

5.2.2.7.26.511 GFX_X11_MEDIUM_VIOLET_RED Macro**C**

```
#define GFX_X11_MEDIUM_VIOLET_RED GFX_RGBConvert(0xC7, 0x15, 0x85)
```

Description

This is macro GFX_X11_MEDIUM_VIOLET_RED.

5.2.2.7.26.512 GFX_X11_MIDNIGHT_BLUE Macro**C**

```
#define GFX_X11_MIDNIGHT_BLUE GFX_RGBConvert(0x19, 0x19, 0x70)
```

Description

This is macro GFX_X11_MIDNIGHT_BLUE.

5.2.2.7.26.513 GFX_X11_MINT_CREAM Macro**C**

```
#define GFX_X11_MINT_CREAM GFX_RGBConvert(0xF5, 0xFF, 0xFA)
```

Description

This is macro GFX_X11_MINT_CREAM.

5.2.2.7.26.514 GFX_X11_MISTY_ROSE Macro**C**

```
#define GFX_X11_MISTY_ROSE GFX_RGBConvert(0xFF, 0xE4, 0xE1)
```

Description

This is macro GFX_X11_MISTY_ROSE.

5.2.2.7.26.515 GFX_X11_MOCCASIN Macro

C

```
#define GFX_X11_MOCCASIN GFX_RGBConvert(0xFF, 0xE4, 0xB5)
```

Description

This is macro GFX_X11_MOCCASIN.

5.2.2.7.26.516 GFX_X11_NAVAJO_WHITE Macro

C

```
#define GFX_X11_NAVAJO_WHITE GFX_RGBConvert(0xFF, 0xDE, 0xAD)
```

Description

This is macro GFX_X11_NAVAJO_WHITE.

5.2.2.7.26.517 GFX_X11_NAVY Macro

C

```
#define GFX_X11_NAVY GFX_RGBConvert(0x00, 0x00, 0x80)
```

Description

This is macro GFX_X11_NAVY.

5.2.2.7.26.518 GFX_X11_OLD_LACE Macro

C

```
#define GFX_X11_OLD_LACE GFX_RGBConvert(0xFD, 0xF5, 0xE6)
```

Description

This is macro GFX_X11_OLD_LACE.

5.2.2.7.26.519 GFX_X11_OLIVE Macro

C

```
#define GFX_X11_OLIVE GFX_RGBConvert(0x80, 0x80, 0x00)
```

Description

This is macro GFX_X11_OLIVE.

5.2.2.7.26.520 GFX_X11_OLIVE_DRAB Macro

C

```
#define GFX_X11_OLIVE_DRAB GFX_RGBConvert(0x6B, 0x8E, 0x23)
```

Description

This is macro GFX_X11_OLIVE_DRAB.

5.2.2.7.26.521 GFX_X11_ORANGE Macro

C

```
#define GFX_X11_ORANGE GFX_RGBConvert(0xFF, 0xA5, 0x00)
```

Description

This is macro GFX_X11_ORANGE.

5.2.2.7.26.522 GFX_X11_ORANGE_RED Macro

C

```
#define GFX_X11_ORANGE_RED GFX_RGBConvert(0xFF, 0x45, 0x00)
```

Description

X11: Orange Colors

5.2.2.7.26.523 GFX_X11_ORCHID Macro

C

```
#define GFX_X11_ORCHID GFX_RGBConvert(0xDA, 0x70, 0xD6)
```

Description

This is macro GFX_X11_ORCHID.

5.2.2.7.26.524 GFX_X11_PALE_GOLDENROD Macro

C

```
#define GFX_X11_PALE_GOLDENROD GFX_RGBConvert(0xEE, 0xE8, 0xAA)
```

Description

This is macro GFX_X11_PALE_GOLDENROD.

5.2.2.7.26.525 GFX_X11_PALE_GREEN Macro

C

```
#define GFX_X11_PALE_GREEN GFX_RGBConvert(0x98, 0xFB, 0x98)
```

Description

This is macro GFX_X11_PALE_GREEN.

5.2.2.7.26.526 GFX_X11_PALE_TURQUOISE Macro

C

```
#define GFX_X11_PALE_TURQUOISE GFX_RGBConvert(0xAF, 0xEE, 0xEE)
```

Description

This is macro GFX_X11_PALE_TURQUOISE.

5.2.2.7.26.527 GFX_X11_PALE_VIOLET_RED Macro

C

```
#define GFX_X11_PALE_VIOLET_RED GFX_RGBConvert(0xDB, 0x70, 0x93)
```

Description

This is macro GFX_X11_PALE_VIOLET_RED.

5.2.2.7.26.528 GFX_X11_PAPAYA_WHIP Macro

C

```
#define GFX_X11_PAPAYA_WHIP GFX_RGBConvert(0xFF, 0xEF, 0xD5)
```

Description

This is macro GFX_X11_PAPAYA_WHIP.

5.2.2.7.26.529 GFX_X11_PEACH_PUFF Macro

C

```
#define GFX_X11_PEACH_PUFF GFX_RGBConvert(0xFF, 0xDA, 0xB9)
```

Description

This is macro GFX_X11_PEACH_PUFF.

5.2.2.7.26.530 GFX_X11_PERU Macro

C

```
#define GFX_X11_PERU GFX_RGBConvert(0xCD, 0x85, 0x3F)
```

Description

This is macro GFX_X11_PERU.

5.2.2.7.26.531 GFX_X11_PINK Macro

C

```
#define GFX_X11_PINK GFX_RGBConvert(0xFF, 0xC0, 0xCB)
```

Description

X11: Pink Colors

5.2.2.7.26.532 GFX_X11_PLUM Macro

C

```
#define GFX_X11_PLUM GFX_RGBConvert(0xDD, 0xA0, 0xDD)
```

Description

This is macro GFX_X11_PLUM.

5.2.2.7.26.533 GFX_X11_POWDER_BLUE Macro

C

```
#define GFX_X11_POWDER_BLUE GFX_RGBConvert(0xB0, 0xE0, 0xE6)
```

Description

This is macro GFX_X11_POWDER_BLUE.

5.2.2.7.26.534 GFX_X11_PURPLE Macro

C

```
#define GFX_X11_PURPLE GFX_RGBConvert(0x80, 0x00, 0x80)
```

Description

This is macro GFX_X11_PURPLE.

5.2.2.7.26.535 GFX_X11_RED Macro

C

```
#define GFX_X11_RED GFX_RGBConvert(0xFF, 0x00, 0x00)
```

Description

This is macro GFX_X11_RED.

5.2.2.7.26.536 GFX_X11_ROSY_BROWN Macro

C

```
#define GFX_X11_ROSY_BROWN GFX_RGBConvert(0xBc, 0x8F, 0x8F)
```

Description

This is macro GFX_X11_ROSY_BROWN.

5.2.2.7.26.537 GFX_X11_ROYAL_BLUE Macro

C

```
#define GFX_X11_ROYAL_BLUE GFX_RGBConvert(0x41, 0x69, 0xE1)
```

Description

This is macro GFX_X11_ROYAL_BLUE.

5.2.2.7.26.538 GFX_X11_SADDLE_BROWN Macro

C

```
#define GFX_X11_SADDLE_BROWN GFX_RGBConvert(0x8B, 0x45, 0x13)
```

Description

This is macro GFX_X11_SADDLE_BROWN.

5.2.2.7.26.539 GFX_X11_SALMON Macro

C

```
#define GFX_X11_SALMON GFX_RGBConvert(0xFA, 0x80, 0x72)
```

Description

This is macro GFX_X11_SALMON.

5.2.2.7.26.540 GFX_X11_SANDY_BROWN Macro

C

```
#define GFX_X11_SANDY_BROWN GFX_RGBConvert(0xF4, 0xA4, 0x60)
```

Description

This is macro GFX_X11_SANDY_BROWN.

5.2.2.7.26.541 GFX_X11_SEA_GREEN Macro

C

```
#define GFX_X11_SEA_GREEN GFX_RGBConvert(0x2E, 0x8B, 0x57)
```

Description

This is macro GFX_X11_SEA_GREEN.

5.2.2.7.26.542 GFX_X11_SEASHELL Macro

C

```
#define GFX_X11_SEASHELL GFX_RGBConvert(0xFF, 0xF5, 0xEE)
```

Description

This is macro GFX_X11_SEASHELL.

5.2.2.7.26.543 GFX_X11_SIENNA Macro

C

```
#define GFX_X11_SIENNA GFX_RGBConvert(0xA0, 0x52, 0x2D)
```

Description

This is macro GFX_X11_SIENNA.

5.2.2.7.26.544 GFX_X11_SILVER Macro

C

```
#define GFX_X11_SILVER GFX_RGBConvert(0xC0, 0xC0, 0xC0)
```

Description

This is macro GFX_X11_SILVER.

5.2.2.7.26.545 GFX_X11_SKY_BLUE Macro

C

```
#define GFX_X11_SKY_BLUE GFX_RGBConvert(0x87, 0xCE, 0xEB)
```

Description

This is macro GFX_X11_SKY_BLUE.

5.2.2.7.26.546 GFX_X11_SLATE_BLUE Macro

C

```
#define GFX_X11_SLATE_BLUE GFX_RGBConvert(0x6A, 0x5A, 0xCD)
```

Description

This is macro GFX_X11_SLATE_BLUE.

5.2.2.7.26.547 GFX_X11_SLATE_GREY Macro

C

```
#define GFX_X11_SLATE_GREY GFX_RGBConvert(0x70, 0x80, 0x90)
```

Description

This is macro GFX_X11_SLATE_GREY.

5.2.2.7.26.548 GFX_X11_SNOW Macro

C

```
#define GFX_X11_SNOW GFX_RGBConvert(0xFF, 0xFA, 0xFA)
```

Description

This is macro GFX_X11_SNOW.

5.2.2.7.26.549 GFX_X11_SPRING_GREEN Macro

C

```
#define GFX_X11_SPRING_GREEN GFX_RGBConvert(0x00, 0xFF, 0x7F)
```

Description

This is macro GFX_X11_SPRING_GREEN.

5.2.2.7.26.550 GFX_X11_STEEL_BLUE Macro

C

```
#define GFX_X11_STEEL_BLUE GFX_RGBConvert(0x46, 0x82, 0xB4)
```

Description

This is macro GFX_X11_STEEL_BLUE.

5.2.2.7.26.551 GFX_X11_TAN Macro

C

```
#define GFX_X11_TAN GFX_RGBConvert(0xD2, 0xB4, 0x8C)
```

Description

This is macro GFX_X11_TAN.

5.2.2.7.26.552 GFX_X11_TEAL Macro

C

```
#define GFX_X11_TEAL GFX_RGBConvert(0x00, 0x80, 0x80)
```

Description

This is macro GFX_X11_TEAL.

5.2.2.7.26.553 GFX_X11_THISTLE Macro

C

```
#define GFX_X11_THISTLE GFX_RGBConvert(0xD8, 0xBF, 0xD8)
```

Description

This is macro GFX_X11_THISTLE.

5.2.2.7.26.554 GFX_X11_TOMATO Macro

C

```
#define GFX_X11_TOMATO GFX_RGBConvert(0xFF, 0x63, 0x47)
```

Description

This is macro GFX_X11_TOMATO.

5.2.2.7.26.555 GFX_X11_TURQUOISE Macro

C

```
#define GFX_X11_TURQUOISE GFX_RGBConvert(0x40, 0xE0, 0xD0)
```

Description

This is macro GFX_X11_TURQUOISE.

5.2.2.7.26.556 GFX_X11_VIOLET Macro

C

```
#define GFX_X11_VIOLET GFX_RGBConvert(0xEE, 0x82, 0xEE)
```

Description

This is macro GFX_X11_VIOLET.

5.2.2.7.26.557 GFX_X11_WHEAT Macro

C

```
#define GFX_X11_WHEAT GFX_RGBConvert(0xF5, 0xDE, 0xB3)
```

Description

This is macro GFX_X11_WHEAT.

5.2.2.7.26.558 GFX_X11_WHITE Macro

C

```
#define GFX_X11_WHITE GFX_RGBConvert(0xFF, 0xFF, 0xFF)
```

Description

X11: [WHITE](#)/GRAY/[BLACK](#) COLORS

5.2.2.7.26.559 GFX_X11_WHITE_SMOKE Macro

C

```
#define GFX_X11_WHITE_SMOKE GFX_RGBConvert(0xF5, 0xF5, 0xF5)
```

Description

This is macro GFX_X11_WHITE_SMOKE.

5.2.2.7.26.560 GFX_X11_YELLOW Macro

C

```
#define GFX_X11_YELLOW GFX_RGBConvert(0xFF, 0xFF, 0x00)
```


Description

X11: Yellow Colors

5.2.2.7.26.561 GFX_X11_YELLOW_GREEN Macro

C

```
#define GFX_X11_YELLOW_GREEN GFX_RGBConvert(0x9A, 0xCD, 0x32)
```

Description

This is macro GFX_X11_YELLOW_GREEN.

5.2.2.7.26.562 GFX_XCHAR Macro

C

```
#define GFX_XCHAR char
```

Description

GFX_CONFIG_FONT_CHAR_SIZE

This configuration sets the text character size used in the library.

To enable support for unicode fonts, GFX_CONFIG_FONT_CHAR_SIZE must be defined as 16 (size is 16 bits). For standard ascii fonts, this can be defined as 8. This changes the GFX_XCHAR definition. See GFX_XCHAR for details.

Set in configuration	GFX_XCHAR	Description
#define GFX_CONFIG_FONT_CHAR_SIZE 16	#define GFX_XCHAR uint16_t	Use multiuint8_t characters (0-2^16 range).
#define GFX_CONFIG_FONT_CHAR_SIZE 8	#define GFX_XCHAR uint8_t	Use multiuint8_t characters (0-255 range).

Remarks

None.

5.2.2.7.26.563 GOLD Macro

C

```
#define GOLD GFX_RGBConvert(255, 215, 0)
```

Description

This is macro GOLD.

5.2.2.7.26.564 GRAPHICS_LIBRARY_VERSION Macro

C

```
#define GRAPHICS_LIBRARY_VERSION 0x0400
```

Description

////////// GRAPHICS_LIBRARY_VERSION ////////// MSB is version, LSB is subversion

5.2.2.7.26.565 GRAY000 Macro**C**

```
#define GRAY000 0
```

Description

This is macro GRAY000.

5.2.2.7.26.566 GRAY001 Macro**C**

```
#define GRAY001 1
```

Description

This is macro GRAY001.

5.2.2.7.26.567 GRAY002 Macro**C**

```
#define GRAY002 2
```

Description

This is macro GRAY002.

5.2.2.7.26.568 GRAY003 Macro**C**

```
#define GRAY003 3
```

Description

This is macro GRAY003.

5.2.2.7.26.569 GRAY004 Macro**C**

```
#define GRAY004 4
```

Description

This is macro GRAY004.

5.2.2.7.26.570 GRAY005 Macro**C**

```
#define GRAY005 5
```

Description

This is macro GRAY005.

5.2.2.7.26.571 GRAY006 Macro**C**

```
#define GRAY006 6
```

Description

This is macro GRAY006.

5.2.2.7.26.572 GRAY007 Macro

C

```
#define GRAY007 7
```

Description

This is macro GRAY007.

5.2.2.7.26.573 GRAY008 Macro

C

```
#define GRAY008 8
```

Description

This is macro GRAY008.

5.2.2.7.26.574 GRAY009 Macro

C

```
#define GRAY009 9
```

Description

This is macro GRAY009.

5.2.2.7.26.575 GRAY010 Macro

C

```
#define GRAY010 GFX_RGBConvert(10, 10, 10)
```

Description

This is macro GRAY010.

5.2.2.7.26.576 GRAY011 Macro

C

```
#define GRAY011 11
```

Description

This is macro GRAY011.

5.2.2.7.26.577 GRAY012 Macro

C

```
#define GRAY012 12
```

Description

This is macro GRAY012.

5.2.2.7.26.578 GRAY013 Macro

C

```
#define GRAY013 13
```

Description

This is macro GRAY013.

5.2.2.7.26.579 GRAY014 Macro

C

```
#define GRAY014 14
```

Description

This is macro GRAY014.

5.2.2.7.26.580 GRAY015 Macro

C

```
#define GRAY015 15
```

Description

This is macro GRAY015.

5.2.2.7.26.581 GRAY032 Macro

C

```
#define GRAY032 GFX_RGBConvert(32, 32, 32)
```

Description

This is macro GRAY032.

5.2.2.7.26.582 GRAY096 Macro

C

```
#define GRAY096 GFX_RGBConvert(96, 96, 96)
```

Description

This is macro GRAY096.

5.2.2.7.26.583 GRAY128 Macro

C

```
#define GRAY128 GFX_RGBConvert(128, 128, 128)
```

Description

This is macro GRAY128.

5.2.2.7.26.584 GRAY160 Macro

C

```
#define GRAY160 GFX_RGBConvert(160, 160, 160)
```

Description

This is macro GRAY160.

5.2.2.7.26.585 GRAY192 Macro

C

```
#define GRAY192 GFX_RGBConvert(192, 192, 192)
```

Description

This is macro GRAY192.

5.2.2.7.26.586 GRAY204 Macro

C

```
#define GRAY204 GFX_RGBConvert(204, 204, 204)
```

Description

This is macro GRAY204.

5.2.2.7.26.587 GRAY224 Macro

C

```
#define GRAY224 GFX_RGBConvert(224, 224, 224)
```

Description

This is macro GRAY224.

5.2.2.7.26.588 GRAY229 Macro

C

```
#define GRAY229 GFX_RGBConvert(229, 229, 229)
```

Description

This is macro GRAY229.

5.2.2.7.26.589 GRAY242 Macro

C

```
#define GRAY242 GFX_RGBConvert(242, 242, 242)
```

Description

This is macro GRAY242.

5.2.2.7.26.590 GREEN Macro

C

```
#define GREEN GFX_RGBConvert(0, 128, 0)
```

Description

This is macro GREEN.

5.2.2.7.26.591 GRID_DISABLED Macro

C

```
#define GRID_DISABLED 0x0002 // Bit for disabled state
```

Description

Bit for disabled state

5.2.2.7.26.592 GRID_DRAW_ALL Macro

C

```
#define GRID_DRAW_ALL 0x4000 // Bit to indicate whole edit box must be redrawn
```

Description

Bit to indicate whole edit box must be redrawn

5.2.2.7.26.593 GRID_DRAW_ITEMS Macro

C

```
#define GRID_DRAW_ITEMS 0x1000 // Bit to indicate that at least one item must be redrawn, but not the entire grid.
```

Description

Bit to indicate that at least one item must be redrawn, but not the entire grid.

5.2.2.7.26.594 GRID_FOCUSED Macro

C

```
#define GRID_FOCUSED 0x0001 // Bit for focused state
```

Description

Bit for focused state

5.2.2.7.26.595 GRID_HEIGHT Macro

C

```
#define GRID_HEIGHT(r, ch) ((GFX_OBJ_EMOSS_SIZE * 2) + (r * (ch + 1)) - 1)
```

Description

This is macro GRID_HEIGHT.

5.2.2.7.26.596 GRID_HIDE Macro

C

```
#define GRID_HIDE 0x8000 // Bit to remove object from screen
```

Description

Bit to remove object from screen

5.2.2.7.26.597 GRID_MAX_COLUMNS Macro

C

```
#define GRID_MAX_COLUMNS(rw, cw) ((rw - (GFX_OBJ_EMOSS_SIZE * 2) + 1) / (cw + 1))
```

Description

This is macro GRID_MAX_COLUMNS.

5.2.2.7.26.598 GRID_MAX_ROWS Macro

C

```
#define GRID_MAX_ROWS(rh, ch) ((rh - (GFX_OBJ_EMOSS_SIZE * 2) + 1) / (ch + 1))
```

Description

This is macro GRID_MAX_ROWS.

5.2.2.7.26.599 GRID_OUT_OF_BOUNDS Macro

C

```
#define GRID_OUT_OF_BOUNDS 0x0001 // Status of an out of bounds cell GridSetCell() operation.
```

Description

Status of an out of bounds cell [GridSetCell\(\)](#) operation.

5.2.2.7.26.600 GRID_SHOW_BORDER_ONLY Macro

C

```
#define GRID_SHOW_BORDER_ONLY 0x0010 // Draw only the outside border of the grid.
```

Description

Draw only the outside border of the grid.

5.2.2.7.26.601 GRID_SHOW_FOCUS Macro

C

```
#define GRID_SHOW_FOCUS 0x0008 // Highlight the focused cell.
```

Description

Highlight the focused cell.

5.2.2.7.26.602 GRID_SHOW_LINES Macro

C

```
#define GRID_SHOW_LINES 0x0004 // Display grid lines
```

Description

Display grid lines

5.2.2.7.26.603 GRID_SHOW_SEPARATORS_ONLY Macro

C

```
#define GRID_SHOW_SEPARATORS_ONLY 0x0020 // Draw only the lines between cells (like
```

Tic-tac-toe)

Description

Draw only the lines between cells (like Tic-tac-toe)

5.2.2.7.26.604 GRID_SUCCESS Macro

```
C
#define GRID_SUCCESS 0x0000    // Status of a successful GridSetCell() operation.
```

Description

Status of a successful [GridSetCell\(\)](#) operation.

5.2.2.7.26.605 GRID_TYPE_MASK Macro

```
C
#define GRID_TYPE_MASK (0x0C)
```

Description

This is macro GRID_TYPE_MASK.

5.2.2.7.26.606 GRID_WIDTH Macro

```
C
#define GRID_WIDTH(c, cw) ((GFX_OBJ_EMOSS_SIZE * 2) + (c * (cw + 1)) - 1)
```

Description

This is macro GRID_WIDTH.

5.2.2.7.26.607 GridGetFocusX Macro

```
C
#define GridGetFocusX(pGrid) pGrid->focusX
```

Description

Macros: GridGetFocusX(pGrid)

This macro returns the x position of the focused cell.

Preconditions

none

Parameters

Parameters	Description
pGrid	Pointer to the object.

Returns

Returns the x position of the focused cell.

Side Effects

none

5.2.2.7.26.608 GridGetFocusY Macro

C

```
#define GridGetFocusY(pGrid) pGrid->focusY
```

Description

Macros: GridGetFocusY(pGrid)

This macro returns the y position of the focused cell.

Preconditions

none

Parameters

Parameters	Description
pGrid	Pointer to the object.

Returns

Returns the y position of the focused cell.

Side Effects

none

5.2.2.7.26.609 GRIDITEM_DRAW Macro

C

```
#define GRIDITEM_DRAW 0x0100 // Draw this cell
```

Description

Draw this cell

5.2.2.7.26.610 GRIDITEM_IS_BITMAP Macro

C

```
#define GRIDITEM_IS_BITMAP 0x0008 // The grid item is a bitmap.
```

Description

The grid item is a bitmap.

5.2.2.7.26.611 GRIDITEM_IS_TEXT Macro

C

```
#define GRIDITEM_IS_TEXT 0x0000 // The grid item is a text string.
```

Description

The grid item is a text string.

5.2.2.7.26.612 GRIDITEM_SELECTED Macro

C

```
#define GRIDITEM_SELECTED 0x0001 // The cell is selected.
```

Description

The cell is selected.

5.2.2.7.26.613 GRIDITEM_TEXTBOTTOM Macro

C

```
#define GRIDITEM_TEXTBOTTOM 0x0040 // Bit to indicate text is top aligned.
```

Description

Bit to indicate text is top aligned.

5.2.2.7.26.614 GRIDITEM_TEXTLEFT Macro

C

```
#define GRIDITEM_TEXTLEFT 0x0020 // Text in the cell is left aligned.
```

Description

Text in the cell is left aligned.

5.2.2.7.26.615 GRIDITEM_TEXTRIGHT Macro

C

```
#define GRIDITEM_TEXTRIGHT 0x0010 // Text in the cell is right aligned.
```

Description

Text in the cell is right aligned.

5.2.2.7.26.616 GRIDITEM_TEXTTOP Macro

C

```
#define GRIDITEM_TEXTTOP 0x0080 // Bit to indicate text is bottom aligned.
```

Description

Bit to indicate text is bottom aligned.

5.2.2.7.26.617 HIDE_DATA Macro

C

```
#define HIDE_DATA 0 // Macro used to reset the data series show flag or indicate  
that the data series will be not be shown when the chart is drawn.
```

Description

Macro used to reset the data series show flag or indicate that the data series will be not be shown when the chart is drawn.

5.2.2.7.26.618 ImageAbort Macro

C

```
#define ImageAbort IMG_bAbortImageDecoding = 1;
```

Description

This function sets the Image Decoder's Abort flag so that decoding aborts in the next decoding loop.

Returns

None

Side Effects

None

Example

```

void callback(void);
void main(void)
{
    IMG_FILE pImageFile;
    IMG_vInitialize();
    ImageLoopCallbackRegister(callback);
    pImageFile = IMG_FOPEN("Image.jpg", "r");
    if(pImageFile == NULL)
    {
        <- Error handling ->
    }
    IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
    IMG_FCLOSE(pImageFile);
    while(1);
}

void callback(void)
{
    if(<- check for abort condition ->)
    {
        ImageAbort();
    }
}

```

5.2.2.7.26.619 ImageFullScreenDecode Macro**C**

```

#define ImageFullScreenDecode(pImageFile, eImgFormat, pFileAPIs, pPixelFormat) \
    ImageDecode(pImageFile, eImgFormat, 0, 0, IMG_SCREEN_WIDTH, IMG_SCREEN_HEIGHT,
    (IMG_ALIGN_CENTER | IMG_DOWN_SCALE), pFileAPIs, pPixelFormat)

```

Description

This function decodes and displays the image on the screen in fullscreen mode with center aligned and downscaled if required

Returns

Error code -> 0 means no error

Side Effects

None

Example

```

void main(void)
{
    IMG_FILE pImageFile;
    IMG_vInitialize();
    pImageFile = IMG_FOPEN("Image.jpg", "r");
    if(pImageFile == NULL)
    {
        <- Error handling ->
    }
}

```

```
    IMG_bFullScreenDecode(plmageFile, IMG_JPEG, NULL, NULL);  
    IMG_FCLOSE(plmageFile);  
    while(1);  
}
```

5.2.2.7.26.620 IMG_ALIGN_CENTER Macro

C

```
#define IMG_ALIGN_CENTER 0x0001
```

Description

Flags

5.2.2.7.26.621 IMG_DECODE_ABORTED Macro

C

```
#define IMG_DECODE_ABORTED 0xFF
```

Description

This is macro IMG_DECODE_ABORTED.

5.2.2.7.26.622 IMG_DOWN_SCALE Macro

C

```
#define IMG_DOWN_SCALE 0x0002
```

Description

This is macro IMG_DOWN_SCALE.

5.2.2.7.26.623 IMG_FCLOSE Macro

C

```
#define IMG_FCLOSE FSfclose
```

Description

This is macro IMG_FCLOSE.

5.2.2.7.26.624 IMG_FEOF Macro

C

```
#define IMG_FEOF IMG_pFileAPIs->pFeof
```

Description

This is macro IMG_FEOF.

5.2.2.7.26.625 IMG_FILE Macro

C

```
#define IMG_FILE void
```

Description

This is macro IMG_FILE.

5.2.2.7.26.626 IMG_FOPEN Macro

C

```
#define IMG_FOPEN FSfopen
```

Description

added by Paolo 9/10/08 for MultiApp Demo

5.2.2.7.26.627 IMG_FREAD Macro

C

```
#define IMG_FREAD IMG_pFileAPIs->pFread
```

Description

This is macro IMG_FREAD.

5.2.2.7.26.628 IMG_FSEEK Macro

C

```
#define IMG_FSEEK IMG_pFileAPIs->pFseek
```

Description

This is macro IMG_FSEEK.

5.2.2.7.26.629 IMG_FTELL Macro

C

```
#define IMG_FTELL IMG_pFileAPIs->pFtell
```

Description

This is macro IMG_FTELL.

5.2.2.7.26.630 IMG_SCREEN_HEIGHT Macro

C

```
#define IMG_SCREEN_HEIGHT (GetMaxY()+1)
```

Description

This is macro IMG_SCREEN_HEIGHT.

5.2.2.7.26.631 IMG_SCREEN_WIDTH Macro

C

```
#define IMG_SCREEN_WIDTH (GetMaxX()+1)
```

Description

The individual image decoders use these defines instead of directly using those provided in the Display driver file

5.2.2.7.26.632 IMG_vCheckAndAbort Macro

C

```
#define IMG_vCheckAndAbort \
```

```

    if(IMG_bAbortImageDecoding == 1) \
    {
        IMG_bAbortImageDecoding = 0; \
        return IMG_DECODE_ABORTED; \
    }

```

Description

This is macro IMG_vCheckAndAbort.

5.2.2.7.26.633 IMG_vLoopCallback Macro

C

```
#define IMG_vLoopCallback
```

Description

This is macro IMG_vLoopCallback.

5.2.2.7.26.634 IMG_vPutPixel Macro

C

```

#define IMG_vPutPixel(x, y) if(IMG_bDownScalingFactor <= 1)\
    {
        if((x) < IMG_wImageWidth && (y) < IMG_wImageHeight) {
            _PutPixel(IMG_wStartX + (x), IMG_wStartY + (y)); }\
        }
        else if((x)%IMG_bDownScalingFactor == 0 &&
            (y)%IMG_bDownScalingFactor == 0) \
        {
            if((x) < IMG_wImageWidth && (y) < IMG_wImageHeight) {
                _PutPixel(IMG_wStartX + ((x)/IMG_bDownScalingFactor), IMG_wStartY +
                    ((y)/IMG_bDownScalingFactor)); }\
            }
        }

```

Description

This is macro IMG_vPutPixel.

5.2.2.7.26.635 IMG_vSetColor Macro

C

```
#define IMG_vSetColor(handle, r, g, b) GFX_PRIM_SetColor(handle, RGBConvert((r), (g), (b)))
```

Description

This is macro IMG_vSetColor.

5.2.2.7.26.636 KHAKI Macro

C

```
#define KHAKI GFX_RGBConvert(240, 230, 140)
```

Description

This is macro KHAKI.

5.2.2.7.26.637 LIGHTBLUE Macro

C

```
#define LIGHTBLUE GFX_RGBConvert(128, 128, 255)
```

Description

This is macro LIGHTBLUE.

5.2.2.7.26.638 LIGHTCYAN Macro

C

```
#define LIGHTCYAN GFX_RGBConvert(128, 255, 255)
```

Description

This is macro LIGHTCYAN.

5.2.2.7.26.639 LIGHTGRAY Macro

C

```
#define LIGHTGRAY GFX_RGBConvert(128, 128, 128)
```

Description

This is macro LIGHTGRAY.

5.2.2.7.26.640 LIGHTGREEN Macro

C

```
#define LIGHTGREEN GFX_RGBConvert(128, 255, 128)
```

Description

This is macro LIGHTGREEN.

5.2.2.7.26.641 LIGHTMAGENTA Macro

C

```
#define LIGHTMAGENTA GFX_RGBConvert(255, 128, 255)
```

Description

This is macro LIGHTMAGENTA.

5.2.2.7.26.642 LIGHTORANGE Macro

C

```
#define LIGHTORANGE GFX_RGBConvert(255, 200, 0)
```

Description

This is macro LIGHTORANGE.

5.2.2.7.26.643 LIGHTRED Macro

C

```
#define LIGHTRED GFX_RGBConvert(255, 128, 128)
```

Description

This is macro LIGHTRED.

5.2.2.7.26.644 LIGHTYELLOW Macro

C

```
#define LIGHTYELLOW GFX_RGBConvert(255, 255, 150)
```

Description

This is macro LIGHTYELLOW.

5.2.2.7.26.645 MAGENTA Macro

C

```
#define MAGENTA GFX_RGBConvert(128, 0, 128)
```

Description

This is macro MAGENTA.

5.2.2.7.26.646 MCHP_BITMAP_NORMAL Macro

C

```
#define MCHP_BITMAP_NORMAL 0x00 // no compression, palette is present
```

Description

no compression, palette is present for color depth = 8, 4 and 1 BPP

5.2.2.7.26.647 MCHP_BITMAP_PALETTE_STR Macro

C

```
#define MCHP_BITMAP_PALETTE_STR 0x10 // palette is provided as a separate
```

Description

palette is provided as a separate object (see [PALETTE_HEADER](#)) for color depth = 8, 4, and 1 BPP, ID to the palette is embedded in the bitmap.

5.2.2.7.26.648 ONEP25 Macro

C

```
#define ONEP25 81920 // 1.25 * 2^16
```

Description

1.25 * 2¹⁶

5.2.2.7.26.649 ORANGE Macro

C

```
#define ORANGE GFX_RGBConvert(255, 187, 76)
```

Description

This is macro ORANGE.

5.2.2.7.26.650 PERU Macro

C

```
#define PERU GFX_RGBConvert(205, 133, 63)
```

Description

This is macro PERU.

5.2.2.7.26.651 RED Macro

C

```
#define RED GFX_RGBConvert(128, 0, 0)
```

Description

This is macro RED.

5.2.2.7.26.652 RequestEntirePaletteChange Macro

C

```
#define RequestEntirePaletteChange(pPalette) RequestPaletteChange(pPalette, 0, 256)
```

Description

Macro: RequestEntirePaletteChange(pPalette)

Loads all the palette entries from the flash during vertical blanking period if possible, otherwise loads immediately.

Preconditions

PPalette must be initialized by [PaletteInit\(\)](#).

Parameters

Parameters	Description
pPalette	Pointer to the palette structure

Returns

none

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

5.2.2.7.26.653 SADDLEBROWN Macro

C

```
#define SADDLEBROWN GFX_RGBConvert(139, 69, 19)
```

Description

This is macro SADDLEBROWN.

5.2.2.7.26.654 SCAN_BS_PRESSED Macro

C

```
#define SCAN_BS_PRESSED 0x0E
```

Description

Back space key pressed.

5.2.2.7.26.655 SCAN_BS_RELEASED Macro**C**

```
#define SCAN_BS_RELEASED 0x8E
```

Description

Back space key released.

5.2.2.7.26.656 SCAN_CR_PRESSED Macro**C**

```
#define SCAN_CR_PRESSED 0x1C
```

Description

Carriage return pressed.

5.2.2.7.26.657 SCAN_CR_RELEASED Macro**C**

```
#define SCAN_CR_RELEASED 0x9C
```

Description

Carriage return released.

5.2.2.7.26.658 SCAN_CRA_PRESSED Macro**C**

```
#define SCAN_CRA_PRESSED 0x2C
```

Description

Carriage return alternate pressed.

5.2.2.7.26.659 SCAN_CRA_RELEASED Macro**C**

```
#define SCAN_CRA_RELEASED 0xAC
```

Description

Carriage return alternate released.

5.2.2.7.26.660 SCAN_DEL_PRESSED Macro**C**

```
#define SCAN_DEL_PRESSED 0x53
```

Description

Delete key pressed.

5.2.2.7.26.661 SCAN_DEL_RELEASED Macro

C

```
#define SCAN_DEL_RELEASED 0xD3
```

Description

Delete key released.

5.2.2.7.26.662 SCAN_DOWN_PRESSED Macro

C

```
#define SCAN_DOWN_PRESSED 0x50
```

Description

Down key pressed.

5.2.2.7.26.663 SCAN_DOWN_RELEASED Macro

C

```
#define SCAN_DOWN_RELEASED 0xD0
```

Description

Down key released.

5.2.2.7.26.664 SCAN_END_PRESSED Macro

C

```
#define SCAN_END_PRESSED 0x4F
```

Description

End key pressed.

5.2.2.7.26.665 SCAN_END_RELEASED Macro

C

```
#define SCAN_END_RELEASED 0xCF
```

Description

End key released.

5.2.2.7.26.666 SCAN_HOME_PRESSED Macro

C

```
#define SCAN_HOME_PRESSED 0x47
```

Description

Home key pressed.

5.2.2.7.26.667 SCAN_HOME_RELEASED Macro

C

```
#define SCAN_HOME_RELEASED 0xC7
```

Description

Home key released.

5.2.2.7.26.668 SCAN_LEFT_PRESSED Macro

C

```
#define SCAN_LEFT_PRESSED 0x4B
```

Description

Left key pressed.

5.2.2.7.26.669 SCAN_LEFT_RELEASED Macro

C

```
#define SCAN_LEFT_RELEASED 0xCB
```

Description

Left key released.

5.2.2.7.26.670 SCAN_PGDOWN_PRESSED Macro

C

```
#define SCAN_PGDOWN_PRESSED 0x51
```

Description

Page down key pressed.

5.2.2.7.26.671 SCAN_PGDOWN_RELEASED Macro

C

```
#define SCAN_PGDOWN_RELEASED 0xD1
```

Description

Page down key released.

5.2.2.7.26.672 SCAN_PGUP_PRESSED Macro

C

```
#define SCAN_PGUP_PRESSED 0x49
```

Description

Page up key pressed.

5.2.2.7.26.673 SCAN_PGUP_RELEASED Macro

C

```
#define SCAN_PGUP_RELEASED 0xC9
```

Description

Page up key released.

5.2.2.7.26.674 SCAN_RIGHT_PRESSED Macro

C

```
#define SCAN_RIGHT_PRESSED 0x4D
```

Description

Right key pressed.

5.2.2.7.26.675 SCAN_RIGHT_RELEASED Macro

C

```
#define SCAN_RIGHT_RELEASED 0xCD
```

Description

Right key released.

5.2.2.7.26.676 SCAN_SPACE_PRESSED Macro

C

```
#define SCAN_SPACE_PRESSED 0x39
```

Description

Space key pressed.

5.2.2.7.26.677 SCAN_SPACE_RELEASED Macro

C

```
#define SCAN_SPACE_RELEASED 0xB9
```

Description

Space key released.

5.2.2.7.26.678 SCAN_TAB_PRESSED Macro

C

```
#define SCAN_TAB_PRESSED 0x0F
```

Description

Tab key pressed.

5.2.2.7.26.679 SCAN_TAB_RELEASED Macro

C

```
#define SCAN_TAB_RELEASED 0x8F
```

Description

Tab key released.

5.2.2.7.26.680 SCAN_UP_PRESSED Macro

C

```
#define SCAN_UP_PRESSED 0x48
```

Description

Up key pressed.

5.2.2.7.26.681 SCAN_UP_RELEASED Macro**C**

```
#define SCAN_UP_RELEASED 0xC8
```

Description

Up key released.

5.2.2.7.26.682 SetEntirePalette Macro**C**

```
#define SetEntirePalette(pPalette) SetPalette(pPalette, 0, 256)
```

Description

Macro: SetEntirePalette(pPalette)

Programs the whole 256 entry palette with new color values from flash.

Preconditions

Palette must be initialized by [PaletteInit\(\)](#).

Parameters

Parameters	Description
pPalette	Pointer to the palette structure

Returns

Returns the status of the palette set. 0 - Set was successful 1 - Set was not successful

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

5.2.2.7.26.683 SHOW_DATA Macro**C**

```
#define SHOW_DATA 1 // Macro used to set the data series show flag or indicate  
that the data series will be shown when the chart is drawn.
```

Description

Macro used to set the data series show flag or indicate that the data series will be shown when the chart is drawn.

5.2.2.7.26.684 SIENNA Macro**C**

```
#define SIENNA GFX_RGBConvert(160, 82, 45)
```

Description

This is macro SIENNA.

5.2.2.7.26.685 SIN45 Macro

C

```
#define SIN45 46341 // sin(45) * 2^16)
```

Description

$\sin(45) * 2^{16}$

5.2.2.7.26.686 TAN Macro

C

```
#define TAN GFX_RGBConvert(210, 180, 140)
```

Description

This is macro TAN.

5.2.2.7.26.687 TE_ROUNDEDBUTTON_RADIUS Macro

C

```
#define TE_ROUNDEDBUTTON_RADIUS 0
```

Description

Optional settings Set this parameter in the GraphicsConfig.h

5.2.2.7.26.688 USE_HORZ_ASCENDING_ORDER Macro

C

```
#define USE_HORZ_ASCENDING_ORDER
```

Description

use ascending order when displaying variables on horizontal mode (bar chart only)

5.2.2.7.26.689 USE_PIE_ENABLE_LABEL Macro

C

```
#define USE_PIE_ENABLE_LABEL
```

Description

use labels: A:10%,30 where each sample is labeled A-Z followed by a colon.

5.2.2.7.26.690 WHEAT Macro

C

```
#define WHEAT GFX_RGBConvert(245, 245, 220)
```

Description

This is macro WHEAT.

5.2.2.7.26.691 WHITE Macro

C

```
#define WHITE GFX_RGBConvert(255, 255, 255)
```

Description

This is macro WHITE.

5.2.2.7.26.692 YELLOW Macro**C**

```
#define YELLOW GFX_RGBConvert(255, 255, 128)
```

Description

This is macro YELLOW.

5.2.2.7.26.693 GFX_GOL_EDITBOX_STATE Enumeration**C**

```
typedef enum {
    GFX_GOL_EDITBOX_FOCUSED_STATE = 0x0001,
    GFX_GOL_EDITBOX_DISABLED_STATE = 0x0002,
    GFX_GOL_EDITBOX_ENABLE_CARET_STATE = 0x0004,
    GFX_GOL_EDITBOX_DRAW_CARET_STATE = 0x1000,
    GFX_GOL_EDITBOX_DRAW_FOCUS_STATE = 0x2000,
    GFX_GOL_EDITBOX_DRAW_STATE = 0x4000,
    GFX_GOL_EDITBOX_HIDE_STATE = 0x8000
} GFX_GOL_EDITBOX_STATE;
```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_EDITBOX_FOCUSED_STATE = 0x0001	Bit for focus state. Cursor caret will be drawn when GFX_GOL_EDITBOX_ENABLE_CARET_STATE is also set.
GFX_GOL_EDITBOX_DISABLED_STATE = 0x0002	Bit for disabled state.
GFX_GOL_EDITBOX_ENABLE_CARET_STATE = 0x0004	Bit to indicate cursor caret will always be shown.
GFX_GOL_EDITBOX_DRAW_CARET_STATE = 0x1000	Bit to indicate the cursor caret will be drawn if GFX_GOL_EDITBOX_FOCUSED_STATE state bit is set and erase when GFX_GOL_EDITBOX_FOCUSED_STATE state bit is not set.
GFX_GOL_EDITBOX_DRAW_FOCUS_STATE = 0x2000	Bit to indicate focus must be redrawn.
GFX_GOL_EDITBOX_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_EDITBOX_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.694 GFX_GOL_PICTURECONTROL Structure**C**

```
typedef struct {
    GFX_GOL_OBJ_HEADER hdr;
    GFX_RESOURCE_HDR * pImage;
    int8_t scaleFactor;
    uint16_t imageLeft;
    uint16_t imageTop;
    uint16_t imageRight;
    uint32_t * stream;
    uint8_t count;
```



```

uint8_t delay;
uint16_t imageBottom;
GFX_PARTIAL_IMAGE_PARAM partial;
} GFX_GOL_PICTURECONTROL;

```

Description

The structure contains data for picture control

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
GFX_RESOURCE_HDR * pImage;	Pointer to the image
int8_t scaleFactor;	Scale factor for the bitmap
uint16_t imageLeft;	image left position when drawn
uint16_t imageTop;	image top position when drawn
uint16_t imageRight;	image right position when drawn
uint8_t count;	Count for the number of bitmaps to be streamed
uint8_t delay;	Delay in between the streaming of bitmaps
uint16_t imageBottom;	image bottom position when drawn
GFX_PARTIAL_IMAGE_PARAM partial;	structure containing partial image data

5.2.2.7.26.695 GFX_GOL_PICTURECONTROLCONTROL_STATE Enumeration

C

```

typedef enum {
    GFX_GOL_PICTURECONTROL_DISABLED_STATE = 0x0002,
    GFX_GOL_PICTURECONTROL_FRAME_STATE = 0x0004,
    GFX_GOL_PICTURECONTROL_STREAM_STATE = 0x0008,
    GFX_GOL_PICTURECONTROL_DRAW_STATE = 0x4000,
    GFX_GOL_PICTURECONTROL_HIDE_STATE = 0x8000
} GFX_GOL_PICTURECONTROLCONTROL_STATE;

```

Description

Object States Definition:

Members

Members	Description
GFX_GOL_PICTURECONTROL_DISABLED_STATE = 0x0002	Bit for disabled state.
GFX_GOL_PICTURECONTROL_FRAME_STATE = 0x0004	Bit for state with frame.
GFX_GOL_PICTURECONTROL_STREAM_STATE = 0x0008	Bit to indicate Picture is streaming
GFX_GOL_PICTURECONTROL_DRAW_STATE = 0x4000	Bit to indicate object must be redrawn.
GFX_GOL_PICTURECONTROL_HIDE_STATE = 0x8000	Bit to indicate object must be removed from screen.

5.2.2.7.26.696 GFX_GOL_DigitalMeterTextAlignmentGet Macro

C

```

#define GFX_GOL_DigitalMeterTextAlignmentGet(pObject) \
    (((GFX_GOL_DIGITALMETER *)pObject)->alignment)

```

Description

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The text alignment set in the object. See [GFX_ALIGNMENT](#) for more details.

Example

None.

5.2.2.7.26.697 GFX_GOL_DigitalMeterTextAlignmentSet Macro**C**

```
#define GFX_GOL_DigitalMeterTextAlignmentSet(pObject, align) \  
(((GFX_GOL_DIGITALMETER *)pObject)->alignment = align)
```

Description

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Returns

None.

Example

None.

5.2.2.7.26.698 GFX_GOL_EditBoxTextAlignmentGet Macro**C**

```
#define GFX_GOL_EditBoxTextAlignmentGet(pObject) \  
(((GFX_GOL_EDITBOX *)pObject)->alignment)
```

Description

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The text alignment set in the object. See [GFX_ALIGNMENT](#) for more details.

Example

None.

5.2.2.7.26.699 GFX_GOL_EditBoxTextAlignmentSet Macro**C**

```
#define GFX_GOL_EditBoxTextAlignmentSet(pObject, align) \
    (((GFX_GOL_EDITBOX *)pObject)->alignment = align)
```

Description

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Returns

None.

Example

None.

5.2.2.7.26.700 GFX_GOL_EditBoxTextGet Macro**C**

```
#define GFX_GOL_EditBoxTextGet(pObject) (((GFX_GOL_EDITBOX *)pObject)->pText)
```

Description

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to text string.

Example

None.

5.2.2.7.26.701 GFX_GOL_ListBoxTextAlignmentGet Macro**C**

```
#define GFX_GOL_ListBoxTextAlignmentGet(pObject) \
    (((GFX_GOL_LISTBOX *)pObject)->alignment)
```

Description

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

The text alignment set in the object. See [GFX_ALIGNMENT](#) for more details.

Example

None.

5.2.2.7.26.702 GFX_GOL_ListBoxTextAlignmentSet Macro**C**

```
#define GFX_GOL_ListBoxTextAlignmentSet(pObject, align) \
    (((GFX_GOL_LISTBOX *)pObject)->alignment = align)
```

Description

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Returns

None.

Example

None.

5.2.2.7.26.703 GFX_GOL_PictureControlImageGet Macro

C

```
#define GFX_GOL_PictureControlImageGet(pObject) \
    (((GFX_GOL_PICTURECONTROL*)pObject)->pImage)
```

Description

This function gets the image used when in the pressed state.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.

Returns

Pointer to the image resource.

Example

None.

5.2.2.7.26.704 GFX_GOL_PictureControlImageSet Macro

C

```
#define GFX_GOL_PictureControlImageSet(pObject, pImage) \
    (((GFX_GOL_PICTURECONTROL*)pObject)->pImage = pImage)
```

Description

This function sets the image to be in the object.

Preconditions

Object must exist in memory.

Parameters

Parameters	Description
pObject	pointer to the object.
pImage	pointer to the image resource.

Returns

None.

Example

```
// assume OrigImage and NewImage are valid GFX_RESOURCE_HDR  
// pointers for images
```

```
GFX_RESOURCE_HDR *pOrigIcon = &OrigImage;  
GFX_RESOURCE_HDR *pNewIcon = &NewImage;  
GFX_GOL_PICTURECONTROL *pPicture;  
  
pPicture = GFX_GOL_PictureControlCreate(  
    10,  
    0, 0,  
    GFX_MaxXGet(), GFX_MaxYGet(),  
    GFX_GOL_PICTURECONTROL_DRAW_STATE,
```

```
        pOrigIcon,
        NULL);

// change the image used
GFX_GOL_PictureControlImageSet(pPicture , pNewIcon);
```

5.2.2.7.26.705 GFX_GOL_PictureControlScaleSet Macro

```
C
#define GFX_GOL_PictureControlScaleSet(pPict, scl) pPict->scaleFactor = scl
```

Description
This function sets the scale factor used to render the bitmap used in the object using scl.

Preconditions
None.

Parameters

Parameters	Description
pPict	Pointer to the object
scl	The scale factor that will be used to display the bitmap.

Returns
None.

Remarks
None.

Example
None.

5.2.2.8 Files

Files

Name	Description
gfx.h	The header file joins all header files used in the graphics library and contains compile options and defaults.
gfx_colors.h	The header file contains default color definitions for each color depth.
gfx_colors_w3.h	The header file contains W3 color definitions.
gfx_colors_x11.h	The header file contains X11 color definitions.

5.2.2.8.1 gfx.h

Module for Microchip Graphics Library

This header file includes all the header files required to use the Microchip Graphics Library. Library features and options defined in the Graphics Library configurations will be included in each build.

Macros

	Name	Description
	GRAPHICS_LIBRARY_VERSION	////////////////// GRAPHICS_LIBRARY_VERSION //////////////////// MSB is version, LSB is subversion

File Name

gfx.h

Company

Microchip Technology Inc.

5.2.2.8.2 gfx_colors.h

Module for Microchip Graphics Library

This header file provides default color definitions as well as color conversion macros. Application may or may not use this file. Application may replace this file with new color definitions that fits the application needs.

Macros

	Name	Description
	BLACK	This is macro BLACK.
	BLUE	This is macro BLUE.
	BRIGHTBLUE	This is macro BRIGHTBLUE.
	BRIGHTCYAN	This is macro BRIGHTCYAN.
	BRIGHTGREEN	This is macro BRIGHTGREEN.
	BRIGHTMAGENTA	This is macro BRIGHTMAGENTA.
	BRIGHTRED	This is macro BRIGHTRED.
	BRIGHTYELLOW	This is macro BRIGHTYELLOW.
	BROWN	This is macro BROWN.
	BURLYWOOD	This is macro BURLYWOOD.
	CYAN	This is macro CYAN.
	DARKGRAY	This is macro DARKGRAY.
	DARKORANGE	This is macro DARKORANGE.
	GOLD	This is macro GOLD.
	GRAY000	This is macro GRAY000.
	GRAY001	This is macro GRAY001.
	GRAY002	This is macro GRAY002.
	GRAY003	This is macro GRAY003.
	GRAY004	This is macro GRAY004.
	GRAY005	This is macro GRAY005.
	GRAY006	This is macro GRAY006.
	GRAY007	This is macro GRAY007.
	GRAY008	This is macro GRAY008.
	GRAY009	This is macro GRAY009.
	GRAY010	This is macro GRAY010.
	GRAY011	This is macro GRAY011.
	GRAY012	This is macro GRAY012.
	GRAY013	This is macro GRAY013.
	GRAY014	This is macro GRAY014.
	GRAY015	This is macro GRAY015.
	GRAY032	This is macro GRAY032.
	GRAY096	This is macro GRAY096.

	GRAY128	This is macro GRAY128.
	GRAY160	This is macro GRAY160.
	GRAY192	This is macro GRAY192.
	GRAY204	This is macro GRAY204.
	GRAY224	This is macro GRAY224.
	GRAY229	This is macro GRAY229.
	GRAY242	This is macro GRAY242.
	GREEN	This is macro GREEN.
	LIGHTBLUE	This is macro LIGHTBLUE.
	LIGHTCYAN	This is macro LIGHTCYAN.
	LIGHTGRAY	This is macro LIGHTGRAY.
	LIGHTGREEN	This is macro LIGHTGREEN.
	LIGHTMAGENTA	This is macro LIGHTMAGENTA.
	LIGHTORANGE	This is macro LIGHTORANGE.
	LIGHTRED	This is macro LIGHTRED.
	LIGHTYELLOW	This is macro LIGHTYELLOW.
	MAGENTA	This is macro MAGENTA.
	ORANGE	This is macro ORANGE.
	PERU	This is macro PERU.
	RED	This is macro RED.
	SADDLEBROWN	This is macro SADDLEBROWN.
	SIENNA	This is macro SIENNA.
	TAN	This is macro TAN.
	WHEAT	This is macro WHEAT.
	WHITE	This is macro WHITE.
	YELLOW	This is macro YELLOW.

File Name

gfx_colors.h

Company

Microchip Technology Inc.

5.2.2.8.3 gfx_colors_w3.h

Module for Microchip Graphics Library

This header file provides W3 color definitions as an option of color definitions to use.

Remarks<http://www.w3.org/TR/SVG/types.html#ColorKeywords>**Macros**

	Name	Description
	GFX_W3_ALICEBLUE	This is macro GFX_W3_ALICEBLUE.
	GFX_W3_ANTIQUWHITE	This is macro GFX_W3_ANTIQUWHITE.
	GFX_W3_AQUA	This is macro GFX_W3_AQUA.
	GFX_W3_AQUAMARINE	This is macro GFX_W3_AQUAMARINE.

	<u>GFX_W3_AZURE</u>	This is macro GFX_W3_AZURE.
	<u>GFX_W3_BEIGE</u>	This is macro GFX_W3_BEIGE.
	<u>GFX_W3_BISQUE</u>	This is macro GFX_W3_BISQUE.
	<u>GFX_W3_BLACK</u>	This is macro GFX_W3_BLACK.
	<u>GFX_W3_BLANCHEDALMOND</u>	This is macro GFX_W3_BLANCHEDALMOND.
	<u>GFX_W3_BLUE</u>	This is macro GFX_W3_BLUE.
	<u>GFX_W3_BLUEVIOLET</u>	This is macro GFX_W3_BLUEVIOLET.
	<u>GFX_W3_BROWN</u>	This is macro GFX_W3_BROWN.
	<u>GFX_W3_BURLYWOOD</u>	This is macro GFX_W3_BURLYWOOD.
	<u>GFX_W3_CADETBLUE</u>	This is macro GFX_W3_CADETBLUE.
	<u>GFX_W3_CHARTREUSE</u>	This is macro GFX_W3_CHARTREUSE.
	<u>GFX_W3_CHOCOLATE</u>	This is macro GFX_W3_CHOCOLATE.
	<u>GFX_W3_CORAL</u>	This is macro GFX_W3_CORAL.
	<u>GFX_W3_CORNFLOWERBLUE</u>	This is macro GFX_W3_CORNFLOWERBLUE.
	<u>GFX_W3_CORNSILK</u>	This is macro GFX_W3_CORNSILK.
	<u>GFX_W3_CRIMSON</u>	This is macro GFX_W3_CRIMSON.
	<u>GFX_W3_CYAN</u>	This is macro GFX_W3_CYAN.
	<u>GFX_W3_DARKBLUE</u>	This is macro GFX_W3_DARKBLUE.
	<u>GFX_W3_DARKCYAN</u>	This is macro GFX_W3_DARKCYAN.
	<u>GFX_W3_darkgoldenrod</u>	This is macro GFX_W3_darkgoldenrod.
	<u>GFX_W3_DARKGRAY</u>	This is macro GFX_W3_DARKGRAY.
	<u>GFX_W3_DARKGREEN</u>	This is macro GFX_W3_DARKGREEN.
	<u>GFX_W3_DARKGREY</u>	This is macro GFX_W3_DARKGREY.
	<u>GFX_W3_DARKHAKI</u>	This is macro GFX_W3_DARKHAKI.
	<u>GFX_W3_DARKMAGENTA</u>	This is macro GFX_W3_DARKMAGENTA.
	<u>GFX_W3_DARKOLIVEGREEN</u>	This is macro GFX_W3_DARKOLIVEGREEN.
	<u>GFX_W3_DARKORANGE</u>	This is macro GFX_W3_DARKORANGE.
	<u>GFX_W3_DARKORCHID</u>	This is macro GFX_W3_DARKORCHID.
	<u>GFX_W3_DARKRED</u>	This is macro GFX_W3_DARKRED.
	<u>GFX_W3_DARKSALMON</u>	This is macro GFX_W3_DARKSALMON.
	<u>GFX_W3_DARKSEAGREEN</u>	This is macro GFX_W3_DARKSEAGREEN.
	<u>GFX_W3_DARKSLATEBLUE</u>	This is macro GFX_W3_DARKSLATEBLUE.
	<u>GFX_W3_DARKSLATEGRAY</u>	This is macro GFX_W3_DARKSLATEGRAY.
	<u>GFX_W3_DARKSLATEGREY</u>	This is macro GFX_W3_DARKSLATEGREY.
	<u>GFX_W3_DARKTURQUOISE</u>	This is macro GFX_W3_DARKTURQUOISE.
	<u>GFX_W3_DARKVIOLET</u>	This is macro GFX_W3_DARKVIOLET.
	<u>GFX_W3_DEEPPINK</u>	This is macro GFX_W3_DEEPPINK.
	<u>GFX_W3_DEEPSKYBLUE</u>	This is macro GFX_W3_DEEPSKYBLUE.
	<u>GFX_W3_DIMGRAY</u>	This is macro GFX_W3_DIMGRAY.
	<u>GFX_W3_DIMGREY</u>	This is macro GFX_W3_DIMGREY.
	<u>GFX_W3_DODGERBLUE</u>	This is macro GFX_W3_DODGERBLUE.
	<u>GFX_W3_FIREBRICK</u>	This is macro GFX_W3_FIREBRICK.
	<u>GFX_W3_FLORALWHITE</u>	This is macro GFX_W3_FLORALWHITE.
	<u>GFX_W3_FORESTGREEN</u>	This is macro GFX_W3_FORESTGREEN.
	<u>GFX_W3_FUCHSIA</u>	This is macro GFX_W3_FUCHSIA.

	<u>GFX_W3_GAINSBORO</u>	This is macro GFX_W3_GAINSBORO.
	<u>GFX_W3_GHOSTWHITE</u>	This is macro GFX_W3_GHOSTWHITE.
	<u>GFX_W3_GOLD</u>	This is macro GFX_W3_GOLD.
	<u>GFX_W3_GOLDENROD</u>	This is macro GFX_W3_GOLDENROD.
	<u>GFX_W3_GRAY</u>	This is macro GFX_W3_GRAY.
	<u>GFX_W3_GREEN</u>	This is macro GFX_W3_GREEN.
	<u>GFX_W3_GREENYELLOW</u>	This is macro GFX_W3_GREENYELLOW.
	<u>GFX_W3_GREY</u>	This is macro GFX_W3_GREY.
	<u>GFX_W3_HONEYDEW</u>	This is macro GFX_W3_HONEYDEW.
	<u>GFX_W3_HOTPINK</u>	This is macro GFX_W3_HOTPINK.
	<u>GFX_W3_INDIANRED</u>	This is macro GFX_W3_INDIANRED.
	<u>GFX_W3_INDIGO</u>	This is macro GFX_W3_INDIGO.
	<u>GFX_W3_IVORY</u>	This is macro GFX_W3_IVORY.
	<u>GFX_W3_LAVENDER</u>	This is macro GFX_W3_LAVENDER.
	<u>GFX_W3_LAVENDERBLUSH</u>	This is macro GFX_W3_LAVENDERBLUSH.
	<u>GFX_W3_LAWNGREEN</u>	This is macro GFX_W3_LAWNGREEN.
	<u>GFX_W3_LEMONCHIFFON</u>	This is macro GFX_W3_LEMONCHIFFON.
	<u>GFX_W3_LIGHTBLUE</u>	This is macro GFX_W3_LIGHTBLUE.
	<u>GFX_W3_LIGHTCORAL</u>	This is macro GFX_W3_LIGHTCORAL.
	<u>GFX_W3_LIGHTCYAN</u>	This is macro GFX_W3_LIGHTCYAN.
	<u>GFX_W3_LIGHTGOLDENRODYELLOW</u>	This is macro GFX_W3_LIGHTGOLDENRODYELLOW.
	<u>GFX_W3_LIGHTGRAY</u>	This is macro GFX_W3_LIGHTGRAY.
	<u>GFX_W3_LIGHTGREEN</u>	This is macro GFX_W3_LIGHTGREEN.
	<u>GFX_W3_LIGHTGREY</u>	This is macro GFX_W3_LIGHTGREY.
	<u>GFX_W3_LIGHTPINK</u>	This is macro GFX_W3_LIGHTPINK.
	<u>GFX_W3_LIGHTSALMON</u>	This is macro GFX_W3_LIGHTSALMON.
	<u>GFX_W3_LIGHTSKYBLUE</u>	This is macro GFX_W3_LIGHTSKYBLUE.
	<u>GFX_W3_LIGHTSLATEGRAY</u>	This is macro GFX_W3_LIGHTSLATEGRAY.
	<u>GFX_W3_LIGHTSLATEGREY</u>	This is macro GFX_W3_LIGHTSLATEGREY.
	<u>GFX_W3_LIGHTSTEELBLUE</u>	This is macro GFX_W3_LIGHTSTEELBLUE.
	<u>GFX_W3_LIGHTYELLOW</u>	This is macro GFX_W3_LIGHTYELLOW.
	<u>GFX_W3_LIGHSEAGREEN</u>	This is macro GFX_W3_LIGHSEAGREEN.
	<u>GFX_W3_LIME</u>	This is macro GFX_W3_LIME.
	<u>GFX_W3_LIMEGREEN</u>	This is macro GFX_W3_LIMEGREEN.
	<u>GFX_W3_LINEN</u>	This is macro GFX_W3_LINEN.
	<u>GFX_W3_MAGENTA</u>	This is macro GFX_W3_MAGENTA.
	<u>GFX_W3_MAROON</u>	This is macro GFX_W3_MAROON.
	<u>GFX_W3_MEDIUMAQUAMARINE</u>	This is macro GFX_W3_MEDIUMAQUAMARINE.
	<u>GFX_W3_MEDIUMBLUE</u>	This is macro GFX_W3_MEDIUMBLUE.
	<u>GFX_W3_MEDIUMORCHID</u>	This is macro GFX_W3_MEDIUMORCHID.
	<u>GFX_W3_MEDIUMPURPLE</u>	This is macro GFX_W3_MEDIUMPURPLE.
	<u>GFX_W3_MEDIUMSEAGREEN</u>	This is macro GFX_W3_MEDIUMSEAGREEN.
	<u>GFX_W3_MEDIUMSLATEBLUE</u>	This is macro GFX_W3_MEDIUMSLATEBLUE.
	<u>GFX_W3_MEDIUMSPRINGGREEN</u>	This is macro GFX_W3_MEDIUMSPRINGGREEN.
	<u>GFX_W3_MEDIUMTURQUOISE</u>	This is macro GFX_W3_MEDIUMTURQUOISE.

	<u>GFX_W3_MEDIUMVIOLETRED</u>	This is macro GFX_W3_MEDIUMVIOLETRED.
	<u>GFX_W3_MIDNIGHTBLUE</u>	This is macro GFX_W3_MIDNIGHTBLUE.
	<u>GFX_W3_MINTCREAM</u>	This is macro GFX_W3_MINTCREAM.
	<u>GFX_W3_MISTYROSE</u>	This is macro GFX_W3_MISTYROSE.
	<u>GFX_W3_MOCCASIN</u>	This is macro GFX_W3_MOCCASIN.
	<u>GFX_W3_NAVAJOWHITE</u>	This is macro GFX_W3_NAVAJOWHITE.
	<u>GFX_W3_NAVY</u>	This is macro GFX_W3_NAVY.
	<u>GFX_W3_OLDLACE</u>	This is macro GFX_W3_OLDLACE.
	<u>GFX_W3_OLIVE</u>	This is macro GFX_W3_OLIVE.
	<u>GFX_W3_OLIVEDRAB</u>	This is macro GFX_W3_OLIVEDRAB.
	<u>GFX_W3_ORANGE</u>	This is macro GFX_W3_ORANGE.
	<u>GFX_W3_ORANGERED</u>	This is macro GFX_W3_ORANGERED.
	<u>GFX_W3_ORCHID</u>	This is macro GFX_W3_ORCHID.
	<u>GFX_W3_PALEGOLDENROD</u>	This is macro GFX_W3_PALEGOLDENROD.
	<u>GFX_W3_PALEGREEN</u>	This is macro GFX_W3_PALEGREEN.
	<u>GFX_W3_PALETURQUOISE</u>	This is macro GFX_W3_PALETURQUOISE.
	<u>GFX_W3_PALEVIOLETRED</u>	This is macro GFX_W3_PALEVIOLETRED.
	<u>GFX_W3_PAPAYAWHIP</u>	This is macro GFX_W3_PAPAYAWHIP.
	<u>GFX_W3_PEACHPUFF</u>	This is macro GFX_W3_PEACHPUFF.
	<u>GFX_W3_PERU</u>	This is macro GFX_W3_PERU.
	<u>GFX_W3_PINK</u>	This is macro GFX_W3_PINK.
	<u>GFX_W3_PLUM</u>	This is macro GFX_W3_PLUM.
	<u>GFX_W3_POWDERBLUE</u>	This is macro GFX_W3_POWDERBLUE.
	<u>GFX_W3_PURPLE</u>	This is macro GFX_W3_PURPLE.
	<u>GFX_W3_RED</u>	This is macro GFX_W3_RED.
	<u>GFX_W3_ROSYBROWN</u>	This is macro GFX_W3_ROSYBROWN.
	<u>GFX_W3_ROYALBLUE</u>	This is macro GFX_W3_ROYALBLUE.
	<u>GFX_W3_SADDLEBROWN</u>	This is macro GFX_W3_SADDLEBROWN.
	<u>GFX_W3_SALMON</u>	This is macro GFX_W3_SALMON.
	<u>GFX_W3_SANDYGREEN</u>	This is macro GFX_W3_SANDYGREEN.
	<u>GFX_W3_SEAGREEN</u>	This is macro GFX_W3_SEAGREEN.
	<u>GFX_W3_SEASHELL</u>	This is macro GFX_W3_SEASHELL.
	<u>GFX_W3_SIENNA</u>	This is macro GFX_W3_SIENNA.
	<u>GFX_W3_SILVER</u>	This is macro GFX_W3_SILVER.
	<u>GFX_W3_SKYBLUE</u>	This is macro GFX_W3_SKYBLUE.
	<u>GFX_W3_SLATEBLUE</u>	This is macro GFX_W3_SLATEBLUE.
	<u>GFX_W3_SLATEGRAY</u>	This is macro GFX_W3_SLATEGRAY.
	<u>GFX_W3_SLATEGREY</u>	This is macro GFX_W3_SLATEGREY.
	<u>GFX_W3_SNOW</u>	This is macro GFX_W3_SNOW.
	<u>GFX_W3_SPRINGGREEN</u>	This is macro GFX_W3_SPRINGGREEN.
	<u>GFX_W3_STEELBLUE</u>	This is macro GFX_W3_STEELBLUE.
	<u>GFX_W3_TAN</u>	This is macro GFX_W3_TAN.
	<u>GFX_W3_TEAL</u>	This is macro GFX_W3_TEAL.
	<u>GFX_W3_THISTLE</u>	This is macro GFX_W3_THISTLE.
	<u>GFX_W3_TOMATO</u>	This is macro GFX_W3_TOMATO.

	GFX_W3_TURQUOISE	This is macro GFX_W3_TURQUOISE.
	GFX_W3_VIOLET	This is macro GFX_W3_VIOLET.
	GFX_W3_WHEAT	This is macro GFX_W3_WHEAT.
	GFX_W3_WHITE	This is macro GFX_W3_WHITE.
	GFX_W3_WHITESMOKE	This is macro GFX_W3_WHITESMOKE.
	GFX_W3_YELLOW	This is macro GFX_W3_YELLOW.
	GFX_W3_YELLOWGREEN	This is macro GFX_W3_YELLOWGREEN.
	KHAKI	This is macro KHAKI.

File Name

gfx_colors_w3.h

Company

Microchip Technology Inc.

5.2.2.8.4 gfx_colors_x11.h

Module for Microchip Graphics Library

This header file provides X11 color definitions as an option of color definitions to use.

Macros

	Name	Description
	GFX_X11_ALICE_BLUE	This is macro GFX_X11_ALICE_BLUE.
	GFX_X11_ANTIQUUE_WHITE	This is macro GFX_X11_ANTIQUUE_WHITE.
	GFX_X11_AQUA	This is macro GFX_X11_AQUA.
	GFX_X11_AQUAMARINE	This is macro GFX_X11_AQUAMARINE.
	GFX_X11_AZURE	This is macro GFX_X11_AZURE.
	GFX_X11_BEIGE	This is macro GFX_X11_BEIGE.
	GFX_X11_BISQUE	This is macro GFX_X11_BISQUE.
	GFX_X11_BLACK	This is macro GFX_X11_BLACK.
	GFX_X11_BLANCHED_ALMOND	This is macro GFX_X11_BLANCHED_ALMOND.
	GFX_X11_BLUE	This is macro GFX_X11_BLUE.
	GFX_X11_BLUE_VIOLET	This is macro GFX_X11_BLUE_VIOLET.
	GFX_X11_BROWN	This is macro GFX_X11_BROWN.
	GFX_X11_BURLY_WOOD	This is macro GFX_X11_BURLY_WOOD.
	GFX_X11_CADEL_BLUE	This is macro GFX_X11_CADEL_BLUE.
	GFX_X11_CHARTEUSE	This is macro GFX_X11_CHARTEUSE.
	GFX_X11_CHOCOLATE	This is macro GFX_X11_CHOCOLATE.
	GFX_X11_CORAL	This is macro GFX_X11_CORAL.
	GFX_X11_CORNFLOWER_BLUE	This is macro GFX_X11_CORNFLOWER_BLUE.
	GFX_X11_CORNSILK	X11: Brown Colors
	GFX_X11_CRIMSON	This is macro GFX_X11_CRIMSON.
	GFX_X11_CYAN	This is macro GFX_X11_CYAN.
	GFX_X11_DARK_BLUE	This is macro GFX_X11_DARK_BLUE.
	GFX_X11_DARK_CYAN	This is macro GFX_X11_DARK_CYAN.
	GFX_X11_DARK_GOLDENROD	This is macro GFX_X11_DARK_GOLDENROD.

	GFX_X11_DARK_GREEN	This is macro GFX_X11_DARK_GREEN.
	GFX_X11_DARK_GREY	This is macro GFX_X11_DARK_GREY.
	GFX_X11_DARK_KHAKI	This is macro GFX_X11_DARK_KHAKI.
	GFX_X11_DARK_MAGENTA	This is macro GFX_X11_DARK_MAGENTA.
	GFX_X11_DARK_OLIVE_GREEN	X11: Green Colors
	GFX_X11_DARK_ORANGE	This is macro GFX_X11_DARK_ORANGE.
	GFX_X11_DARK_ORCHID	This is macro GFX_X11_DARK_ORCHID.
	GFX_X11_DARK_RED	This is macro GFX_X11_DARK_RED.
	GFX_X11_DARK_SALMON	This is macro GFX_X11_DARK_SALMON.
	GFX_X11_DARK_SEA_GREEN	This is macro GFX_X11_DARK_SEA_GREEN.
	GFX_X11_DARK_SLATE_BLUE	This is macro GFX_X11_DARK_SLATE_BLUE.
	GFX_X11_DARK_SLATE_GREY	This is macro GFX_X11_DARK_SLATE_GREY.
	GFX_X11_DARK_TURQUOISE	This is macro GFX_X11_DARK_TURQUOISE.
	GFX_X11_DARK_VIOLET	This is macro GFX_X11_DARK_VIOLET.
	GFX_X11_DEEP_PINK	This is macro GFX_X11_DEEP_PINK.
	GFX_X11_DEEP_SKY_BLUE	This is macro GFX_X11_DEEP_SKY_BLUE.
	GFX_X11_DIM_GREY	This is macro GFX_X11_DIM_GREY.
	GFX_X11_DODGER_BLUE	This is macro GFX_X11_DODGER_BLUE.
	GFX_X11_FIRE_BRICK	This is macro GFX_X11_FIRE_BRICK.
	GFX_X11_FLORAL_WHITE	This is macro GFX_X11_FLORAL_WHITE.
	GFX_X11_FOREST_GREEN	This is macro GFX_X11_FOREST_GREEN.
	GFX_X11_FUCHSIA	This is macro GFX_X11_FUCHSIA.
	GFX_X11_GAINSBORO	This is macro GFX_X11_GAINSBORO.
	GFX_X11_GHOST_WHITE	This is macro GFX_X11_GHOST_WHITE.
	GFX_X11_GOLD	This is macro GFX_X11_GOLD.
	GFX_X11_GOLDENROD	This is macro GFX_X11_GOLDENROD.
	GFX_X11_GREEN	This is macro GFX_X11_GREEN.
	GFX_X11_GREEN_YELLOW	This is macro GFX_X11_GREEN_YELLOW.
	GFX_X11_GREY	This is macro GFX_X11_GREY.
	GFX_X11_HONEYDEW	This is macro GFX_X11_HONEYDEW.
	GFX_X11_HOT_PINK	This is macro GFX_X11_HOT_PINK.
	GFX_X11_INDIAN_RED	This is macro GFX_X11_INDIAN_RED.
	GFX_X11_INDIGO	This is macro GFX_X11_INDIGO.
	GFX_X11_IVORY	This is macro GFX_X11_IVORY.
	GFX_X11_KHAKI	This is macro GFX_X11_KHAKI.
	GFX_X11_LAVENDER	X11: BLUE COLORS
	GFX_X11_LAVENDOR_BLUSH	This is macro GFX_X11_LAVENDOR_BLUSH.
	GFX_X11_LAWN_GREEN	This is macro GFX_X11_LAWN_GREEN.
	GFX_X11_LEMON_CHIFFON	This is macro GFX_X11_LEMON_CHIFFON.
	GFX_X11_LIGHT_BLUE	This is macro GFX_X11_LIGHT_BLUE.
	GFX_X11_LIGHT_CAROL	This is macro GFX_X11_LIGHT_CAROL.
	GFX_X11_LIGHT_CYAN	This is macro GFX_X11_LIGHT_CYAN.
	GFX_X11_LIGHT_GOLDENROD_YELLOW	This is macro GFX_X11_LIGHT_GOLDENROD_YELLOW.
	GFX_X11_LIGHT_GRAY	This is macro GFX_X11_LIGHT_GRAY.
	GFX_X11_LIGHT_GREEN	This is macro GFX_X11_LIGHT_GREEN.

	GFX_X11_LIGHT_PINK	This is macro GFX_X11_LIGHT_PINK.
	GFX_X11_LIGHT_SALMON	X11: Red Colors
	GFX_X11_LIGHT_SEA_GREEN	This is macro GFX_X11_LIGHT_SEA_GREEN.
	GFX_X11_LIGHT_SKY_BLUE	This is macro GFX_X11_LIGHT_SKY_BLUE.
	GFX_X11_LIGHT_SLATE_GREY	This is macro GFX_X11_LIGHT_SLATE_GREY.
	GFX_X11_LIGHT_STEEL_BLUE	X11: BLUE COLORS
	GFX_X11_LIGHT_YELLOW	This is macro GFX_X11_LIGHT_YELLOW.
	GFX_X11_LIME	This is macro GFX_X11_LIME.
	GFX_X11_LIME_GREEN	This is macro GFX_X11_LIME_GREEN.
	GFX_X11_LINEN	This is macro GFX_X11_LINEN.
	GFX_X11_MAGENTA	This is macro GFX_X11_MAGENTA.
	GFX_X11_MARRON	This is macro GFX_X11_MARRON.
	GFX_X11_MEDIUM_AQUAMARINE	X11: CYAN COLORS
	GFX_X11_MEDIUM_BLUE	This is macro GFX_X11_MEDIUM_BLUE.
	GFX_X11_MEDIUM_ORCHID	This is macro GFX_X11_MEDIUM_ORCHID.
	GFX_X11_MEDIUM_PURPLE	This is macro GFX_X11_MEDIUM_PURPLE.
	GFX_X11_MEDIUM_SEA_GREEN	This is macro GFX_X11_MEDIUM_SEA_GREEN.
	GFX_X11_MEDIUM_SLATE_BLUE	This is macro GFX_X11_MEDIUM_SLATE_BLUE.
	GFX_X11_MEDIUM_SPRING_GREEN	This is macro GFX_X11_MEDIUM_SPRING_GREEN.
	GFX_X11_MEDIUM_TURQUOISE	This is macro GFX_X11_MEDIUM_TURQUOISE.
	GFX_X11_MEDIUM_VIOLET_RED	This is macro GFX_X11_MEDIUM_VIOLET_RED.
	GFX_X11_MIDNIGHT_BLUE	This is macro GFX_X11_MIDNIGHT_BLUE.
	GFX_X11_MINT_CREAM	This is macro GFX_X11_MINT_CREAM.
	GFX_X11_MISTY_ROSE	This is macro GFX_X11_MISTY_ROSE.
	GFX_X11_MOCCASIN	This is macro GFX_X11_MOCCASIN.
	GFX_X11_NAVAJO_WHITE	This is macro GFX_X11_NAVAJO_WHITE.
	GFX_X11_NAVY	This is macro GFX_X11_NAVY.
	GFX_X11_OLD_LACE	This is macro GFX_X11_OLD_LACE.
	GFX_X11_OLIVE	This is macro GFX_X11_OLIVE.
	GFX_X11_OLIVE_DRAB	This is macro GFX_X11_OLIVE_DRAB.
	GFX_X11_ORANGE	This is macro GFX_X11_ORANGE.
	GFX_X11_ORANGE_RED	X11: Orange Colors
	GFX_X11_ORCHID	This is macro GFX_X11_ORCHID.
	GFX_X11_PALE_GOLDENROD	This is macro GFX_X11_PALE_GOLDENROD.
	GFX_X11_PALE_GREEN	This is macro GFX_X11_PALE_GREEN.
	GFX_X11_PALE_TURQUOISE	This is macro GFX_X11_PALE_TURQUOISE.
	GFX_X11_PALE_VIOLET_RED	This is macro GFX_X11_PALE_VIOLET_RED.
	GFX_X11_PAPAYA_WHIP	This is macro GFX_X11_PAPAYA_WHIP.
	GFX_X11_PEACH_PUFF	This is macro GFX_X11_PEACH_PUFF.
	GFX_X11_PERU	This is macro GFX_X11_PERU.
	GFX_X11_PINK	X11: Pink Colors
	GFX_X11_PLUM	This is macro GFX_X11_PLUM.
	GFX_X11_POWDER_BLUE	This is macro GFX_X11_POWDER_BLUE.
	GFX_X11_PURPLE	This is macro GFX_X11_PURPLE.
	GFX_X11_RED	This is macro GFX_X11_RED.

	GFX_X11_ROSY_BROWN	This is macro GFX_X11_ROSY_BROWN.
	GFX_X11_ROYAL_BLUE	This is macro GFX_X11_ROYAL_BLUE.
	GFX_X11_SADDLE_BROWN	This is macro GFX_X11_SADDLE_BROWN.
	GFX_X11_SALMON	This is macro GFX_X11_SALMON.
	GFX_X11_SANDY_BROWN	This is macro GFX_X11_SANDY_BROWN.
	GFX_X11_SEA_GREEN	This is macro GFX_X11_SEA_GREEN.
	GFX_X11_SEASHELL	This is macro GFX_X11_SEASHELL.
	GFX_X11_SIENNA	This is macro GFX_X11_SIENNA.
	GFX_X11_SILVER	This is macro GFX_X11_SILVER.
	GFX_X11_SKY_BLUE	This is macro GFX_X11_SKY_BLUE.
	GFX_X11_SLATE_BLUE	This is macro GFX_X11_SLATE_BLUE.
	GFX_X11_SLATE_GREY	This is macro GFX_X11_SLATE_GREY.
	GFX_X11_SNOW	This is macro GFX_X11_SNOW.
	GFX_X11_SPRING_GREEN	This is macro GFX_X11_SPRING_GREEN.
	GFX_X11_STEEL_BLUE	This is macro GFX_X11_STEEL_BLUE.
	GFX_X11_TAN	This is macro GFX_X11_TAN.
	GFX_X11_TEAL	This is macro GFX_X11_TEAL.
	GFX_X11_THISTLE	This is macro GFX_X11_THISTLE.
	GFX_X11_TOMATO	This is macro GFX_X11_TOMATO.
	GFX_X11_TURQUOISE	This is macro GFX_X11_TURQUOISE.
	GFX_X11_VIOLET	This is macro GFX_X11_VIOLET.
	GFX_X11_WHEAT	This is macro GFX_X11_WHEAT.
	GFX_X11_WHITE	X11: WHITE /GRAY/ BLACK COLORS
	GFX_X11_WHITE_SMOKE	This is macro GFX_X11_WHITE_SMOKE.
	GFX_X11_YELLOW	X11: Yellow Colors
	GFX_X11_YELLOW_GREEN	This is macro GFX_X11_YELLOW_GREEN.

File Name

gfx_colors_x11.h

Company

Microchip Technology Inc.

5.2.2.8.5 gfx_config_template.h

- Module for Microchip Graphics Library
- This file contains compile time options for the Graphics Library.

- FileName: GraphicsConfig.h
- Dependencies: none
- Processor: PIC24F, PIC24H, dsPIC, PIC32
- Compiler: C30 V3.00/C32
- Company: Microchip Technology, Inc.

*

- Software License Agreement

*

- Copyright © 2008 Microchip Technology Inc. All rights reserved.
- Microchip licenses to you the right to use, modify, copy and distribute
- Software only when embedded on a Microchip microcontroller or digital
- signal controller, which is integrated into your product or third party
- product (pursuant to the sublicense terms in the accompanying license
- agreement).

*

- You should refer to the license agreement accompanying this Software
- for additional information regarding your rights and obligations.

*

- SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
- KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
- OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
- PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
- OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
- BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
- DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
- INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
- COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
- CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
- OR OTHER SIMILAR COSTS.

*

- Author Date Comment

*~-----

- Anton Alkhimenok 10/28/2007 Initial Version
- Pradeep Budagutta 10/28/2007 Display related defines
- moved to DisplayConfig.h

Macros

	Name	Description
	COLOR_DEPTH	Specifies the color depth used in the application defined in GraphicsConfig.h.

5.2.2.8.6 gfx_gol.h

Module for Microchip Graphics Library - Graphic Object Layer

This module implements the common routines for the Graphics Object Layer of the Microchip Graphics Library. The routines are independent of the Display Driver Layer and should be compatible with any Display Driver that is compliant with the requirements of the Display Driver Layer of the Graphics Library. The module utilizes the Graphics Primitive Layer to render the objects.

Enumerations

	Name	Description
	<u>GFX_GOL_OBJ_TYPE</u>	Specifies the different object types used in the library.
	<u>GFX_GOL_TRANSLATED_ACTION</u>	Specifies the different object actions supported in the library.
	<u>INPUT_DEVICE_EVENT</u>	Specifies the different user input device events supported in the library.
	<u>INPUT_DEVICE_TYPE</u>	Specifies the different user input devices supported in the library.

Functions

	Name	Description
	<u>GFX_GOL_DrawCallbackSet</u>	This function sets the draw callback function that the application will use to render application specific rendering.
	<u>GFX_GOL_MessageCallbackSet</u>	This function sets the message callback function that the application will use to evaluate user inputs that affects the objects and application behavior.
	<u>GFX_GOL_ObjectAdd</u>	This function adds an object to the tail of the currently active list.
	<u>GFX_GOL_ObjectBackGroundSet</u>	This function sets the background information.
	<u>GFX_GOL_ObjectByIDDelete</u>	This function removes an object with the given user defined ID from the currently active list.
	<u>GFX_GOL_ObjectCanBeFocused</u>	Checks if the object can be focused.
	<u>GFX_GOL_ObjectDelete</u>	This function removes an object from the currently active list.
	<u>GFX_GOL_ObjectDrawDisable</u>	This function resets the drawing state bits of the object.
	<u>GFX_GOL_ObjectDrawEnable</u>	This function sets the object to be redraw.
	<u>GFX_GOL_ObjectFind</u>	This function returns the pointer to object in the list with the user defined ID assigned to it.
	<u>GFX_GOL_ObjectFocusGet</u>	This function returns the pointer to the object that is currently receiving keyboard input (or focused).
	<u>GFX_GOL_ObjectFocusNextGet</u>	This function returns the pointer to the next object in the active list of objects which can be focused.
	<u>GFX_GOL_ObjectFocusPrevGet</u>	This function returns the pointer to the previous object in the active list of objects which can be focused.
	<u>GFX_GOL_ObjectFocusSet</u>	This function sets the object to be focused.
	<u>GFX_GOL_ObjectHideDraw</u>	This is function <u>GFX_GOL_ObjectHideDraw</u> .
	<u>GFX_GOL_ObjectIDGet</u>	This function returns the object ID.
	<u>GFX_GOL_ObjectIsRedrawSet</u>	This function checks if the object needs to be redrawn or not.
	<u>GFX_GOL_ObjectListDraw</u>	This function redraws all objects in the current active list that has the rendering state bits set.
	<u>GFX_GOL_ObjectListFree</u>	This function sets the active list to the new list.
	<u>GFX_GOL_ObjectListGet</u>	This function returns the current active list.
	<u>GFX_GOL_ObjectListNew</u>	This function removes an object with the given user defined ID from the currently active list.
	<u>GFX_GOL_ObjectListSet</u>	This function sets the active list to the new list.
	<u>GFX_GOL_ObjectMessage</u>	This function process the received message from the user to determine the affected objects. Depending on the message and the affected objects, object states are modified based on the default behaviour or user defined behaviour.
	<u>GFX_GOL_ObjectNextGet</u>	This function returns the pointer to next object in the list after the specified object.
	<u>GFX_GOL_ObjectRectangleRedraw</u>	This function marks all objects in the active list intersected by the given rectangular area to be redrawn.

	GFX_GOL_ObjectTypeGet	This function returns the object type.
	GFX_GOL_PanelAlphaParameterSet	This is function GFX_GOL_PanelAlphaParameterSet.
	GFX_GOL_PanelBackgroundSet	This is function GFX_GOL_PanelBackgroundSet.
	GFX_GOL_PanelDraw	This is function GFX_GOL_PanelDraw.
	GFX_GOL_PanelGradientParameterSet	This is function GFX_GOL_PanelGradientParameterSet.
	GFX_GOL_PanelParameterSet	This is function GFX_GOL_PanelParameterSet.
	GFX_GOL_TwoTonePanelDraw	This is function GFX_GOL_TwoTonePanelDraw.

Macros

	Name	Description
	GFX_GOL_ObjectStateClear	This function clears the state bits of the given object.
	GFX_GOL_ObjectStateGet	This function retrieves the current value of the state bits of an object.
	GFX_GOL_ObjectStateSet	This function sets the state bits of the given object.

Structures

	Name	Description
	GFX_GOL_MESSAGE	Specifies message structure used in the library.
	GFX_GOL_OBJ_HEADER	Specifies Graphics Object Layer structure used in objects.
	GOL_PANEL_PARAM	Specifies panel parameters.

Types

	Name	Description
	GFX_GOL_DRAW_CALLBACK_FUNC	Draw callback function definition. This application defined function allows the application to perform application specific rendering.
	GFX_GOL_TRANSLATED_ACTION, \ GFX_GOL_OBJ_HEADER *, \ GFX_GOL_MESSAGE * \)	Message callback function definition. This application defined function allows the application to perform application specific processing of user messages.

File Name

gfx_gol.h

Company

Microchip Technology Inc.

5.2.2.8.7 gfx_gol_analog_clock.h

Graphics Library Implementation

Refer to Microchip Graphics Library for complete documentation of the AnalogClock Object.

Functions

	Name	Description
	GFX_AnalogClockCreate	Creates a analog clock object.
	GFX_GOL_AnalogClockDraw	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.
	GFX_GOL_AnalogClockHandsDraw	Draws the current position of the clock hands with the given thickness and color.
	GFX_GOL_AnalogClockHourSet	Sets the hour value of the analog clock.

	GFX_GOL_AnalogClockMinuteSet	Sets the minute value of the analog clock.
	GFX_GOL_AnalogClockSecondSet	Sets the second value of the analog clock.

Macros

	Name	Description
	GFX_GOL_ANALOGCLOCK_DISABLED	Bit for disabled state.
	GFX_GOL_ANALOGCLOCK_DRAW	Bit to indicate button must be redrawn.
	GFX_GOL_ANALOGCLOCK_HIDE	Bit to indicate button must be removed from screen.
	GFX_GOL_ANALOGCLOCK_PRESSED	Bit for press state.
	GFX_GOL_ANALOGCLOCK_TICK	Bit to tick second hand
	GFX_GOL_ANALOGCLOCK_UPDATE_HOUR	Bit to indicate hour hand must be redrawn
	GFX_GOL_ANALOGCLOCK_UPDATE_MINUTE	Bit to indicate minute hand must be redrawn
	GFX_GOL_ANALOGCLOCK_UPDATE_SECOND	Bit to indicate minute hand must be redrawn

Structures

	Name	Description

5.2.2.8.8 gfx_gol_button.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the [GFX_GOL_BUTTON](#) Object.

Enumerations

	Name	Description
	GFX_GOL_BUTTON_STATE	Object States Definition:

Functions

	Name	Description
	GFX_GOL_ButtonActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ButtonActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_ButtonCreate	This function creates a GFX_GOL_BUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ButtonDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ButtonTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_ButtonTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_ButtonTextSet	This function sets the address of the current text string used by the object.

Macros

	Name	Description
	GFX_GOL_ButtonPressStateImageGet	This function gets the image used when in the pressed state.
	GFX_GOL_ButtonPressStateImageSet	This function sets the image to be used when in the pressed state.
	GFX_GOL_ButtonReleaseStateImageGet	This function gets the image used when in the released state.
	GFX_GOL_ButtonReleaseStateImageSet	This function sets the image to be used when in the released state.
	GFX_GOL_ButtonTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description

5.2.2.8.9 gfx_gol_chart.h

- Module for Microchip Graphics Library
- GOL Layer
- Chart

- FileName: gfx_chart.h
- Dependencies: None
- Processor: PIC24F, PIC24H, dsPIC, PIC32
- Compiler: MPLAB C30, MPLAB C32
- Linker: MPLAB LINK30, MPLAB LINK32
- Company: Microchip Technology Incorporated

*

- Software License Agreement

*

- Copyright © 2008 Microchip Technology Inc. All rights reserved.
- Microchip licenses to you the right to use, modify, copy and distribute
- Software only when embedded on a Microchip microcontroller or digital
- signal controller, which is integrated into your product or third party
- product (pursuant to the sublicense terms in the accompanying license
- agreement).

*

- You should refer to the license agreement accompanying this Software
- for additional information regarding your rights and obligations.

*

- SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
- KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
- OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
- PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
- OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
- BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
- DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
- INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
- COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
- CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
- OR OTHER SIMILAR COSTS.

*

- Date Comment

* ~~~~~

- 4/8/08 ...

Functions

	Name	Description
	GFX_GOL_ChartActionGet	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.
	GFX_GOL_ChartCreate	This function creates a CHART object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ChartDataSeriesAdd	This function creates a DATASERIES object and populates the structure with the given parameters.
	GFX_GOL_ChartDataSeriesFree	This function removes DATASERIES object from the list of DATASERIES objects and frees the memory used of that removed object.
	GFX_GOL_ChartDataSeriesRemove	This function removes DATASERIES object from the list of DATASERIES objects and frees the memory used of that removed object. The position of the object to be removed is specified by the number parameter. If the list has only one member, it removes the member regardless of the number given.
	GFX_GOL_ChartDataSeriesSet	This is function GFX_GOL_ChartDataSeriesSet .
	GFX_GOL_ChartDraw	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. The colors of the bars of the bar chart or sectors of the pie chart can be the default color table or user defined color table set by ChSetColorTable() function. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started.... more
	GFX_GOL_ChartPercentRangeSet	This function sets the minimum and maximum range of percentage that the bar chart will show. The criteria is that min <= max. This affects bar charts only and CH_PERCENTAGE bit state is set.
	GFX_GOL_ChartSampleRangeSet	This function sets the sample start and sample end when drawing the chart. Together with the data series' SHOW_DATA flags the different way of displaying the chart data is achieved.
	GFX_GOL_ChartValueRangeSet	This function sets the minimum and maximum range of values that the bar chart will show. The criteria is that min <= max.

Macros

	Name	Description
	CH_CLR0	Bright Blue
	CH_CLR1	Bright Red
	CH_CLR10	Light Blue
	CH_CLR11	Light Red
	CH_CLR12	Light Green
	CH_CLR13	Light Yellow
	CH_CLR14	Light Orange
	CH_CLR15	Gold
	CH_CLR2	Bright Green

	CH_CLR3	Bright Yellow
	CH_CLR4	Orange
	CH_CLR5	Blue
	CH_CLR6	Red
	CH_CLR7	Green
	CH_CLR8	Yellow
	CH_CLR9	Dark Orange
	CHART_MARGIN	Margin from the edges.
	CHART_YGRIDCOUNT	defines the number of grids to be drawn on the y-axis (includes the origin at y).
	GFX_GOL_CHART_3D_ENABLE	Bit to indicate that bar charts are to be drawn with 3-D effect
	GFX_GOL_CHART_BAR	Bit to indicate the chart is type bar. If both PIE and BAR types are set BAR type has higher priority.
	GFX_GOL_CHART_BAR_HOR	These bits (with CH_BAR bit set), sets the bar chart to be drawn horizontally.
	GFX_GOL_CHART_DISABLED	Bit for disabled state.
	GFX_GOL_CHART_DONUT	These bits (with CH_PIE bit set), sets the pie chart to be drawn in a donut shape.
	GFX_GOL_CHART_DRAW	Bit to indicate chart must be redrawn.
	GFX_GOL_CHART_DRAW_DATA	Bit to indicate data portion of the chart must be redrawn.
	GFX_GOL_CHART_HIDE	Bit to indicate chart must be removed from screen.
	GFX_GOL_CHART_LEGEND	Bit to indicate that legend is to be shown. Usable only when seriesCount > 1.
	GFX_GOL_CHART_NUMERIC	This bit is used only for bar charts. If this bit is set, it indicates that the bar chart labels for variables are numeric. If this bit is not set, it indicates that the bar chart labels for variables are alphabets.
	GFX_GOL_CHART_PERCENT	Bit to indicate that the pie chart will be drawn with percentage values shown for the sample data. For bar chart, if CH_VALUE is set, it toggles the value shown to percentage.
	GFX_GOL_CHART_PIE	Bit to indicate the chart is type pie. If both PIE and BAR types are set BAR type has higher priority.
	GFX_GOL_CHART_VALUE	Bit to indicate that the values of the bar chart data or pie chart data are to be shown
	GFX_GOL_ChartAxisLabelFontGet	Macros: GFX_GOL_ChartAxisLabelFontGet(pCh) This macro returns the location of the font used for the X and Y axis labels of the chart.
	GFX_GOL_ChartAxisLabelFontSet	Macros: GFX_GOL_ChartAxisLabelFontSet(pCh, pNewFont) This macro sets the location of the font used for the X and Y axis labels of the chart.
	GFX_GOL_ChartColorTableGet	Macros: GFX_GOL_ChartColorTableGet(pCh) This macro returns the current color table used for the pie and bar charts.
	GFX_GOL_ChartColorTableSet	Macros: GFX_GOL_ChartColorTableSet(pCh, pNewTable) This macro sets the color table used to draw the data in pie and bar charts.
	GFX_GOL_ChartGridLabelFontGet	Macros: GFX_GOL_ChartGridLabelFontGet(pCh) This macro returns the location of the font used for the X and Y axis grid labels of the chart.
	GFX_GOL_ChartGridLabelFontSet	Macros: GFX_GOL_ChartGridLabelFontSet(pCh, pNewFont) This macro sets the location of the font used for the X and Y axis grid labels of the chart.

	GFX_GOL_ChartPercentMaxGet	Macros: ChGetPercentMax(pCh) This macro returns the current maximum value of the percentage range that will be drawn for bar charts when CH_PERCENTAGE bit state is set.
	GFX_GOL_ChartPercentMinGet	Macros: ChGetPercentMin(pCh) This macro returns the current minimum value of the percentage range that will be drawn for bar charts when CH_PERCENTAGE bit state is set.
	GFX_GOL_ChartPercentRangeGet	This macro gets the percentage range for bar charts. The value returned is calculated from percentage max - min. To get the minimum use ChGetPercentMin() and to get the maximum use ChGetPercentMax().
	GFX_GOL_ChartSampleEndGet	Macros: GFX_GOL_ChartSampleEndGet(pCh) This macro returns the sampling end value.
	GFX_GOL_ChartSampleLabelGet	Macros: GFX_GOL_ChartSampleLabelGet(pCh) This macro returns the address of the current text string used for the sample axis label of the bar chart.
	GFX_GOL_ChartSampleLabelSet	Macros: GFX_GOL_ChartSampleLabelSet(pCh, pNewXLabel) This macro sets the address of the current text string used for the sample axis label of the bar chart.
	GFX_GOL_ChartSampleRangeGet	This macro gets the sample range for pie or bar charts. The value returned is calculated from smplEnd - smplStart.
	GFX_GOL_ChartSampleStartGet	Macros: GFX_GOL_ChartSampleStartGet(pCh) This macro returns the sampling start value.
	GFX_GOL_ChartSeriesHide	Macros: GFX_GOL_ChartSeriesHide(CHART *pCh, uint16_t seriesNum) This macro sets the specified data series number show flag to be set to HIDE_DATA .
	GFX_GOL_ChartShowSeriesCountGet	Macros: GFX_GOL_ChartShowSeriesCountGet(pCh) This macro shows the number of data series that has its show flag set to SHOW_DATA .
	GFX_GOL_ChartShowSeriesSet	Macros: short GFX_GOL_ChartShowSeriesSet(CHART *pCh, uint16_t seriesNum) This macro sets the specified data series number show flag to be set to SHOW_DATA .
	GFX_GOL_ChartShowSeriesStatusGet	Macros: GFX_GOL_ChartShowSeriesStatusGet(pDSeries) This macro returns the show ID status of the DATASERIES .
	GFX_GOL_ChartTitleFontGet	Macros: GFX_GOL_ChartTitleFontGet(pCh) This macro returns the location of the font used for the title of the chart.
	GFX_GOL_ChartTitleFontSet	Macros: GFX_GOL_ChartTitleFontSet(pCh, pNewFont) This macro sets the location of the font used for the title of the chart.
	GFX_GOL_ChartTitleGet	Macros: GFX_GOL_ChartTitleGet(pCh) This macro returns the address of the current text string used for the title of the chart.
	GFX_GOL_ChartTitleSet	Macros: GFX_GOL_ChartTitleSet(pCh, pNewTitle) This macro sets the address of the current text string used for the title of the chart.
	GFX_GOL_ChartValueLabelGet	Macros: GFX_GOL_ChartValueLabelGet(pCh) This macro returns the address of the current text string used for the value axis label of the bar chart.
	GFX_GOL_ChartValueLabelSet	Macros: GFX_GOL_ChartValueLabelSet(pCh, pNewYLabel) This macro sets the address of the current text string used for the value axis label of the bar chart.

	GFX_GOL_ChartValueMaxGet	Macros: GFX_GOL_ChartValueMaxGet(pCh) This macro returns the current maximum value that will be drawn for bar charts.
	GFX_GOL_ChartValueMinGet	Macros: GFX_GOL_ChartValueMinGet(pCh) This macro returns the current minimum value that will be drawn for bar charts.
	GFX_GOL_ChartValueRangeGet	This macro gets the current range for bar charts. The value returned is calculated from the current (valMax - valMin) set. To get the minimum use ChGetValueMin() and to get the maximum use ChGetValueMax().
	HIDE_DATA	Macro used to reset the data series show flag or indicate that the data series will be not be shown when the chart is drawn.
	SHOW_DATA	Macro used to set the data series show flag or indicate that the data series will be shown when the chart is drawn.
	USE_HORZ_ASCENDING_ORDER	use ascending order when displaying variables on horizontal mode (bar chart only)
	USE_PIE_ENABLE_LABEL	use labels: A:10%,30 where each sample is labeled A-Z followed by a colon.

Structures

	Name	Description
	CHARTPARAM	Defines the parameters for the CHART object.
	DATASERIES	Defines a variable for the CHART object. It specifies the number of samples, pointer to the array of samples for the data series and pointer to the next data series. A member of this structure (show) is used as a flag to determine if the series is to be drawn or not. Together with the smplStart and smplEnd it will determine what kind of chart will be drawn.
	GFX_GOL_CHART	Defines the parameters required for a chart Object.

5.2.2.8.10 gfx_gol_check_box.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the CheckBox Object.

Enumerations

	Name	Description

5.2.2.8.11 gfx_gol_custom_control.h

Graphics Library Implementation

Refer to Microchip Graphics Library for complete documentation of the CustomControl Object.

Functions

	Name	Description
	GFX_GOL_CustomControlActionGet	translates the GOL message for the custom control
	GFX_GOL_CustomControlActionSet	changes the state of the object
	GFX_GOL_CustomControlCreate	creates the custom control
	GFX_GOL_CustomControlDraw	draws custom control

Macros

	Name	Description
	GFX_GOL_CUSTOMCONTROL_DISABLED	disabled
	GFX_GOL_CUSTOMCONTROL_DRAW	whole control must be redrawn
	GFX_GOL_CUSTOMCONTROL_DRAW_BAR	control bar should be redrawn
	GFX_GOL_CUSTOMCONTROL_HIDE	must be removed from screen
	GFX_GOL_CustomControlGetPos	Macros: GFX_GOL_CustomControlGetPos(pCc) returns position
	GFX_GOL_CustomControlSetPos	Macros: GFX_GOL_CustomControlSetPos(pCc,position) sets the current position of the control

Structures

	Name	Description
	GFX_GOL_CUSTOMCONTROL	The structure contains data for the control

Company

Microchip Technology Inc.

FileName: gfx_gol_custom_control.c

5.2.2.8.12 gfx_gol_digital_meter.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Digital Meter Object.

Enumerations

	Name	Description
	GFX_GOL_DIGITALMETER_STATE	Object States Definition:

Functions

	Name	Description
	GFX_GOL_DigitalMeterActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_DigitalMeterCreate	This function creates a GFX_GOL_DIGITALMETER object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_DigitalMeterDecrement	This function decrements the meter value by the delta value set.
	GFX_GOL_DigitalMeterDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_DigitalMeterIncrement	This function increments the meter value by the delta value set.
	GFX_GOL_DigitalMeterValueSet	This function sets the value of the meter.

Macros

	Name	Description
	GFX_GOL_DIGITALMETER_INDENT	This is a DigitalMeter compile time option to define the indent constant for the text used in the frame.
	GFX_GOL_DIGITALMETER_WIDTH	This is a DigitalMeter compile time option to define the user should change this value depending on the number of digits he wants to display.

	GFX_GOL_DigitalMeterTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_DigitalMeterTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_DigitalMeterValueGet	This function returns the current value of the digital meter.

Structures

	Name	Description
	GFX_GOL_DIGITALMETER	Properties for a digital meter.

File Name

gfx_gol_digital_meter.h

Company

Microchip Technology Inc.

5.2.2.8.13 gfx_gol_edit_box.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Edit Box Object.

Enumerations

	Name	Description
	GFX_GOL_EDITBOX_STATE	Object States Definition:

Functions

	Name	Description
	GFX_GOL_EditBoxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_EditBoxActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_EditBoxCharAdd	This function adds a character at the end of the text used by the object.
	GFX_GOL_EditBoxCharRemove	This function removes a character at the end of the text used by the object.
	GFX_GOL_EditBoxCreate	This function creates a GFX_GOL_EDITBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_EditBoxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_EditBoxTextSet	This function sets the text in the object.

Macros

	Name	Description
	GFX_GOL_EditBoxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_EditBoxTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_EditBoxTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_EDITBOX	Defines the parameters required for a Edit Box Object.

File Name

gfx_gol_edit_box.h

Company

Microchip Technology Inc.

5.2.2.8.14 gfx_gol_font_default.h

AUTO-GENERATED CODE: Graphics Resource Converter version: 4.00.39 BETA

This file is generated by the Graphics Resource Converter containing resources such as images and fonts that can be used by Microchip's Graphics Library, which is part of the MLA.

Macros

	Name	Description
	<u>GFX_GOL_FONT_DEFAULT_H_FILE</u>	

File Name

gfx_gol_font_default.h

Company

Microchip Technology, Inc.

5.2.2.8.15 gfx_gol_grid.h

- Module for Microchip Graphics Library
- Grid control

- FileName: gfx_grid.h
- Dependencies: none
- Processor: PIC24F, PIC24H, dsPIC, PIC32
- Compiler: MPLAB C30, MPLAB C32
- Linker: MPLAB LINK30, MPLAB LINK32
- Company: Microchip Technology Incorporated

*

- Software License Agreement

*

- Copyright © 2008 Microchip Technology Inc. All rights reserved.
- Microchip licenses to you the right to use, modify, copy and distribute
- Software only when embedded on a Microchip microcontroller or digital
- signal controller, which is integrated into your product or third party
- product (pursuant to the sublicense terms in the accompanying license
- agreement).

*

- You should refer to the license agreement accompanying this Software
- for additional information regarding your rights and obligations.

*

- SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
- KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
- OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
- PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
- OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
- BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
- DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
- INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
- COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
- CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
- OR OTHER SIMILAR COSTS.

*

- Date Comment

*~~~~~

- 02/29/08 ...
- 04/16/10 Added Grid Item as text.

Functions

	Name	Description
	<u>GridClearCellState</u>	This function clears the state of the cell (or Grid Item) specified by the column and row.
	<u>GridCreate</u>	This function creates a <u>GRID</u> object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	<u>GridDraw</u>	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
	<u>GridFreeItems</u>	This function removes all grid items for the given Grid and frees the memory used.
	<u>GridGetCell</u>	This function removes all grid items for the given Grid and frees the memory used.
	<u>GridMsgDefault</u>	This function performs the actual state change based on the translated message given. The following state changes are supported:
	<u>GridSetCell</u>	This function sets the Grid Item state and data.
	<u>GridSetCellState</u>	This function sets the state of the Grid Item or cell.
	<u>GridSetFocus</u>	This function sets the focus of the specified Grid Item or cell.
	<u>GridTranslateMsg</u>	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

	Name	Description
	<u>GRID_DISABLED</u>	Bit for disabled state
	<u>GRID_DRAW_ALL</u>	Bit to indicate whole edit box must be redrawn
	<u>GRID_DRAW_ITEMS</u>	Bit to indicate that at least one item must be redrawn, but not the entire grid.
	<u>GRID_FOCUSED</u>	Bit for focused state
	<u>GRID_HEIGHT</u>	This is macro GRID_HEIGHT.
	<u>GRID_HIDE</u>	Bit to remove object from screen
	<u>GRID_MAX_COLUMNS</u>	This is macro GRID_MAX_COLUMNS.
	<u>GRID_MAX_ROWS</u>	This is macro GRID_MAX_ROWS.
	<u>GRID_OUT_OF_BOUNDS</u>	Status of an out of bounds cell <u>GridSetCell()</u> operation.
	<u>GRID_SHOW_BORDER_ONLY</u>	Draw only the outside border of the grid.
	<u>GRID_SHOW_FOCUS</u>	Highlight the focused cell.
	<u>GRID_SHOW_LINES</u>	Display grid lines
	<u>GRID_SHOW_SEPARATORS_ONLY</u>	Draw only the lines between cells (like Tic-tac-toe)
	<u>GRID_SUCCESS</u>	Status of a successful <u>GridSetCell()</u> operation.
	<u>GRID_TYPE_MASK</u>	This is macro GRID_TYPE_MASK.
	<u>GRID_WIDTH</u>	This is macro GRID_WIDTH.
	<u>GridGetFocusX</u>	Macros: GridGetFocusX(pGrid) This macro returns the x position of the focused cell.
	<u>GridGetFocusY</u>	Macros: GridGetFocusY(pGrid) This macro returns the y position of the focused cell.
	<u>GRIDITEM_DRAW</u>	Draw this cell
	<u>GRIDITEM_IS_BITMAP</u>	The grid item is a bitmap.
	<u>GRIDITEM_IS_TEXT</u>	The grid item is a text string.
	<u>GRIDITEM_SELECTED</u>	The cell is selected.
	<u>GRIDITEM_TEXTBOTTOM</u>	Bit to indicate text is top aligned.
	<u>GRIDITEM_TEXTLEFT</u>	Text in the cell is left aligned.
	<u>GRIDITEM_TEXTRIGHT</u>	Text in the cell is right aligned.
	<u>GRIDITEM_TEXTTOP</u>	Bit to indicate text is bottom aligned.

Structures

	Name	Description
	<u>GRID</u>	Defines the parameters required for a grid Object. Clipping is not supported in grid object.
	<u>GRIDITEM</u>	Defines the grid item.

5.2.2.8.16 gfx_gol_group_box.h

Graphics Library Implementation

Refer to Microchip Graphics Library for complete documentation of the Group Box Object.

Enumerations

	Name	Description
	<u>GFX_GOL_GROUPBOX_STATE</u>	Object States Definition:

Functions

	Name	Description
	GFX_GOL_GroupboxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_GroupboxCreate	This function creates a GFX_GOL_GROUPBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_GroupboxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_GroupboxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_GroupboxTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_GroupboxTextSet	This function sets the address of the current text string used by the object.

Macros

	Name	Description
	GFX_GOL_GroupboxTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_GROUPBOX	Defines the parameters required for a group box Object. The textwidth and textHeight is not checked with the actual dimension of the object. Clipping is not supported in group box object. It is possible for the text to exceed the dimension of the Object.

Company

Microchip Technology Inc.

FileName: gfx_gol_group_box.h

5.2.2.8.17 gfx_gol_list_box.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the ListBox Object.

Enumerations

	Name	Description
	GFX_GOL_LISTBOX_ITEM_STATUS	Bit definitions for the status of an item
	GFX_GOL_LISTBOX_STATE	This is type GFX_GOL_LISTBOX_STATE.

Functions

	Name	Description
	GFX_GOL_ListBoxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ListBoxActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_ListBoxCreate	This function creates a GFX_GOL_LISTBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

	GFX_GOL_ListBoxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ListBoxItemAdd	This function adds an item to the list box.
	GFX_GOL_ListBoxItemFocusGet	This function returns the index of the focused item in the list box.
	GFX_GOL_ListBoxItemFocusSet	This function sets the focus of the item with the index number specified by index.
	GFX_GOL_ListBoxItemListRemove	This function removes the entire items list of the list box and free the memory used.
	GFX_GOL_ListBoxItemRemove	This function removes an item from the list box and free the memory used.
	GFX_GOL_ListBoxSelectionChange	This function changes the selection status of the given item in the list box.
	GFX_GOL_ListBoxSelectionGet	This function searches for selected items from the list box.
	GFX_GOL_ListBoxVisibleItemCountGet	This function returns the count of items visible in the list box.

Macros

	Name	Description
	GFX_GOL_ListBoxItemCountGet	This function returns the count of items in the list box.
	GFX_GOL_ListBoxItemImageGet	This function gets the image set for a list box item.
	GFX_GOL_ListBoxItemImageSet	This function sets the image for a list box item.
	GFX_GOL_ListBoxItemListGet	This function returns the pointer to the item list of the list box.
	GFX_GOL_ListBoxItemSelectStatusClear	This function clears the selection status of the item.
	GFX_GOL_ListBoxItemSelectStatusSet	This function sets the selection status of the item to selected.
	GFX_GOL_ListBoxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_ListBoxTextAlignmentSet	This function sets the text alignment of the text string used by the object.

Structures

	Name	Description
	GFX_GOL_LISTBOX	Defines the parameters required for a list box Object.
	GFX_GOL_LISTITEM	Defines the parameters required for a list item used in list box.

File Name

gfx_gol_list_box.h

Company

Microchip Technology Inc.

5.2.2.8.18 gfx_gol_meter.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Meter Object.

Enumerations

	Name	Description

5.2.2.8.19 gfx_gol_picture.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Picture Object.

Enumerations

	Name	Description
	GFX_GOL_PICTURECONTROLCONTROL_STATE	Object States Definition:

Functions

	Name	Description
	GFX_GOL_PictureControlActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_PictureControlCreate	This function creates a GFX_GOL_PICTURECONTROL object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_PictureControlDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_PictureControlPartialSet	This function sets the partial image parameters to be in the object.

Macros

	Name	Description
	GFX_GOL_PictureControlImageGet	This function gets the image used when in the pressed state.
	GFX_GOL_PictureControlImageSet	This function sets the image to be in the object.
	GFX_GOL_PictureControlScaleSet	Sets the scale factor used to render the bitmap used in the object.

Structures

	Name	Description
	GFX_GOL_PICTURECONTROL	The structure contains data for picture control

File Name

gfx_gol_picture.h

Company

Microchip Technology Inc.

5.2.2.8.20 gfx_gol_progress_bar.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the ProgressBar Object.

Enumerations

	Name	Description
	GFX_GOL_PROGRESSBAR_STATE	Object States Definition:

Functions

	Name	Description
	GFX_GOL_ProgressBarActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ProgressBarCreate	This function creates a GFX_GOL_PROGRESSBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ProgressBarDraw	This function renders the object on the screen based on the current state of the object.

	GFX_GOL_ProgressBarPositionSet	This function sets the position of the progress bar.
	GFX_GOL_ProgressBarRangeSet	This function sets the range of the progress bar.

Macros

	Name	Description
	GFX_GOL_ProgressBarPositionGet	This function returns the current position of the progress bar.
	GFX_GOL_ProgressBarRangeGet	This function returns the range of the progress bar.

Structures

	Name	Description
	GFX_GOL_PROGRESSBAR	The structure contains data for the progress bar

File Name

gfx_gol_progress_bar.h

Company

Microchip Technology Inc.

5.2.2.8.21 gfx_gol_radio_button.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the RadioButton Object.

Enumerations

	Name	Description
	GFX_GOL_RADIOBUTTON_STATE	Object States Definition:

Functions

	Name	Description
	GFX_GOL_RadioButtonActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_RadioButtonActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_RadioButtonCheckGet	This function returns the ID of the currently checked radio button in the group.
	GFX_GOL_RadioButtonCheckSet	This function sets the ID of the currently checked radio button in the group.
	GFX_GOL_RadioButtonCreate	This function creates a GFX_GOL_RADIOBUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_RadioButtonDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_RadioButtonTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_RadioButtonTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_RadioButtonTextSet	This function sets the address of the current text string used by the object.

Macros

	Name	Description
	GFX_GOL_RadioButtonTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_RADIOBUTTON	the structure contains data for the Radio Button

File Name

gfx_gol_radio_button.h

Company

Microchip Technology Inc.

5.2.2.8.22 gfx_gol_round_dial.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Round dial Object.

Functions

	Name	Description
	GFX_GOL_RoundDialActionGet	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen inputs.
	GFX_GOL_RoundDialActionSet	This function performs the actual state change based on the translated message given. The following state changes are supported:
	GFX_GOL_RoundDialCreate	This function creates a ROUNDDIAL object with the parameters given.
	GFX_GOL_RoundDialDraw	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the center (x,y) position and the radius parameters. The colors used are dependent on the state of the object. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
	RdiaCosine	This is function RdiaCosine.
	RdiaSine	This is function RdiaSine.

Macros

	Name	Description
	GFX_GOL_ROUNDIAL_DISABLED	Bit for disabled state.
	GFX_GOL_ROUNDIAL_DRAW	Bit to indicate object must be redrawn.
	GFX_GOL_ROUNDIAL_HIDE	Bit to indicate object must be removed from screen.
	GFX_GOL_ROUNDIAL_MAX_POSITIONS	This is macro GFX_GOL_ROUNDIAL_MAX_POSITIONS.
	GFX_GOL_ROUNDIAL_QUADRANT_POSITIONS	This is macro GFX_GOL_ROUNDIAL_QUADRANT_POSITIONS.
	GFX_GOL_ROUNDIAL_ROT_CCW	Bit for rotate counter clockwise state.
	GFX_GOL_ROUNDIAL_ROT_CW	Bit for rotate clockwise state.
	GFX_GOL_RoundDialValueDecrement	Used to directly decrement the value. The delta change used is the resolution setting (res).

	<u>GFX_GOL_RoundDialValueGet</u>	Returns the current dial value. Value is always in the 0-max range inclusive.
	<u>GFX_GOL_RoundDialValueIncrement</u>	Used to directly increment the value. The delta change used is the resolution setting (res).
	<u>GFX_GOL_RoundDialValueSet</u>	Sets the value to the given new value. Value set must be in 0-max range inclusive.

Structures

	Name	Description
	<u>GFX_GOL_ROUNDIAL</u>	Defines the parameters required for a dial Object.

Company

Microchip Technology Inc.

FileName: gfx_gol_round_dial.h

5.2.2.8.23 gfx_gol_scan_codes.h

Module for Microchip Graphics Library - Graphic Object Layer

This header file contains scan codes for the standard AT keyboard that can be used to process user action in GOL messages.

Macros

	Name	Description
	<u>SCAN_BS_PRESSED</u>	Back space key pressed.
	<u>SCAN_BS_RELEASED</u>	Back space key released.
	<u>SCAN_CR_PRESSED</u>	Carriage return pressed.
	<u>SCAN_CR_RELEASED</u>	Carriage return released.
	<u>SCAN_CRA_PRESSED</u>	Carriage return alternate pressed.
	<u>SCAN_CRA_RELEASED</u>	Carriage return alternate released.
	<u>SCAN_DEL_PRESSED</u>	Delete key pressed.
	<u>SCAN_DEL_RELEASED</u>	Delete key released.
	<u>SCAN_DOWN_PRESSED</u>	Down key pressed.
	<u>SCAN_DOWN_RELEASED</u>	Down key released.
	<u>SCAN_END_PRESSED</u>	End key pressed.
	<u>SCAN_END_RELEASED</u>	End key released.
	<u>SCAN_HOME_PRESSED</u>	Home key pressed.
	<u>SCAN_HOME_RELEASED</u>	Home key released.
	<u>SCAN_LEFT_PRESSED</u>	Left key pressed.
	<u>SCAN_LEFT_RELEASED</u>	Left key released.
	<u>SCAN_PGDOWN_PRESSED</u>	Page down key pressed.
	<u>SCAN_PGDOWN_RELEASED</u>	Page down key released.
	<u>SCAN_PGUP_PRESSED</u>	Page up key pressed.
	<u>SCAN_PGUP_RELEASED</u>	Page up key released.
	<u>SCAN_RIGHT_PRESSED</u>	Right key pressed.
	<u>SCAN_RIGHT_RELEASED</u>	Right key released.
	<u>SCAN_SPACE_PRESSED</u>	Space key pressed.
	<u>SCAN_SPACE_RELEASED</u>	Space key released.
	<u>SCAN_TAB_PRESSED</u>	Tab key pressed.

	SCAN_TAB_RELEASED	Tab key released.
	SCAN_UP_PRESSED	Up key pressed.
	SCAN_UP_RELEASED	Up key released.

File Name

gfx_gol_scan_codes.h

Company

Microchip Technology Inc.

5.2.2.8.24 gfx_gol_scheme.h

Module for Microchip Graphics Library - Graphic Object Layer Scheme

This module implements the common routines for the Graphics Object Layer of the Microchip Graphics Library. The routines are independent of the Display Driver Layer and should be compatible with any Display Driver that is compliant with the requirements of the Display Driver Layer of the Graphics Library. The module utilizes the Graphics Primitive Layer to render the objects.

Functions

	Name	Description
	GFX_GOL_SchemeAlphaPrecentSet	This is function GFX_GOL_SchemeAlphaPrecentSet.
	GFX_GOL_SchemeBackgroundColorGet	This is function GFX_GOL_SchemeBackgroundColorGet.
	GFX_GOL_SchemeBackgroundColorSet	This is function GFX_GOL_SchemeBackgroundColorSet.
	GFX_GOL_SchemeBackgroundImageSet	This is function GFX_GOL_SchemeBackgroundImageSet.
	GFX_GOL_SchemeBackgroundTypeGet	This is function GFX_GOL_SchemeBackgroundTypeGet.
	GFX_GOL_SchemeBackgroundTypeSet	This is function GFX_GOL_SchemeBackgroundTypeSet.
	GFX_GOL_SchemeColor0Get	This is function GFX_GOL_SchemeColor0Get.
	GFX_GOL_SchemeColor0Set	This is function GFX_GOL_SchemeColor0Set.
	GFX_GOL_SchemeColor1Get	This is function GFX_GOL_SchemeColor1Get.
	GFX_GOL_SchemeColor1Set	This is function GFX_GOL_SchemeColor1Set.
	GFX_GOL_SchemeColorDisabledGet	This is function GFX_GOL_SchemeColorDisabledGet.
	GFX_GOL_SchemeColorDisabledSet	This is function GFX_GOL_SchemeColorDisabledSet.
	GFX_GOL_SchemeColorSet	This is function GFX_GOL_SchemeColorSet.
	GFX_GOL_SchemeEmbossDarkColorGet	This is function GFX_GOL_SchemeEmbossDarkColorGet.
	GFX_GOL_SchemeEmbossDarkColorSet	This is function GFX_GOL_SchemeEmbossDarkColorSet.
	GFX_GOL_SchemeEmbossLightColorGet	This is function GFX_GOL_SchemeEmbossLightColorGet.
	GFX_GOL_SchemeEmbossLightColorSet	This is function GFX_GOL_SchemeEmbossLightColorSet.
	GFX_GOL_SchemeEmbossSet	This is function GFX_GOL_SchemeEmbossSet.
	GFX_GOL_SchemeEmbossSizeGet	This is function GFX_GOL_SchemeEmbossSizeGet.
	GFX_GOL_SchemeEmbossSizeSet	This is function GFX_GOL_SchemeEmbossSizeSet.
	GFX_GOL_SchemeFillStyleGet	This is function GFX_GOL_SchemeFillStyleGet.
	GFX_GOL_SchemeFillStyleSet	This is function GFX_GOL_SchemeFillStyleSet.
	GFX_GOL_SchemeFontGet	This is function GFX_GOL_SchemeFontGet.
	GFX_GOL_SchemeFontSet	This is function GFX_GOL_SchemeFontSet.
	GFX_GOL_SchemeGradientColorSet	This is function GFX_GOL_SchemeGradientColorSet.
	GFX_GOL_SchemeGradientEndColorGet	This is function GFX_GOL_SchemeGradientEndColorGet.
	GFX_GOL_SchemeGradientEndColorSet	This is function GFX_GOL_SchemeGradientEndColorSet.

	GFX_GOL_SchemeGradientStartColorGet	This is function GFX_GOL_SchemeGradientStartColorGet.
	GFX_GOL_SchemeGradientStartColorSet	This is function GFX_GOL_SchemeGradientStartColorSet.
	GFX_GOL_SchemeTextColor0Get	This is function GFX_GOL_SchemeTextColor0Get.
	GFX_GOL_SchemeTextColor0Set	This is function GFX_GOL_SchemeTextColor0Set.
	GFX_GOL_SchemeTextColor1Get	This is function GFX_GOL_SchemeTextColor1Get.
	GFX_GOL_SchemeTextColor1Set	This is function GFX_GOL_SchemeTextColor1Set.
	GFX_GOL_SchemeTextColorDisableGet	This is function GFX_GOL_SchemeTextColorDisableGet.
	GFX_GOL_SchemeTextColorDisableSet	This is function GFX_GOL_SchemeTextColorDisableSet.
	GFX_GOL_SchemeTextColorSet	This is function GFX_GOL_SchemeTextColorSet.

Structures

	Name	Description
	GFX_GOL_OBJ_SCHEME	This structure specifies the style scheme components of an object.

File Name

gfx_gol_scheme.h

Company

Microchip Technology Inc.

5.2.2.8.25 gfx_gol_scroll_bar.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Slider Object.

Enumerations

	Name	Description
	GFX_GOL_SCROLLBAR_STATE	Object States Definition:

Functions

	Name	Description
	GFX_GOL_ScrollBarActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ScrollBarActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_ScrollBarCreate	This function creates a GFX_GOL_SCROLLBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ScrollBarDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ScrollBarPageGet	This function returns the page size of the object.
	GFX_GOL_ScrollBarPageSet	This function sets the page size of the object.
	GFX_GOL_ScrollBarPositionDecrement	This function decrements the scroll bar position by the delta change (page) value set.
	GFX_GOL_ScrollBarPositionGet	This function returns the current position of the scroll bar thumb.
	GFX_GOL_ScrollBarPositionIncrement	This function increments the scroll bar position by the delta change (page) value set.
	GFX_GOL_ScrollBarPositionSet	This function sets the position of the scroll bar thumb.
	GFX_GOL_ScrollBarRangeGet	This function returns the range of the thumb of the scroll bar.

	GFX_GOL_ScrollBarRangeSet	This function sets the range of the thumb of the scroll bar.

Structures

	Name	Description
	GFX_GOL_SCROLLBAR	Defines the parameters required for a slider/scrollbar Object. Depending on the GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE state bit slider or scrollbar mode is set. If GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE is set, mode is slider; if not set mode is scroll bar. For scrollbar mode, focus rectangle is not drawn.

File Name

gfx_gol_scroll_bar.h

Company

Microchip Technology Inc.

5.2.2.8.26 gfx_gol_static_text.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Static Text Object.

Enumerations

	Name	Description
	GFX_GOL_STATICTEXT_STATE	Object States Definition:

Functions

	Name	Description
	GFX_GOL_StaticTextActionGet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_StaticTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_StaticTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_StaticTextCreate	This function creates a GFX_GOL_STATICTEXT object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_StaticTextDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_StaticTextGet	This function returns the address of the current text string used by the object.
	GFX_GOL_StaticTextSet	This function sets the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_STATICTEXT	Defines the parameters required for a Static Text Object.

File Name

gfx_gol_static_text.h

Company

Microchip Technology Inc.

5.2.2.8.27 gfx_gol_text_entry.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Text Entry Object.

Enumerations

	Name	Description
	GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE	Optional COMMANDS assigned to keys
	GFX_GOL_TEXTENTRY_STATE	Object States Definition:

Functions

	Name	Description
	GFX_GOL_TextEntryActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_TextEntryActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_TextEntryBufferClear	This function will clear the data in the display.
	GFX_GOL_TextEntryBufferGet	This function returns the buffer used to display text.
	GFX_GOL_TextEntryBufferSet	This function sets the buffer used to display text.
	GFX_GOL_TextEntryCharAdd	This function will insert a character to the end of the buffer.
	GFX_GOL_TextEntryCreate	This function creates a GFX_GOL_TEXTENTRY object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_TextEntryDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_TextEntryKeyCommandGet	This function will return the currently assigned command to the key with the given index.
	GFX_GOL_TextEntryKeyCommandSet	This function will assign a command to a key with the given index.
	GFX_GOL_TextEntryKeysPressed	This function will test if a key given by its index in the object is pressed.
	GFX_GOL_TextEntryKeyListCreate	This function will create the list of KEYMEMBERS that holds the information on each key.
	GFX_GOL_TextEntryKeyMemberListDelete	This function will delete the KEYMEMBER list assigned to the object.
	GFX_GOL_TextEntryKeyTextSet	This function will set the text assigned to a key with the given index.
	GFX_GOL_TextEntryLastCharDelete	This function will remove the last character of the buffer and replace it with a string terminator.
	GFX_GOL_TextEntrySpaceCharAdd	This function will insert a space character to the end of the buffer.

Macros

	Name	Description
	TE_ROUNDEDBUTTON_RADIUS	Optional settings Set this parameter in the GraphicsConfig.h

Structures

	Name	Description
	GFX_GOL_TEXTENTRY	Defines the parameters required for a TextEntry Object.
	KEYMEMBER	Defines the parameters and the strings assigned for each key.

File Name

gfx_gol_text_entry.h

Company

Microchip Technology Inc.

5.2.2.8.28 gfx_gol_window.h

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Window Object.

Enumerations

	Name	Description
	<u>GFX_GOL_WINDOW_STATE</u>	Object States Definition:

Functions

	Name	Description
	<u>GFX_GOL_WindowActionGet</u>	This function evaluates the message from a user if the message will affect the object or not.
	<u>GFX_GOL_WindowCreate</u>	This function creates a <u>GFX_GOL_WINDOW</u> object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	<u>GFX_GOL_WindowDraw</u>	This function renders the object on the screen based on the current state of the object.
	<u>GFX_GOL_WindowTextAlignmentGet</u>	This function returns the text alignment of the text string used by the object.
	<u>GFX_GOL_WindowTextAlignmentSet</u>	This function sets the text alignment of the text string used by the object.
	<u>GFX_GOL_WindowTextSet</u>	This function sets the address of the current text string used by the object.

Macros

	Name	Description
	<u>GFX_GOL_WindowImageGet</u>	This function gets the image used.
	<u>GFX_GOL_WindowTextGet</u>	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	<u>GFX_GOL_WINDOW</u>	The structure contains data for the window

Company

Microchip Technology Inc.

FileName: gfx_gol_window.h

5.2.2.8.29 gfx_imageDecoder.h

This is file gfx_imageDecoder.h.

Functions

	Name	Description
	<u>ImageDecoderInit</u>	This function initializes the global variables to 0 and then initializes the driver. This must be called once before any other function of the library is called
	<u>ImageDecodeTask</u>	This function completes one small part of the image decode function

	ImageLoopCallbackRegister	This function registers the loop callback function so that the decoder calls this function in every decoding loop. This can be used by the application program to do maintainance activities such as fetching data, updating the display, etc...
	IMG_vSetboundaries	

Macros

	Name	Description
	ImageAbort	This function sets the Image Decoder's Abort flag so that decoding aborts in the next decoding loop.
	ImageFullScreenDecode	This function decodes and displays the image on the screen in fullscreen mode with center aligned and downscaled if required
	IMG_ALIGN_CENTER	Flags
	IMG_DECODE_ABORTED	This is macro IMG_DECODE_ABORTED.
	IMG_DOWN_SCALE	This is macro IMG_DOWN_SCALE.
	IMG_FCLOSE	This is macro IMG_FCLOSE.
	IMG_FEOF	This is macro IMG_FEOF.
	IMG_FILE	This is macro IMG_FILE.
	IMG_FOPEN	added by Paolo 9/10/08 for MultiApp Demo
	IMG_FREAD	This is macro IMG_FREAD.
	IMG_FSEEK	This is macro IMG_FSEEK.
	IMG_FTELL	This is macro IMG_FTELL.
	IMG_SCREEN_HEIGHT	This is macro IMG_SCREEN_HEIGHT.
	IMG_SCREEN_WIDTH	The individual image decoders use these defines instead of directly using those provided in the Display driver file
	IMG_vCheckAndAbort	This is macro IMG_vCheckAndAbort.
	IMG_vLoopCallback	This is macro IMG_vLoopCallback.
	IMG_vPutPixel	This is macro IMG_vPutPixel.
	IMG_vSetColor	This is macro IMG_vSetColor.

Types

	Name	Description
	FileFeof	This is type FileFeof.
	FileRead	Typedefs of the used File System APIs
	FileSeek	This is type FileSeek.
	FileTell	This is type FileTell.
	IMG_FILE_FORMAT	IMG_ImageFileFormat specifies all the supported image file formats
	IMG_FILE_SYSTEM_API	IMG_FileSystemAPI holds function pointers to the used File System APIs
	IMG_LOOP_CALLBACK	IMG_LoopCallback is a callback function which is called in every loop of the decoding cycle so that user application can do some maintainance activities
	IMG_PIXEL_OUTPUT	IMG_PixelOutput is a callback function which receives the color information of the output pixel
	IMG_PIXEL_XY_RGB_888	IMG_PixelRgb holds the RGB information of a pixel in BYTES

Variables

	Name	Description
	IMG_bAlignCenter	This is variable IMG_bAlignCenter.
	IMG_bDownScalingFactor	This is variable IMG_bDownScalingFactor.
	IMG_bIAbortImageDecoding	The global variables which define the image position and size

	IMG_pCurrentFile	This is variable IMG_pCurrentFile.
	IMG_pFileAPIs	This is variable IMG_pFileAPIs.
	IMG_PixelXYColor	This is variable IMG_PixelXYColor.
	IMG_pLoopCallbackFn	This is variable IMG_pLoopCallbackFn.
	IMG_pPixelOutput	This is variable IMG_pPixelOutput.
	IMG_wHeight	This is variable IMG_wHeight.
	IMG_wImageHeight	This is variable IMG_wImageHeight.
	IMG_wImageWidth	This is variable IMG_wImageWidth.
	IMG_wStartX	This is variable IMG_wStartX.
	IMG_wStartY	This is variable IMG_wStartY.
	IMG_wWidth	This is variable IMG_wWidth.

5.2.2.8.30 gfx_palette.h

- Module for Microchip Graphics Library
- Palette Support

- FileName: gfx_palette.h
- Dependencies: Graphics.h
- Processor: PIC24, PIC32
- Compiler: MPLAB C30, MPLAB C32
- Linker: MPLAB LINK30, MPLAB LINK32
- Company: Microchip Technology Incorporated

*

- Software License Agreement

*

- Copyright © 2008 Microchip Technology Inc. All rights reserved.
- Microchip licenses to you the right to use, modify, copy and distribute
- Software only when embedded on a Microchip microcontroller or digital
- signal controller, which is integrated into your product or third party
- product (pursuant to the sublicense terms in the accompanying license
- agreement).

*

- You should refer to the license agreement accompanying this Software
- for additional information regarding your rights and obligations.

*

- SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
- KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
- OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
- PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR

- OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
- BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
- DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
- INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
- COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
- CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
- OR OTHER SIMILAR COSTS.

*

- Date Comment

*~~~~~

- 11/06/09 Initial Release
- 08/20/10 Modified `PALETTE_EXTERNAL` to be of type `GFX_EXTDATA`.
- 03/20/12 Modified `PALETTE_ENTRY` structure to have packed attribute.

Functions

	Name	Description
	<u>ClearPaletteChangeError</u>	Clears the Palette change error status
	<u>DisablePalette</u>	Disables the Palette mode.
	<u>EnablePalette</u>	Enables the Palette mode.
	<u>GetPaletteChangeError</u>	Returns the Palette change error status
	<u>IsPaletteEnabled</u>	Returns if the Palette mode is enabled or not.
	<u>PaletteInit</u>	Initializes the color look up table (CLUT).
	<u>RequestPaletteChange</u>	Loads the palettes from the flash during vertical blanking period if possible, otherwise loads immediately.
	<u>SetPalette</u>	Programs a block of palette entries starting from startEntry and until startEntry + length from the flash immediately.
	<u>SetPaletteBpp</u>	Sets the color look up table (CLUT) number of valid entries.
	<u>SetPaletteFlash</u>	Loads the palettes from the flash immediately.

Macros

	Name	Description
	<u>RequestEntirePaletteChange</u>	Macro: RequestEntirePaletteChange(pPalette) Loads all the palette entries from the flash during vertical blanking period if possible, otherwise loads immediately.
	<u>SetEntirePalette</u>	Macro: SetEntirePalette(pPalette) Programs the whole 256 entry palette with new color values from flash.

Structures

	Name	Description
	<u>PALETTE_FLASH</u>	Structure for the palette stored in FLASH memory.
	<u>PALETTE_HEADER</u>	Structure for the palette header.

Types

	Name	Description
	<u>PALETTE_EXTERNAL</u>	Structure for palette stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

Unions

	Name	Description

5.2.2.8.31 gfx_primitive.h

Module for Microchip Graphics Library - Graphic Primitive Layer

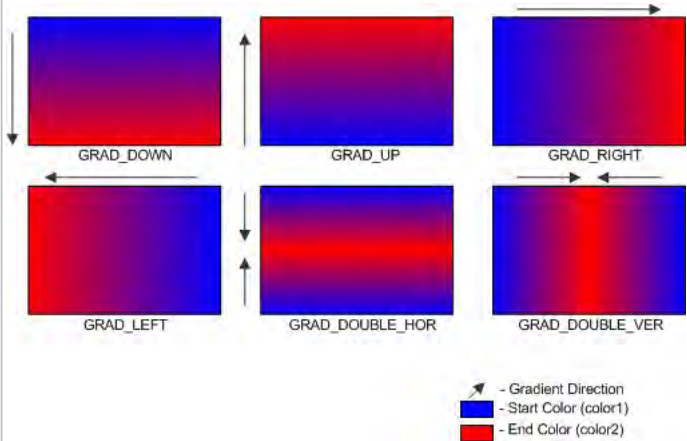
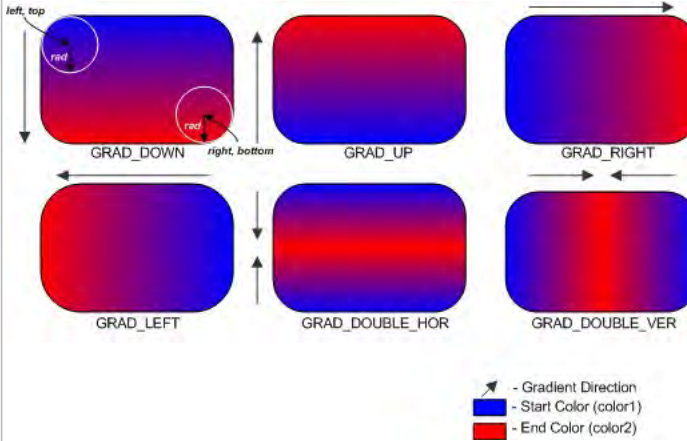
This module implements the primitive rendering routines of the Microchip Graphics Library. The routines are independent of the Display Driver Layer and should be compatible with any Display Driver that is compliant with the requirements of the Display Driver Layer of the Graphics Library.

Enumerations

	Name	Description
	GFX_TRIG_FUNCTION_TYPE	DOM-IGNORE-START

Functions

	Name	Description
	DrawArc	This renders an arc with from startAngle to endAngle with the thickness of r2-r1. The function returns 1 when the arc is rendered successfully and returns a 0 when it is not yet finished. The next call to the function will continue the rendering.
	GFX_AlphaBlendingValueGet	This function returns the current alpha value set for alpha blending rendering.
	GFX_AlphaBlendingValueSet	This function sets the alpha value for alpha blending rendering.
	GFX_BackgroundColorGet	This function returns the color used in the current background.
	GFX_BackgroundImageGet	This function returns the image used in the current background.
	GFX_BackgroundImageLeftGet	This function returns the horizontal starting position of the current background.
	GFX_BackgroundImageTopGet	This function returns the vertical starting position of the current background.
	GFX_BackgroundSet	This function sets the background information.
	GFX_BackgroundTypeGet	This function returns the type of the current background.
	GFX_BackgroundTypeSet	This function sets the background type.
	GFX_BarAlphaDraw	This function renders a alpha blended bar shape using the currently set fill style and color.
	GFX_BarDraw	This function renders a bar shape using the currently set fill style and color.

	GFX_BarGradientDraw	This renders a bar onto the screen, but instead of one color, a gradient is drawn depending on the direction (GFX_GRADIENT_TYPE), length, and colors chosen. This function is a blocking call. 
	GFX_BevelDraw	This is function GFX_BevelDraw.
	GFX_BevelDrawTypeGet	
	GFX_BevelDrawTypeSet	
	GFX_BevelGradientDraw	This renders a filled bevel with gradient color on the fill. It works the same as the fillbevel function, except a gradient out of color1 and color2 is drawn depending on the direction (GFX_GRADIENT_TYPE). This function is a blocking call. 
	GFX_CircleDraw	This function renders a circular shape using the currently set line style and color.
	GFX_CircleFillDraw	This function renders a filled circle shape using the currently set fill style and colors.
	GFX_ColorGet	This function returns the color currently set to render primitive shapes and text.
	GFX_ColorSet	This function sets the color to be used in rendering primitive shapes and text.
	GFX_DoubleBufferAreaGet	
	GFX_DoubleBufferAreaMark	
	GFX_DoubleBufferDisable	
	GFX_DoubleBufferEnable	
	GFX_DoubleBufferStatusGet	

	GFX_DoubleBufferSyncAllStatusClear	
	GFX_DoubleBufferSyncAllStatusGet	
	GFX_DoubleBufferSyncAllStatusSet	
	GFX_DoubleBufferSyncAreaCountGet	
	GFX_DoubleBufferSyncAreaCountSet	
	GFX_DoubleBufferSynchronize	
	GFX_DoubleBufferSynchronizeCancel	
	GFX_DoubleBufferSynchronizeRequest	
	GFX_DoubleBufferSynchronizeSet	
	GFX_DoubleBufferSynchronizeStatusGet	
	GFX_DrawBufferGet	
	GFX_DrawBufferSelect	
	GFX_DrawBufferSet	
	GFX_DRV_FontSet	This is function GFX_DRV_FontSet.
	GFX_DRV_TextStringWidthGet	This is function GFX_DRV_TextStringWidthGet.
	GFX_ExternalCharInfoGet	This is function GFX_ExternalCharInfoGet.
	GFX_ExternalResourceCallback	This function must be implemented in the application. The library will call this function each time when the external memory data will be required. The application must copy requested uint8_ts quantity into the buffer provided. Data start address in external memory is a sum of the address in GFX_EXTDATA structure and offset.
	GFX_FillStyleGet	Return the fill style used when rendering filled shapes.
	GFX_FillStyleSet	Set the fill style to be used when rendering filled shapes.
	GFX_FlashCharInfoGet	This is function GFX_FlashCharInfoGet.
	GFX_FontAntiAliasGet	This function returns the current font anti-aliasing mode.
	GFX_FontAntiAliasSet	This function sets the font anti-aliasing mode.
	GFX_FontGet	This function returns the current font used when rendering strings and characters.
	GFX_FontSet	This function sets the current font used when rendering strings and characters.
	GFX_FrameBufferGet	
	GFX_FrameBufferSelect	
	GFX_GradientColorSet	Sets the gradient fill start and end color.
	GFX_GradientEndColorGet	Return the gradient end color when rendering gradient filled shapes.
	GFX_GradientStartColorGet	Return the gradient start color when rendering gradient filled shapes.
	GFX_ImageDraw	This function renders an image to the frame buffer.
	GFX_ImageHeaderGet	This function fills the given bitmap header with the image's header information.
	GFX_ImageHeightGet	This function returns the height of the given image.
	GFX_ImageOffsetAddressGet	This is function GFX_ImageOffsetAddressGet.
	GFX_ImagePartialDraw	This function renders a portion of an image to the frame buffer.
	GFX_ImageWidthGet	This function returns the width of the given image.
	GFX_Initialize	Initialize the Graphics Library.
	GFX_LineDraw	This function renders a line from x1,y1 to x2,y2 using the currently set line style (see GFX_LineStyleSet()).
	GFX_LinePositionRelativeSet	This function sets the line cursor to a new position relative to the current position.

	GFX_LinePositionSet	This function sets the line cursor to a new position.
	GFX_LinePositionXGet	This function returns the current x position of the line cursor.
	GFX_LinePositionYGet	This function returns the current y position of the line cursor.
	GFX_LineStyleGet	Return the line style used when rendering lines.
	GFX_LineStyleSet	Set the line style to be used when rendering lines.
	GFX_LineToDraw	This function renders a line from current line cursor position (x,y) to (x2,y2) using the currently set line style (see GFX_LineStyleSet()).
	GFX_LineToRelativeDraw	This function renders a line from current line cursor position (x,y) to (x+dX,y+dY) using the currently set line style (see GFX_LineStyleSet()).
	GFX_PolygonDraw	This function renders a polygon using the currently set line style (see GFX_LineStyleSet()) and color (see GFX_ColorSet()).
	GFX_Primitive_Initialize	This is function GFX_Primitive_Initialize.
	GFX_RectangleDraw	This function renders a rectangular shape using the currently set line style and color.
	GFX_RectangleFillDraw	This function renders a filled rectangular shape using the currently set fill style and colors.
	GFX_RectangleRoundDraw	This function renders a rounded corner rectangular shape using the currently set line style and color.
	GFX_RectangleRoundFillDraw	This function renders a filled rectangular shape with round corners using the currently set fill style and colors.
	GFX_RenderingBufferGet	This function is an internal function and should not be called by the application.
	GFX_RenderToDisplayBufferDisable	This is function GFX_RenderToDisplayBufferDisable.
	GFX_RenderToDisplayBufferDisableFlagGet	This is function GFX_RenderToDisplayBufferDisableFlagGet.
	GFX_RenderToDisplayBufferEnable	This is function GFX_RenderToDisplayBufferEnable.
	GFX_ScreenClear	Clears the screen to the currently set color (GFX_ColorSet()).
	GFX_SineCosineGet	This is function GFX_SineCosineGet.
	GFX_SolidLineDraw	This is function GFX_SolidLineDraw.
	GFX_StyledLineDraw	This is function GFX_StyledLineDraw.
	GFX_TextAreaBottomGet	This is function GFX_TextAreaBottomGet.
	GFX_TextAreaBottomSet	This is function GFX_TextAreaBottomSet.
	GFX_TextAreaLeftGet	This is function GFX_TextAreaLeftGet.
	GFX_TextAreaLeftSet	This is function GFX_TextAreaLeftSet.
	GFX_TextAreaRightGet	This is function GFX_TextAreaRightGet.
	GFX_TextAreaRightSet	This is function GFX_TextAreaRightSet.
	GFX_TextAreaTopGet	This is function GFX_TextAreaTopGet.
	GFX_TextAreaTopSet	This is function GFX_TextAreaTopSet.
	GFX_TextCharDraw	This function renders the given character using the currently set color using the currently set font.
	GFX_TextCursorPositionSet	This function sets the text cursor to a new position.
	GFX_TextCursorPositionXGet	This function returns the current x position of the text cursor.
	GFX_TextCursorPositionYGet	This function returns the current y position of the text cursor.
	GFX_TextStringBoxDraw	This function renders the given string using the currently set color and font into a rectangular area.
	GFX_TextStringDraw	This function renders the given string of character using the currently set color using the currently set font.
	GFX_TextStringHeightGet	This function returns the height of the given font.

	GFX_TextStringWidthGet	This function returns the width of the given string using the given font.
	GFX_ThickBevelDraw	This is function GFX_ThickBevelDraw.
	GFX_TransparentColorDisable	This function disables the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.
	GFX_TransparentColorEnable	This function sets the transparent color used in GFX_ImageDraw() functions and enables the transparent color feature.
	GFX_TransparentColorGet	This returns the current transparent color set for the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.
	GFX_TransparentColorStatusGet	This returns the current state of the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.
	GFXCirclePointGet	This is function GFXCirclePointGet.

Macros

	Name	Description
	GFX_CosineGet	This is macro GFX_CosineGet.
	GFX_SineGet	This is macro GFX_SineGet.
	ONEP25	$1.25 * 2^{16}$
	SIN45	$\sin(45) * 2^{16}$

File Name

gfx_primitive.h

Company

Microchip Technology Inc.

5.2.2.8.32 gfx_primitive_local.h

Clock System Service Interface Definition

None.

Enumerations

	Name	Description
	FONT_ORIENTATION	Types of orientation
	IMAGE_STRETCH	Types of image stretching
	LAYER_ACTIONS	

Functions

	Name	Description
	GFX_AlphaBlendWindow	
	GFX_DoubleBufferVisualPageUpdate	
	GFX_Layer	
	GFX_PageSet	

Macros

	Name	Description
	CopyPageWindow	
	GFX_ActivePageGet	Returns the active page.
	GFX_AlphaParamsSet	

	GFX_BUFFER1	This is macro GFX_BUFFER1.
	GFX_BUFFER2	This is macro GFX_BUFFER2.
	GFX_FontAlignmentSet	Sets font orientation vertical or horizontal.
	GFX_GradientTypeSet	This is macro GFX_GradientTypeSet.
	GFX_ImageStretchSet	This is macro GFX_ImageStretchSet.
	GFX_MaxXGet	
	GFX_MaxYGet	
	GFX_PixelPut	This is macro GFX_PixelPut.
	GFX_PixelsPut	This function renders one line of pixels of a given length
	GFX_PrevAlphaColorGet	
	GFX_PrevAlphaColorSet	
	GFX_SetFontOrientation	Sets font orientation vertical or horizontal.
	GFX_VisualPageGet	Returns the visual page.

Structures

	Name	Description
	GFX_FONT_CURRENT	Internal structure for currently set font.
	GFX_GRADIENT_STYLE	This structure is used to describe the gradient style.
	GFX_Primitive_DATA	
	GFX_TRANSITION_PARAMS	
	PUTIMAGE_PARAM	Partial image area to render.

Variables

	Name	Description
	DisplayUpdatePending	This is variable DisplayUpdatePending.
	doubleBuffEnabled	This is variable doubleBuffEnabled.
	GFX_Primitive_instance	structure to store the primitive driver structures
	primitiveTaskImage	This is variable primitiveTaskImage.

File Name

[gfx_primitive.h](#)

Company

Microchip Technology Inc.

5.2.2.8.33 gfx_transitions.h

- Module for Microchip Graphics Library
- Primitive Layer
- Transition

- FileName: Transitions.h
- Dependencies: None
- Processor: PIC24F, PIC24H, dsPIC, PIC32
- Compiler: MPLAB C30, MPLAB C32

- Linker: MPLAB LINK30, MPLAB LINK32
- Company: Microchip Technology Incorporated
- *
 - Software License Agreement
- *
 - Copyright © 2010 Microchip Technology Inc. All rights reserved.
 - Microchip licenses to you the right to use, modify, copy and distribute
 - Software only when embedded on a Microchip microcontroller or digital
 - signal controller, which is integrated into your product or third party
 - product (pursuant to the sublicense terms in the accompanying license
 - agreement).
- *
 - You should refer to the license agreement accompanying this Software
 - for additional information regarding your rights and obligations.
- *
 - SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 - KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
 - OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
 - PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
 - OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
 - BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
 - DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
 - INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
 - COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
 - CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
 - OR OTHER SIMILAR COSTS.
- *
 - Author Date Comment
- *~~~~~
 - Microchip 24/12/2010 Initial version

Enumerations

	Name	Description
	<u>GFX_TRANSITION_DIRECTION</u>	Direction enumeration to determine the direction of the selected <u>GFX_TRANSITION_TYPE</u> .
	<u>GFX_TRANSITION_TYPE</u>	Transition type enumeration to determine the type of the transition. Each type will require specific parameters when setting up the transition operation (GFXSetupTransition() or GFXTransition()).

Functions

	Name	Description
	GFX_SetTransitionParameters	This sets up the transition effect using the GFX_TRANSITION_TYPE and the given parameters. The actual transition execution will occur when <code>GFXExecutePendingTransition()</code> is called. When <code>DOUBLE_BUFFERING</code> is enabled, <code>GFXExecutePendingTransition()</code> is executed after the current screen is fully rendered.
	GFX_Transition	This immediately executes the transition effect using the GFX_TRANSITION_TYPE and the given parameters.

5.2.2.8.34 gfx_types_font.h

Graphics Resource Types Header for Microchip Graphics Library

This header defines the different structures used for fonts resources in the Microchip Graphics Library.

Macros

	Name	Description
	GFX_FONT_SPACE	Font space section. The fonts can be located in psv (constant) or program space in PIC24/dsPIC MCUs. This define allows for switching of the pointer type used to access the font structure in memory See: <code>GraphicsConfig.h</code> for the application define.
	GFX_TYPES_FONTS_H	This is macro <code>GFX_TYPES_FONTS_H</code> .
	GFX_UXCHAR	This is macro <code>GFX_UXCHAR</code> .
	GFX_XCHAR	Configuration for the text character size.

Structures

	Name	Description
	GFX_FONT_GLYPH_ENTRY	Structures describing the glyph entry.
	GFX_FONT_GLYPH_ENTRY_EXTENDED	This is type <code>GFX_FONT_GLYPH_ENTRY_EXTENDED</code> .
	GFX_FONT_HEADER	Structure describing the font header.
	GFX_FONT_OUTCHAR	Structure for character information when rendering the character.

File Name

`gfx_font_types.h`

Company

Microchip Technology Inc.

5.2.2.8.35 gfx_types_image.h

Graphics Resource Types Header for Microchip Graphics Library

This header defines the different structures used for images resources in the Microchip Graphics Library.

Macros

	Name	Description
	GFX_TYPES_IMAGE_H	This is macro <code>GFX_TYPES_IMAGE_H</code> .

Structures

	Name	Description
	GFX_MCHP_BITMAP_HEADER	Structure describing the bitmap header.

Types

	Name	Description
	int16_gfx_image_prog	This is type int16_gfx_image_prog.
	int16_prog	This is type int16_prog.
	int16_prog_pack	This is type int16_prog_pack.
	int32_gfx_image_prog	This is type int32_gfx_image_prog.
	int32_prog	This is type int32_prog.
	int32_prog_pack	This is type int32_prog_pack.
	int8_gfx_image_prog	This is type int8_gfx_image_prog.
	int8_prog	This is type int8_prog.
	int8_prog_pack	This is type int8_prog_pack.
	uint16_gfx_image_prog	This is type uint16_gfx_image_prog.
	uint16_prog	This is type uint16_prog.
	uint16_prog_pack	This is type uint16_prog_pack.
	uint32_gfx_image_prog	This is type uint32_gfx_image_prog.
	uint32_prog	This is type uint32_prog.
	uint32_prog_pack	This is type uint32_prog_pack.
	uint8_gfx_image_prog	GFX Image data Types
	uint8_prog	General program space data types
	uint8_prog_pack	This is type uint8_prog_pack.

File Name

gfx_image_types.h

Company

Microchip Technology Inc.

5.2.2.8.36 gfx_types_macros.h

Module for Microchip Graphics Library

This module defines the data types used in the Microchip Graphics Library for use of Primitive and Driver Layers.

Enumerations

	Name	Description
	GFX_ALIGNMENT	Defines the types of text alignment. The text alignment is used in GFX_TextStringBoxDraw() .
	GFX_BACKGROUND_TYPE	Defines the types of background information. Knowing the background information allows easier refresh of an area on the display buffer.
	GFX_BEVEL_RENDER_TYPE	Bevel shapes can be rendered in full or half size. The following types defines the full, top and bottom halves.
	GFX_FEATURE_STATUS	Defines the states of a Graphics Library feature that can nbe enabled/disabled at run time.
	GFX_FILL_STYLE	Defines the types of fill styles.

	GFX_FONT_ANTIALIAS_TYPE	Transparency type when rendering characters with anti-aliasing.
	GFX_LINE_STYLE	Defines the types of line styles.
	GFX_STATUS	Status types for rendering.
	GFX_STATUS_BIT	Bit masks for additional status information.

Macros

	Name	Description
	ConvertColor25	This is macro ConvertColor25.
	ConvertColor50	75, 50, 25 percentage color conversion macros. Used in converting colors to these percentages when performing alpha blending of colors using software alpha blending operation.
	ConvertColor75	This is macro ConvertColor75.
	GFX_MAX_INVALIDATE_AREAS	This is macro GFX_MAX_INVALIDATE_AREAS.
	GFX_RGBConvert	Macros: GFX_RGBConvert(red, green, blue) This macro converts the color data to the selected COLOR_DEPTH . This macro is only valid when COLOR_DEPTH is 8, 16, or 24.

Structures

	Name	Description

5.2.2.8.37 gfx_types_palette.h

FileName: gfx_palette_types.h Dependencies: See Includes Section Processor: PIC24, PIC32 Compiler: XC16/XC32 Linker: XC16/XC32

Macros

	Name	Description
	GFX_TYPES_PALETTE_H	This is macro GFX_TYPES_PALETTE_H.

Unions

	Name	Description

5.2.2.8.38 gfx_types_resource.h

Graphics Resource Types Header for Microchip Graphics Library

This header defines the different structures used for the Graphics Resources used in the Graphics Library. This also contains helpful macros as well as data types used.

Enumerations

	Name	Description
	GFX_RESOURCE	Memory type enumeration to determine the source of data. Used in interpreting images, fonts and palette from different memory sources.

Macros

	Name	Description
	GFX_COMP_MASK	This is macro GFX_COMP_MASK.
	GFX_MEM_MASK	This is macro GFX_MEM_MASK.
	GFX_TYPE_MASK	This is macro GFX_TYPE_MASK.
	GFX_TYPES_RESOURCE_H	This is macro GFX_TYPES_RESOURCE_H.

	MCHP_BITMAP_NORMAL	no compression, palette is present for color depth = 8, 4 and 1 BPP
	MCHP_BITMAP_PALETTE_STR	palette is provided as a separate object (see PALETTE_HEADER) for color depth = 8, 4, and 1 BPP, ID to the palette is embedded in the bitmap.

Structures

	Name	Description
	GFX_RESOURCE_BINARY	Structure for Graphics Resource - binary data.
	GFX_RESOURCE_FONT	Structure for Graphics Resource - Fonts.
	GFX_RESOURCE_HDR	Structure for Graphics Resource Header. This is the common header for all Graphics Resources.
	GFX_RESOURCE_IMAGE	Structure for Graphics Resource - image.
	GFX_RESOURCE_PALETTE	Structure for Graphics Resource - Palette.

File Name

gfx_resource_types.h

Company

Microchip Technology Inc.

5.2.3 Graphics Primitive Library

5.2.3.1 Introduction

Graphics Primitive Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Graphics Primitive Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the library without the necessity of interacting directly with a graphics controller, there by hiding differences from one graphics controller variant to another.

Description

This module is part of a larger Graphics Library. The Primitive Layer is used to [send](#) rendered shape data to the appropriate graphics controller driver. The primitive layer is also a scheduler to make sure that shapes are rendered in the proper order. This helps ensure a widget appears on the screen correctly. Shapes (Bars, Lines, Bevels, etc.) also include text and images.

5.2.3.2 Release Notes

MPLAB Harmony Version: v0.70b **Graphics Primitive Library Version :** 4.00b **Release Date:** 18Nov2013

Contents:

This release consists of the Graphics Primitive Layer.

Graphic Layer	Description
Primitive	The primitive layer sends rendered shape and image data to the appropriate graphics controller driver

New This Release:

A summary of the new features is as follows:

- Renamed Primitive API functions
- Remove instance argument from APIs

Feature	Description	Comments
Multi-Display support	Only one LCD display is supported in this release.	Support available in upcoming release.

Known Issues:

- No multi-display support

Feature	Category	Description	Work-around
Multi-Display support	Driver	Only one LCD display is supported in this release.	Support available in upcoming release.

5.2.3.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.2.3.4 Using the Library

This topic describes the basic architecture of the Graphics Primitive Library and provides information and examples on how to use it.

Interface Header File: [gfx_primitive.h](#)

The interface to the Graphics Primitive library is defined in the "primitive.h" header file. This file is included by the "[gfx.h](#)" file. Any C language source (.c) file that uses the Graphics Primitive library should include "primitive.h".

5.2.3.4.1 Abstraction Model

The Primitive Layer provide a set of low level drawing functions. These functions include drawing lines, rectangles, circles, text, and bitmap images.

5.2.3.4.2 Library Overview

Refer to the section [Driver Overview](#) for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Graphics module.

Library Interface Section	Description
Setup and Management Functions	Contains the APIs for setting up primitive rendering functions and managing the primitive layer
Text Functions	Contains the APIs for rendering text
Shape Functions	Contains the APIs for shapes that can be rendered
Image Functions	Contains the APIs for rendering images
Color Functions	Contains the APIs for setting/getting colors
Double Buffering Functions	Contains the APIs for setting double buffering

5.2.3.4.3 How the Library Works

The primitive layer is defined as the mediator between the graphics object layer and the graphics controller driver layer. Once the primitive layer is initialized the application just has to make calls to which specific primitive it is rendering. This layers serves other purposes:

- Schedules primitive tasks to ensure proper primitive rendering. This feature makes it possible to know that when an object is called to be drawn on a screen before another object that the order of these calls will be kept intact. This is important when using features where the image on the LCD matters before making the function call.
- In the case of multiple graphics controller drivers, it organizes which driver to make the primitive update to.

Below is a flow chart showing how the primitive layer works in conjunction with the driver layer and the object layer.

<FlowChart>

5.2.3.4.3.1 Core Functionality

The primitive layer consists of a command based circular buffer. Primitives are scheduled in the buffer and the commands are serviced each time `primitivetasks()` function is called.

Description

The library relies on a graphics controller driver to render images to an LCD panel. Below shows how the primitive layer files interact with the driver layer files.

<PrimitiveLayerOrg>

5.2.3.4.4 Using Primitive Rendering Functions

Basic rendering functions such as `LineDraw()`, `BarDraw()`, `CircleDraw()` etc are referred to as functions in the Graphics Primitive

Layer. These functions can also be implemented in the device driver layer if the display device supports hardware acceleration of the function. Applications that directly calls these functions can take advantage of the hardware accelerated primitives. How these functions are used will depend on the configuration.

Description

All primitive rendering functions returns a handle to their location in the queue.

```
// all primitives are non-blocking calls
while(GFX_PRIM_LineDraw(instance,a,b) == NULL);
while(GFX_PRIM_BarDraw(instance,c,d,e,f) == NULL);
while(GFX_PRIM_BarFill(instance,c+2, d+2, e-2, f-2) == NULL);
```

If the while check is not in place, it possible that the only primitive that you will see in the screen is the GFX_PRIM_LineDraw().

For case 2, one can also be creative in the application code and implement some form of non-blocking scheme and make use of the time while waiting for the primitives to render.

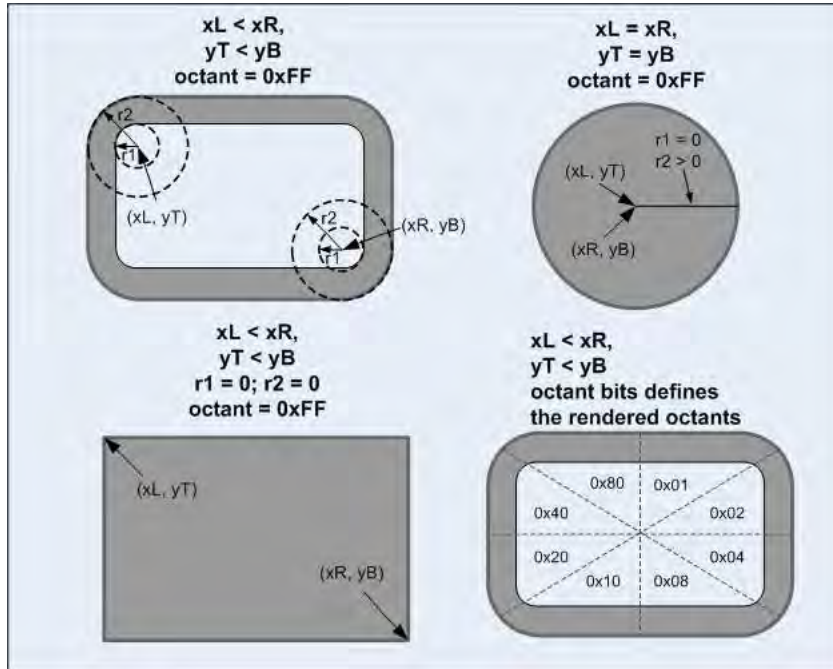
Another example for case 2:

```
uint16_t DrawMyFigure(instance, a, b, c, d, e, f)
{
    typedef enum {
        DRAW_LINE,
        DRAW_RECT,
        DRAW_BAR,
    } DRAW_MYFIGURE_STATES;
    static DRAW_MYFIGURE_STATES state = DRAW_LINE;

    switch(state){
        case DRAW_LINE:
            if (GFX_PRIM_LineDraw(instance, a, b) == NULL)
                return 0;
            state = DRAW_RECT;
        case DRAW_RECT:
            if(GFX_PRIM_BarDraw(instance,c,d,e,f) == NULL)
                return 0;
            state = DRAW_BAR;
        case DRAW_BAR:
            if(GFX_PRIM_BarFill(instance, c+2, d+2, e-2, f-2) == NULL);
                return 0;
            state = DRAW_LINE;
            return 1;
    }
}
```

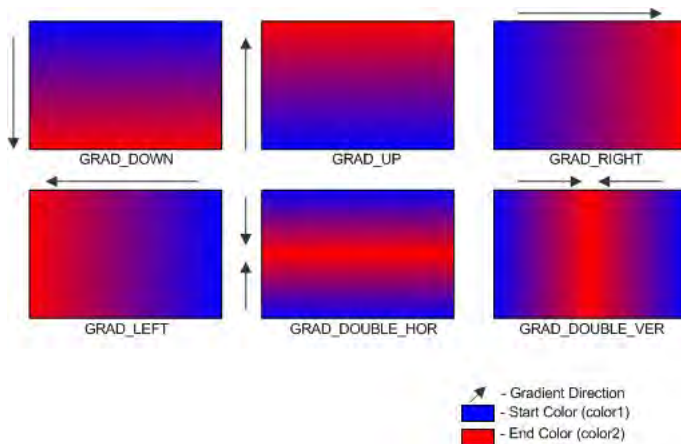
This non-blocking code can be used in the application and the application can do other tasks whenever DrawMyFigure() returns 0. Application should call DrawMyFigure() again until it return a 1 signifying that the Line, Rectangle and GFX_PRIM_BarDraw were drawn successfully.

5.2.3.4.4.1 Arcs

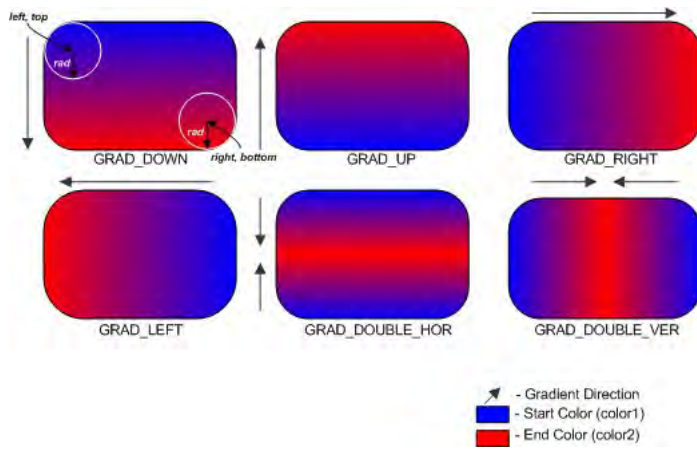


5.2.3.4.4.2 Gradients

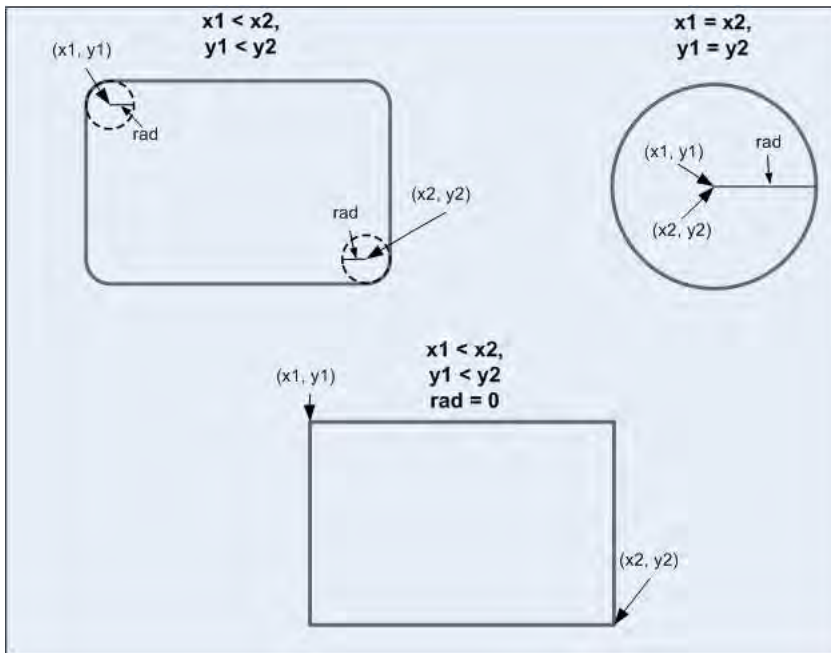
Below is an image that helps describe the `GFX_PRIM_BarGradient()` function in more detail.

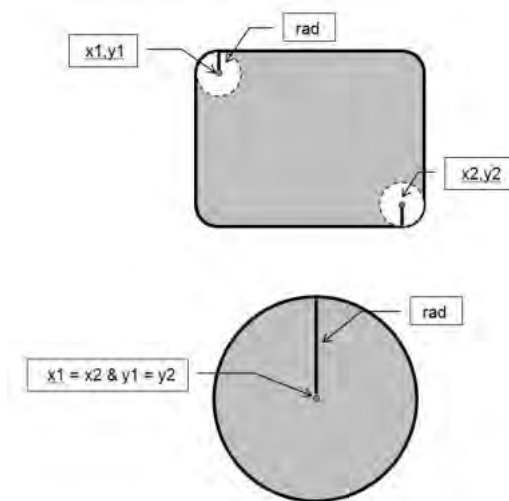


Below is an image that helps describe the `GFX_PRIM_BevelGradient()` function in more detail.



5.2.3.4.4.3 Bevels





5.2.3.4.4 Alpha Blending

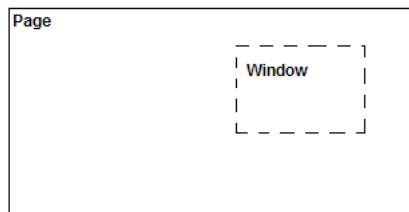
This section helps describe how alpha blending works in the primitive layer.

Description

Alpha-Blend support are two levels. `USE_ALPHABLEND_LITE` and `USE_ALPHA_BLEND`.

Alpha Blend Lite is a "desructive" alpha blend where the blending happens in one frame or window. Once the colors are blending the original color data in the visual page is lost. This method only requires one frame and `GetPixel` type functionality to work.

The other Alpha Blending uses blocks of pixels in the memory called Windows. A window can be any rectangular area in a display buffer. The display buffer is also considered as a page. For a QVGA display buffer a page would be 320x240 pixels. A window is a certain width and height contained inside a page.

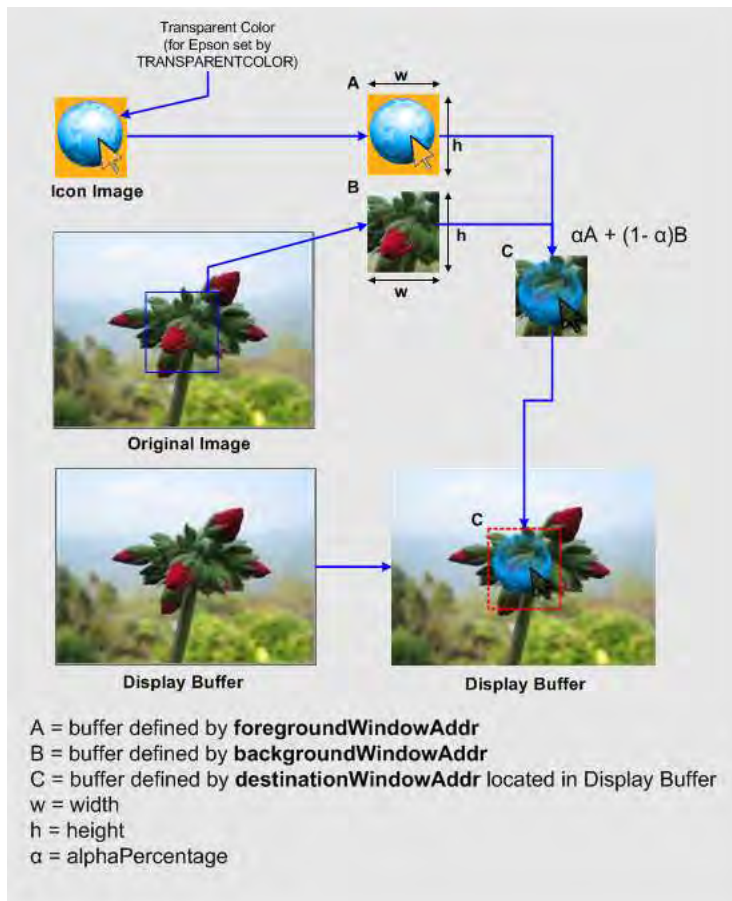


Alpha blending equation:

$$\text{OutPut Window} = \text{Foreground Window} * (\text{Alpha}) + \text{Background Window} * (1-\text{Alpha})$$

when done in software requires a lot of CPU bandwidth. Epson Controller implements alpha blending in hardware which offloads the CPU.

The Display Driver Layer of the Graphics Library utilizes the Epson Controller alpha blending through the `AlphaBlendWindow()` function. Three windows are specified with equal widths and heights. An alpha parameter is passed to define the level of alpha blending. The logical flow of the operation is shown below:



An icon image with a given width (w) and height (h) is needed to be alpha blended into the display buffer.

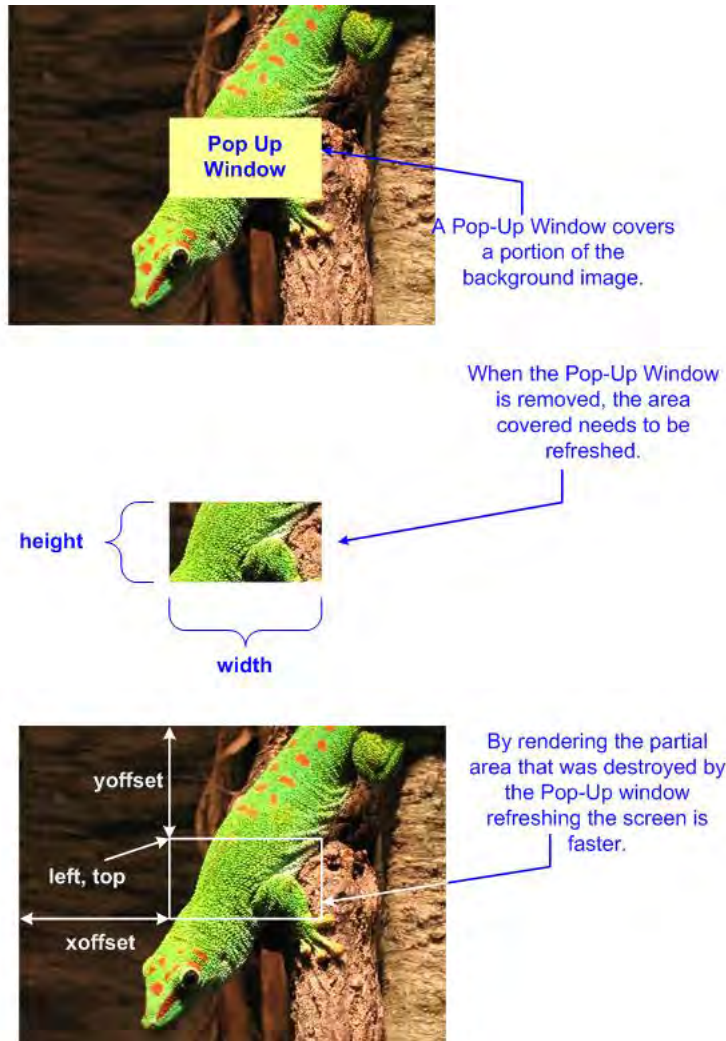
- Buffer A is allocated in memory and its location set by `foregroundWindowAddr`
- Buffer B is allocated in memory and its location set by `backgroundWindowAddr` and
- Buffer C is allocated in memory and its location set by `destinationWindowAddr`. Note that the destination window can be located within the display buffer. Doing this removes an intermediate step after `AlphaBlendWindow()` call to put the result of alpha blend in the display buffer.

The final location of the icon on the Display Buffer is used to locate the affected pixels in the Display Buffer. These affected pixels are copied into Buffer B while Buffer A is filled up with the Icon Image. Once Buffers A and B are ready they are alpha blended to Buffer C. Buffer C is then copied to the Display Buffer.

Note that the Icon Image has a background color of **ORANGE**. To have the effect of the final output Display Buffer, set the Epson Controllers transparent color to **ORANGE**. This is set in the `TRANSPARENTCOLOR` macro defined in the Epson S1D13517 Driver. This `TRANSPARENTCOLOR` macro is not to be confused with the `TransparentColorEnable()` function of the Display Device Driver Level Primitives. `TRANSPARENTCOLOR` is set at build time. In the future release of the Epson driver, this will be converted to use the `TransparentColorEnable()` function.

5.2.3.4.4.5 Images

The image support on the primitive layer can be done "partially" to achieve more functionality in where a background can be updated more efficiently. More details are below:



5.2.3.4.5 Defining Colors

In most cases, the application will define its own set of colors and not use the default colors that comes with the Graphics Library. This section shows an example on how to do it.

Description

To override or define a new set of colors follow this steps:

1. Create a new color header file. The color values and data types will depend on the `GFX_COLOR` type used. This data type is defined by the `COLOR_DEPTH` macro. See `COLOR_DEPTH` for details.
2. In the application code, include the created color header file ahead of the `#include "Graphics/Graphics.h"`. When `Graphics.h` includes the `gfxcolors.h` it will ignore color macros that has been defined already in the new color header file.

In the GOL Palette Demo, there is an example on how it is done. In that project there is a file in the application (or project directory) named `PaletteColorDefines.h`. This file contains all the color definition used in the demo. The main header file of the demo (`Main.h`) includes the `PaletteColorDefines.h` file ahead of the Graphics Library header files. Since the `PaletteColorDefines.h` declared the color values, the macros redefined in `gfxcolors.h` will be ignored. How is this done? If you look at the `gfxcolors.h` file you will notice that all colors defined have a check (for example `BLACK()`):

```
#ifndef BLACK()
```

```
#define BLACK() 0
#endif
```

So if `BLACK()` is defined previously, the definition in `gfxcolors.h` will not take effect.

5.2.3.4.6 Changing the Default Font

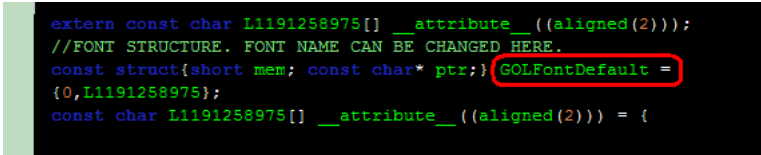
The library comes with the default font (Gentium 18). This font can be changed in two ways.

This instructions assumes that you have performed the conversion of your raster font or True Type font into C file. For the conversion of your raster font please see the help file of the Graphics Resource Converter utility that comes with the Graphics Library.

There are two ways to replace the default font in the Graphics Library.

Renaming User Font to GOLFontDefault:

1. Open the generated font file (generated by the "Graphics Resource Converter" utility) and change the font structure name to **GOLFontDefault** (see figure below).



```
extern const char L1191258975[] __attribute__((aligned(2)));
//FONT STRUCTURE. FONT NAME CAN BE CHANGED HERE.
const struct{short mem; const char* ptr;} GOLFontDefault =
{0,L1191258975};
const char L1191258975[] __attribute__((aligned(2))) = {
```

2. Remove the file **GOLFONTDefault.c** in the project and replace it with your own font file that contains the **GOLFontDefault**.
3. Build and test your new font.

Replacing the GOLFontDefault with any user defined fonts

1. Open the **GraphicsConfig.h** file and add the following lines:

```
#define FONTDEFAULT yourFontName
```

2. In the project, add the generated font file (generated by the "Graphics Resource Converter" utility).
3. Build and test your new font.

In this method, `GOL.h` and `GOLFontDefault.c` files checks if `FONTDEFAULT` is defined. If it is, it skips the declaration of the `GOLFontDefault` and uses the user defined font.

5.2.3.4.7 Advanced Font Features

This topic explains anti aliasing, glyphs, and advanced font features.

Description

Fonts used in the library can be configured to use anti-aliasing and extended glyph support.

Font Anti-Aliasing

Anti-aliasing is a technique used to make the edges of text appear smooth. This is useful especially with characters like 'A', 'O', etc which has slant or curved lines. Since the pixels of the display are arranged in rectangular fashion, slant edges can't be represented smoothly. To make them appear smooth, a pixel adjacent to the pixels is painted with an average of the foreground and background colors as depicted in Figure 1.



Figure 1: Font with Anti-Aliasing

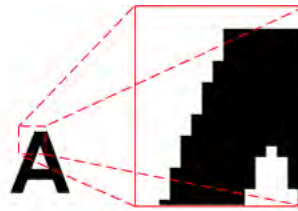


Figure 2: Font with No Anti-Aliasing

When anti-aliasing is turned off, the pixels abruptly changes from background color to foreground color shown in Figure 2. To implement anti-aliasing, adjacent pixels transitions from background to foreground color using 25% or 75% mid-color values from background to foreground colors. This feature in fonts will require roughly twice the size of memory storage required for font glyphs with no anti-aliasing.

Since the average of foreground and background colors needs to be calculated at runtime, the rendering of anti-aliased fonts take more time than rendering normal fonts. To optimize the rendering speed, a macro named `GFX_Font_SetAntiAliasType()` is available where anti-alias type can be set to `ANTIALIAS_OPAQUE` or `ANTIALIAS_TRANSLUCENT`.

- `ANTIALIAS_OPAQUE` (default after initialization of graphics) - mid colors are calculated once while rendering each character which is ideal for rendering text over a constant background.
- `ANTIALIAS_TRANSLUCENT` - the mid values are calculated for every necessary pixel and this feature is useful while rendering text over an image or on a non-constant color background.

As a result, rendering anti-aliased text takes longer with `ANTIALIAS_TRANSLUCENT` type than compared to `ANTIALIAS_OPAQUE` type.

To use anti-aliasing, enable the compiler switch `#define USE_ANTIALIASED_FONTS` in the `GraphicsConfig.h` file and enable the anti-alias checkbox in the Graphics Resource Converter (GRC) tool while selecting the font.

Note: Even when anti-aliasing is enabled, normal fonts can be used without the antialias effect.

Extended Glyphs

Extended glyphs are needed to render characters of certain languages which use more than one byte to represent a single character. For example: Asian languages like Thai, Hindi, etc. In these character set, more than one glyph overlaps each other to form a single character of that language as shown in Figure 3. To use this feature, enable the Extended Glyph checkbox in the Graphics Resource Converter (GRC) tool while selecting the font.

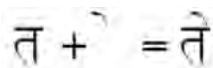


Figure 3: Example of a Character that is Formed by Two Overlapping Glyphs

Note: The fonts used with extended glyphs are normal ANSI fonts and not Unicode fonts.

5.2.3.4.8 Double Buffering

In the Microchip Graphics Library, if double-buffering is enabled, the frame buffer and draw buffer are exchanged after the changes of all the widgets on a screen are done (i.e., the new screen appears after the whole screen is updated and not after updating an individual widget). This feature is enabled only on the following drivers:

- Microchip Graphics Display Driver - customizable driver for RGB Glass. Currently used in PIC24FJ256DA210 device family.
- Microchip Low-Cost Controllerless (LCC) Graphics Display Driver - customizable driver for RGB Glass. Currently used for selected PIC32MX device families.

Description

Manipulating pixels on the screen requires direct writes to the frame buffer. While these changes are being executed, the screen is also refreshed. This means that the changes are displayed immediately as the frame buffer is being updated. This is not a suitable scheme when the changes are slow, for example, decoding an image or having a large number of widgets on a screen. The display may appear slow in that case. One solution to this problem is to use a double-buffering scheme supported by the Microchip Graphics Library. In this scheme, the changes are not directly written to the frame buffer, but instead, they are written to a separate buffer, called the 'Draw Buffer'. After all the changes are made, the draw buffer and frame buffer are exchanged. Now the draw buffer becomes the frame buffer, and because of all the changes drawn there, the changes appear spontaneous to the user. Of course, there will be a delay, as all the changes have to be written to the draw buffer before displaying it. This delay is generally more tolerable than displaying the changes slowly. After exchanging of the buffers, the latest buffer (which is now the frame buffer) is copied to the new draw buffer in the background and then the new draw buffer is in sync with what is being displayed. New changes are then written to the draw buffer and the cycle continues. As the double-buffering scheme uses two buffers, the RAM requirement will double.

In the Microchip Graphics Library, if double-buffering is enabled, the frame buffer and draw buffer are exchanged after the changes of all the widgets on a screen are done (i.e., the new screen appears after the whole screen is updated and not after updating an individual widget).

The work flow of double-buffering is graphically explained along with tips on deciding when to use double buffering in the APPENDIX B of the Application note AN1368: Developing Embedded Graphics Applications using PIC® Microcontrollers with Integrated Graphics Controller.

To use double buffering in an application, follow the steps described below:

1. Enable the option `USE_DOUBLE_BUFFERING` in `GraphicsConfig.h`
2. Update `GFX_DISPLAY_BUFFER_LENGTH` to include both the buffers in `HardwareProfile.h`. Note that when using external memory the `GFX_DISPLAY_BUFFER_LENGTH` must fit into the external memory size. For example in PIC24FJ256DA210 external memory can be added using EPMP. External memory mapped to the EPMP must be big enough to accommodate the display buffers required by double buffering. See Set Up Display Interface for more information on memory requirements.

3. Check the jumper settings to enable the required RAM address space on the development board

If Graphics Objects Layer (GOL) is used in the application, the switching of buffers is handled automatically in order to keep the switching task transparent to the users. If double buffering is enabled in applications using only the Primitive layer, then the switching of buffers has to be handled by the application itself as explained in the following steps.

Steps required for manually handling the switching of buffers:

1. After `InitGraph()` is called, call the APIs `InvalidateAll()` followed by `UpdateDisplayNow()`. The two buffers are properly setup after these calls and from this point onwards, the drawing happens on the draw-buffer.
2. When a shape is drawn on the draw buffer, that rectangular area has to be marked as invalid by using the API `InvalidateRectangle(left, top, right, bottom)`. Only the invalidated rectangular areas are copied to the frame buffer in order to reduce the copy operations thereby reducing the overall time and energy required to sync the two buffers. Hence, it is important to invalidate the changed rectangular areas failing which the change doesn't show up on the display.
3. Call either `RequestDisplayUpdate()` or `UpdateDisplayNow()` to sync the two buffers and as a result the changes appear on the display. The former API exchanges the buffers during the next display blanking period on TFT LCDs causing the change to appear smooth and immediate whereas the latter API exchanges the two buffers at the time of the API call probably causing a slight flicker on the display. On displays which doesn't support blanking periods (e.g. CSTN LCDs), the effect of `RequestDisplayUpdate()` is same as that of `UpdateDisplayNow()`.

Even if double buffering is enabled at compile time, it can be switched off and on at run time using APIs `SwitchOffDoubleBuffering()` and `SwitchOnDoubleBuffering()`. Switching double buffering on/off at runtime is useful in applications which need some screens having fast updates like waveform or animation which requires double buffering to be switched off and some other screens where double buffering is beneficial.

Note: In applications using Graphics Objects Layer and double buffering, the double buffering is not immediately enabled after the API `GFX_OBJ_Initialize()` is called in order to support hassles splash screens but is automatically enabled from the second screen update onwards. If double buffering is needed from the first screen itself, then follow step 1 immediately after calling `GFX_OBJ_Initialize()`.

5.2.3.4.9 Transitions

This section describes screen transition effect when changing screens.

Description

Applications often require some transition effects while changing screens in order to enhance the look and feel which can be achieved by using Transition APIs provided with the Microchip Graphics Library. Refer to Appendix C of the application note [AN1368: Developing Embedded Graphics Applications using PIC® Microcontrollers with Integrated Graphics Controller](#) for a graphical illustration of how transitions can be achieved.

In order to translate a new screen, the new screen has to be developed in a separate buffer and has to be copied to the frame buffer (visible display buffer) part by part. The way the new screen data is copied into the frame buffer determines the transition effect. A number of transition effects can be achieved by using the API: `GFXTransition(left, top, right, bottom, type, srcpageaddr, destpageaddr, delay_ms, param1, param2)`. This API copies the rectangular area defined by left, top, right and bottom from address defined by `srcpageaddr` to the `destpageaddr` as per the transition defined by type, param1 and param2. The

srcpageaddr should contain the new screen data and the destpageaddr must be the frame buffer address. The speed of the transition can be configured by the parameter delay_ms which determines the delay between each copy operations. Note that this API must be called only at the end of GFX_OBJ_DrawCallback() function in order to ensure that the new screen is fully created in the RAM.

If double buffering is enabled, then using transitions is made easier by another API: GFXSetupTransition(left, top, right, bottom, type, delay_ms, param1, param2). This API can be called immediately after creating the new screen and the graphics library will store the request and initiate the transition after the new screen is fully created in the RAM. Note that this API doesn't have address parameters as the address of draw-buffer is already known to the double buffering subsystem of the graphics engine.

5.2.3.5 Configuring the Library

The configuration of the Graphics Primitive Library is based on the file system_config.h

This header file contains the configuration selection for the Graphics Primitive Library. Based on the selections made, the Graphics Primitive Library will support or not support selected features. These configuration settings will apply to all instances of the Graphics Primitive Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.2.3.5.1 Fonts

This compile option selects if the support for unicode fonts, unsigned char or the default signed char type fonts.

There are three types of font (characters) that can be used in the Graphics Library. This gives the user the option to implement multi-language application or use the default signed char type.

Define in GraphicsConfig.h	XCHAR type	Description
#define USE_MULTIBYTECHAR	#define XCHAR unsigned short	Enable support for multi-byte fonts such as Unicode fonts.
#define USE_UNSIGNED_XCHAR	#define XCHAR unsigned char	Enable support for character range of 0-255.
none of the two are defined	#define XCHAR char	Character range is set to 0-127.

Note: Only one of the two or none at all are defined in GraphicsConfig.h.

- #define [USE_MULTIBYTECHAR](#)

- #define USE_UNSIGNED_XCHAR

- when none are defined, XCHAR defaults to type char.

See XCHAR for details.

5.2.3.5.2 Alpha Blending

This compile option enables the Alpha-Blend feature in Primitive Layer.

5.2.3.5.3 Colors

Defines the color depth of the system

5.2.3.5.4 Images

5.2.3.6 Building the Library

This section list the files that are available in the \src of the primitive driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- [gfx.h](#)

5.2.3.7 Library Interface

Data Types and Constants

	Name	Description
	GFX_CosineGet	This is macro GFX_CosineGet.
	GFX_GradientTypeSet	This is macro GFX_GradientTypeSet.
	GFX_SineGet	This is macro GFX_SineGet.

Setup and Management Functions

	Name	Description
	GFX_SineCosineGet	This is function GFX_SineCosineGet.

Shape Functions

	Name	Description
	GFXCirclePointGet	This is function GFXCirclePointGet.

Description

This section describes the Application Programming Interface (API) functions of the Graphics Primitive Library.

5.2.3.7.1 Setup and Management Functions

5.2.3.7.1.1 GFX_SineCosineGet Function

C

```
int16_t GFX_SineCosineGet(
    int16_t v,
    GFX_TRIG_FUNCTION_TYPE type
);
```

Description

This is function GFX_SineCosineGet.

5.2.3.7.2 Text Functions

5.2.3.7.3 Shape Functions

5.2.3.7.3.1 GFXCirclePointGet Function

C

```
void GFXCirclePointGet(  
    int16_t radius,  
    int16_t angle,  
    int16_t * x,  
    int16_t * y  
);
```

Description

This is function GFXCirclePointGet.

5.2.3.7.4 Color Functions

5.2.3.7.5 Draw Functions

5.2.3.7.6 Image Functions

5.2.3.7.7 Double Buffering Functions

5.2.3.7.8 Data Types and Constants

5.2.3.7.8.1 GFX_CosineGet Macro

C

```
#define GFX_CosineGet(angle) GFX_SineCosineGet(angle, GFX_TRIG_COSINE_TYPE)
```

Description

This is macro GFX_CosineGet.

5.2.3.7.8.2 GFX_GradientTypeSet Macro

C

```
#define GFX_GradientTypeSet(value) GFX_Primitive_instance.gradientStyle.type = value
```

Description

This is macro GFX_GradientTypeSet.

5.2.3.7.8.3 GFX_SineGet Macro

C

```
#define GFX_SineGet(angle) GFX_SineCosineGet(angle, GFX_TRIG_SINE_TYPE)
```

Description

This is macro GFX_SineGet.

5.2.3.8 Files

The Graphics Primitive Library files include:

- [gfx_colors.h](#)
- [gfx_primitive.h](#)
- [gfx_transitions.h](#)
- [gfx_types_palette.h](#)
- [gfx_config_template.h](#)

5.3 Cryptographic (Crypto) Library

5.3.1 Introduction

Cryptographic Library for Microchip Microcontrollers

This library provides a software Cryptographic Library that is available on the Microchip family of microcontrollers with a convenient C language interface.

Description

The Cryptographic Library includes functions to perform encryption, decryption, hashing, authentication, and compression within the embedded application. Random number generation (RNG) functions are also provided.

Block Ciphers

The library provides DES, 3DES, and AES for block cipher needs. Depending on the algorithm in use, CBC and CTR modes are supported.

Public Key Cryptography

The library provides RSA and Elliptic Curve Cryptography (ECC) for Public Key Cryptography, and Diffie-Hellman (DH) for key agreement arrangements.

Hash Functions

The library provides MD5, SHA, SHA-256, SHA-384, and SHA-512 for hashing. These functions do not require keys or initialization vectors (IV).

Random Number Generation Functions

The library provides functions to generate either a single pseudo-random number, or a block of such numbers.

5.3.2 Release Notes

MPLAB Harmony Version: v0.70b **Cryptographic Library:** 1.00 **Release Date:** 18Nov2013

New This Release:

Cryptographic Library

- Provides cryptographic, authentication, compression, and random number routines for use on PIC32MX and PIC32MZ families
- Authentication capabilities:

- MD5
- SHA-1
- SHA-256
- SHA-384
- SHA-512
- HMAC
- Cryptographic capabilities:
 - AES:
 - 128-, 192-, 256-bit key lengths
 - CBC, CTR, GCM, and CCM-8 modes
 - RSA
 - DES/Triple DES
 - Elliptic Curve Cryptography (ECC)
- Random Number capabilities:
 - Single random number
 - Block of random numbers
- Compression capabilities:
 - Huffman Encoding

Compression/Decompression "zlib" Library

The library includes the following new features:

- The library is a general purpose data compression library.
- Read and write gzip (.gz) format with an interface similar to that of stdio
- Huffman Encoding algorithms

Known Issues:

Cryptographic Library

- Nothing to report in this release.

Compression/Decompression "zlib" Library

- Nothing to report in this release

5.3.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO

YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.3.4 Using the Library

This topic describes the basic architecture of the Cryptographic (Crypto) Library and provides information and examples on how to use it.

Interface Header File: [crypto.h](#)

The interface to the Cryptographic (Crypto) Library is defined in the "crypto.h.h" header file. This file is included by the "crypto.h.h" file. Any C language source (.c) file that uses the Cryptographic (Crypto) Library should include "crypto.h.h".

Library File:

The Cryptographic (Crypto) Library archive (.a) file installed with MPLAB Harmony.

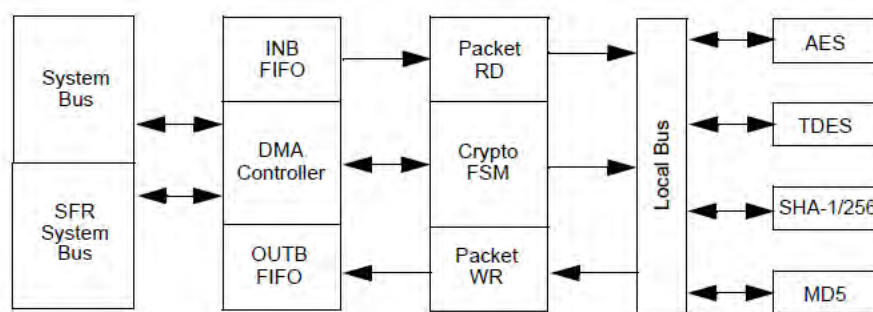
Please refer to the [MPLAB Harmony Overview](#) for how the Cryptographic (Crypto) Library interacts with the framework.

5.3.4.1 Abstraction Model

This library provides the low-level abstraction of the Cryptographic Library module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Cryptographic (Crypto) Software Abstraction Block Diagram



5.3.4.2 Library Overview

The [Library Interface](#) routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Cryptographic (Crypto) Library module.

5.3.5 Configuring the Library

The configuration of the Cryptographic Library is based on the file `sys_config.h`.

This header file contains the configuration selection for the Cryptographic Library. Based on the selections made, the Cryptographic Library will support or not support selected features. These configuration settings will apply to all instances of the Cryptographic Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.3.6 Building the Library

This section lists the files that are available in the `\src` of the Cryptographic Library driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.3.7 Library Interface

Data Types and Constants

	Name	Description
	CRYPT_AES_CTX	AES
	CRYPT_ECC_CTX	ECC
	CRYPT_HMAC_CTX	HMAC
	CRYPT_MD5_CTX	MD5
	CRYPT_RNG_CTX	RNG
	CRYPT_RSA_CTX	RSA
	CRYPT_SHA_CTX	SHA
	CRYPT_SHA256_CTX	SHA-256
	CRYPT_SHA384_CTX	SHA-384
	CRYPT_SHA512_CTX	SHA-512
	CRYPT_TDES_CTX	TDES
	MC_CRYPT_API_H	Defines Microchip CRYPTO API layer

AES Encryption/Decryption Functions

	Name	Description
	CRYPT_AES_CBC_Decrypt	
	CRYPT_AES_CBC_Encrypt	
	CRYPT_AES_CTR_Encrypt	
	CRYPT_AES_DIRECT_Decrypt	Direct encryption of one block of data.
	CRYPT_AES_DIRECT_Encrypt	Direct encryption of one block of data.
	CRYPT_AES_IvSet	

	CRYPT_AES_KeySet	

Compression Functions

	Name	Description
	CRYPT_HUFFMAN_Compress	Compress a block of data.
	CRYPT_HUFFMAN_DeCompress	Decompress a block of data.

ECC Encryption/Decryption Functions

	Name	Description
	CRYPT_ECC_DHE_KeyMake	
	CRYPT_ECC_DHE_SharedSecretMake	
	CRYPT_ECC_DSA_HashSign	
	CRYPT_ECC_DSA_HashVerify	
	CRYPT_ECC_Free	
	CRYPT_ECC_Initialize	
	CRYPT_ECC_KeySizeGet	
	CRYPT_ECC_PrivateImport	
	CRYPT_ECC_PublicExport	
	CRYPT_ECC_PublicImport	
	CRYPT_ECC_SignatureSizeGet	

General Functions

	Name	Description
	CRYPT_ERROR_StringGet	

HMAC Hash Functions

	Name	Description
	CRYPT_HMAC_DataAdd	Add data to the HMAC calculation.
	CRYPT_HMAC_Finalize	Complete the HMAC calculation and get the results.
	CRYPT_HMAC_SetKey	Initialize the HMAC context and set the key for the hash.

MD5 Functions

	Name	Description
	CRYPT_MD5_DataAdd	Updates the hash with the data provided.
	CRYPT_MD5_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_MD5_Initialize	Initializes the internal structures necessary for MD5 hash calculations.

Random Number Generator Functions

	Name	Description
	CRYPT_RNG_BlockGenerate	Create several random numbers.
	CRYPT_RNG_Get	Get one random number.
	CRYPT_RNG_Initialize	Initialize random number generator.

RSA Encryption/Decryption Functions

	Name	Description
	CRYPT_RSA_EncryptSizeGet	
	CRYPT_RSA_Free	
	CRYPT_RSA_Initialize	

	CRYPT_RSA_PrivateDecrypt	
	CRYPT_RSA_PrivateKeyDecode	
	CRYPT_RSA_PublicEncrypt	
	CRYPT_RSA_PublicKeyDecode	

SHA Hash functions

	Name	Description
	CRYPT_SHA_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA_Initialize	Initializes the internal structures necessary for SHA hash calculations.

SHA256 Hash Functions

	Name	Description
	CRYPT_SHA256_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA256_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA256_Initialize	Initializes the internal structures necessary for SHA256 hash calculations.

SHA384 Hash Functions

	Name	Description
	CRYPT_SHA384_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA384_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA384_Initialize	Initializes the internal structures necessary for SHA384 hash calculations.

SHA512 Hash Functions

	Name	Description
	CRYPT_SHA512_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA512_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA512_Initialize	Initializes the internal structures necessary for SHA512 hash calculations.

Triple DES (3DES) Encryption/Decryption Functions

	Name	Description
	CRYPT_TDES_CBC_Decrypt	Decrypt a data block using Triple DES.
	CRYPT_TDES_CBC_Encrypt	Encrypt a data block using Triple DES.
	CRYPT_TDES_IvSet	Set the Initialization Vector (IV) for a Triple DES operation.
	CRYPT_TDES_KeySet	Initialization of Triple DES context.

Description

This section describes the Application Programming Interface (API) functions of the Cryptographic Library.

Refer to each section for a detailed description.

5.3.7.1 General Functions

5.3.7.1.1 CRYPT_ERROR_StringGet Function

C

```
int CRYPT_ERROR_StringGet(
```

```
    int,  
    char*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Remarks

String needs to be >= 80 chars.

Example

5.3.7.2 SHA512 Hash Functions

5.3.7.2.1 CRYPT_SHA512_DataAdd Function

C

```
int CRYPT_SHA512_DataAdd(  
    CRYPT_SHA512_CTX*,  
    const unsigned char*,  
    unsigned int  
);
```

Description

This function updates the hash with the data provided.

Preconditions

The SHA512 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.
input	Pointer to the data to use to update the hash.
sz	Size of the data (in bytes) of the data to use to update the hash.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha512 or input.
- 0 - Otherwise.

Remarks

In order to preserve the validity of the SHA512 hash, nothing must modify the context holding variable between calls to CRYPT_SHA512_DataAdd.

Example

```
CRYPT_SHA512_CTX sha512;
```

```
uint8_t buffer[1024];
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

5.3.7.2.2 CRYPT_SHA512_Finalize Function

C

```
int CRYPT_SHA512_Finalize(
    CRYPT_SHA512_CTX*,
    unsigned char*
);
```

Description

This function finalizes the hash and puts the result into digest.

Preconditions

The SHA512 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha512 or digest.
- 0 - Otherwise.

Remarks

In order to preserve the validity of the SHA512 hash, nothing must modify the context holding variable between calls to [CRYPT_SHA512_DataAdd](#) and [CRYPT_SHA512_Finalize](#).

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t buffer[1024];
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

5.3.7.2.3 CRYPT_SHA512_Initialize Function

C

```
int CRYPT_SHA512_Initialize(
    CRYPT_SHA512_CTX*
);
```

Description

This function initializes the internal structures necessary for SHA512 hash calculations.

Preconditions

None.

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX strucure which holds the hash values.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - Otherwise.

Remarks

All SHA512 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

5.3.7.3 HMAC Hash Functions

5.3.7.3.1 CRYPT_HMAC_DataAdd Function

C

```
int CRYPT_HMAC_DataAdd(
    CRYPT_HMAC_CTX*,
    const unsigned char*,
    unsigned int
);
```

Description

This function adds data to the HMAC so that multiple blocks of data can be processed.

Preconditions

The [CRYPT_HMAC_CTX](#) context must be initialized using the [CRYPT_HMAC_SetKey](#) function prior to any call to this function.

Parameters

Parameters	Description
hmac	Pointer to context which saves state between calls.
input	Pointer to the data to use to update the hash.
sz	Size of the input data in bytes.

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Remarks**Example**

```
CRYPT_HMAC_CTX mCHmac;
byte          mcDigest[CRYPT_SHA512_DIGEST_SIZE];
```

```
CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);  
CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);  
CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

5.3.7.3.2 CRYPT_HMAC_Finalize Function

C

```
int CRYPT_HMAC_Finalize(  
    CRYPT_HMAC_CTX*,  
    unsigned char*  
);
```

Description

This function completes the HMAC calculations. The results are placed in the location pointed to by digest.

Preconditions

The [CRYPT_HMAC_CTX](#) context must be initialized using the [CRYPT_HMAC_SetKey](#) function prior to any call to this function.

Parameters

Parameters	Description
hmac	Pointer to context which saves state between calls.
digest	Pointer to place to put the final HMAC digest results.

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Remarks

The area pointed to by digest must be large enough to hold the results.

Example

```
CRYPT_HMAC_CTX mcHmac;  
byte mcDigest[CRYPT_SHA512_DIGEST_SIZE];  
  
CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);  
CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);  
CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

5.3.7.3.3 CRYPT_HMAC_SetKey Function

C

```
int CRYPT_HMAC_SetKey(  
    CRYPT_HMAC_CTX*,  
    int,  
    const unsigned char*,  
    unsigned int  
);
```

Description

This function initializes the HMAC context and set the key for the hash.

Preconditions

None.

Parameters

Parameters	Description
hmac	Pointer to context which saves state between calls.

5.3.7.4 SHA384 Hash Functions

5.3.7.4.1 CRYPT_SHA384_DataAdd Function

C

```
int CRYPT_SHA384_DataAdd(  
    CRYPT_SHA384_CTX*,  
    const unsigned char*,  
    unsigned int  
);
```

Description

This function updates the hash with the data provided.

Preconditions

The SHA384 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.
input	Pointer to the data to use to update the hash.
sz	Size of the data (in bytes) of the data to use to update the hash.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha384 or input.
- 0 - Otherwise.

Remarks

In order to preserve the validity of the SHA384 hash, nothing must modify the context holding variable between calls to CRYPT_SHA384_DataAdd.

Example

```
CRYPT_SHA384_CTX sha384;  
uint8_t buffer[1024];  
uint8_t shaSum[SHA384_DIGEST_SIZE];  
  
CRYPT_SHA384_Initialize(&sha384);  
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));  
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

5.3.7.4.2 CRYPT_SHA384_Finalize Function

C

```
int CRYPT_SHA384_Finalize(  
    CRYPT_SHA384_CTX*,  
    unsigned char*  
);
```

Description

This function finalizes the hash and puts the result into digest.

Preconditions

The SHA384 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX strucure which holds the hash values.
digest	Pointer to byte array to store hash result.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha384 or digest.
- 0 - Otherwise.

Remarks

In order to preserve the validity of the SHA384 hash, nothing must modify the context holding variable between calls to [CRYPT_SHA384_DataAdd](#) and [CRYPT_SHA384_Finalize](#).

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t buffer[1024];
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

5.3.7.4.3 CRYPT_SHA384_Initialize Function

C

```
int CRYPT_SHA384_Initialize(
    CRYPT_SHA384_CTX*
```

```
);
```

Description

This function initializes the internal structures necessary for SHA384 hash calculations.

Preconditions

None.

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX strucure which holds the hash values.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - Otherwise.

Remarks

All SHA384 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

5.3.7.5 SHA256 Hash Functions

5.3.7.5.1 CRYPT_SHA256_DataAdd Function

C

```
int CRYPT_SHA256_DataAdd(
    CRYPT_SHA256_CTX*,
    const unsigned char*,
    unsigned int
);
```

Description

This function updates the hash with the data provided.

Preconditions

The SHA256 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX strucure which holds the hash values.
input	Pointer to the data to use to update the hash.
sz	Size of the data (in bytes) of the data to use to update the hash.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha256 or input.
- 0 - Otherwise.

Remarks

In order to preserve the validity of the SHA256 hash, nothing must modify the context holding variable between calls to CRYPT_SHA256_DataAdd.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha256);
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha256, shaSum);
```

5.3.7.5.2 CRYPT_SHA256_Finalize Function

C

```
int CRYPT_SHA256_Finalize(
```

```
    CRYPT_SHA256_CTX*,  
    unsigned char*  
);
```

Description

This function finalizes the hash and puts the result into digest.

Preconditions

The SHA256 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or digest.
- 0 - Otherwise.

Remarks

In order to preserve the validity of the SHA256 hash, nothing must modify the context holding variable between calls to [CRYPT_SHA256_DataAdd](#) and [CRYPT_SHA256_Finalize](#).

Example

```
CRYPT_SHA256_CTX sha256;  
uint8_t buffer[1024];  
uint8_t shaSum[SHA256_DIGEST_SIZE];  
  
CRYPT_SHA256_Initialize(&sha256);  
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));  
CRYPT_SHA256_Finalize(&sha256, shaSum);
```

5.3.7.5.3 CRYPT_SHA256_Initialize Function

C

```
int CRYPT_SHA256_Initialize(  
    CRYPT_SHA256_CTX*  
);
```

Description

This function initializes the internal structures necessary for SHA256 hash calculations.

Preconditions

None.

Parameters

Parameters	Description
sha256	Pointer to context which saves state between calls.

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the

structure.

Example

```
CRYPT_SHA256_CTX sha;
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha);
CRYPT_SHA256_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha, shaSum);
```

5.3.7.6 SHA Hash functions

5.3.7.6.1 CRYPT_SHA_DataAdd Function

C

```
int CRYPT_SHA_DataAdd(
    CRYPT_SHA_CTX*,
    const unsigned char*,
    unsigned int
);
```

Description

This function updates the hash with the data provided.

Preconditions

The SHA context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX strucure which holds the hash values.
input	Pointer to the data to use to update the hash.
sz	Size of the data (in bytes) of the data to use to update the hash.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or input.
- 0 - Otherwise.

Remarks

In order to preserve the validity of the SHA hash, nothing must modify the context holding variable between calls to CRYPT_SHA_DataAdd.

Example

```
CRYPT_SHA_CTX sha;
uint8_t buffer[1024];
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shaSum);
```

5.3.7.6.2 CRYPT_SHA_Finalize Function

C

```
int CRYPT_SHA_Finalize(  
    CRYPT_SHA_CTX*,  
    unsigned char*  
);
```

Description

This function finalizes the hash and puts the result into digest.

Preconditions

The SHA context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Returns

BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or digest. 0 - Otherwise.

Remarks

In order to preserve the validity of the SHA hash, nothing must modify the context holding variable between calls to [CRYPT_SHA_DataAdd](#) and [CRYPT_SHA_Finalize](#).

Example

```
CRYPT_SHA_CTX sha;  
uint8_t buffer[1024];  
uint8_t shaSum[SHA_DIGEST_SIZE];  
  
CRYPT_SHA_Initialize(&sha);  
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));  
CRYPT_SHA_Finalize(&sha, shaSum);
```

5.3.7.6.3 CRYPT_SHA_Initialize Function

C

```
int CRYPT_SHA_Initialize(  
    CRYPT_SHA_CTX*  
);
```

Description

This function initializes the internal structures necessary for SHA hash calculations.

Preconditions

None.

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - Otherwise.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Example

```
CRYPT_SHA_CTX sha;
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shaSum);
```

5.3.7.7 Random Number Generator Functions

5.3.7.7.1 CRYPT_RNG_BlockGenerate Function

C

```
int CRYPT_RNG_BlockGenerate(
    CRYPT_RNG_CTX*,
    unsigned char*,
    unsigned int
);
```

Description

This function generates several random numbers and places them in the space allocated.

Preconditions

RNG context was initialized using the [CRYPT_RNG_Initialize](#) function.

Parameters

Parameters	Description
rng	Pointer to context which saves state between calls.
b	Pointer to buffer to store the random numbers.
sz	Number of 8-bit random numbers to generate.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - Otherwise.

Example

```
#define RANDOM_BYTE_SZ 32

int ret;
CRYPT_RNG_CTX mcRng;
byte out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);
```

```
ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

5.3.7.7.2 CRYPT_RNG_Get Function

C

```
int CRYPT_RNG_Get(  
    CRYPT_RNG_CTX*,  
    unsigned char*  
);
```

Description

This function gets one random number from the random number generator.

Preconditions

RNG context was initialized using the [CRYPT_RNG_Initialize](#) function.

Parameters

Parameters	Description
rng	Pointer to context which saves state between calls.
b	Pointer to 8-bit location to store the result.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- <0 - An error occurred.
- 0 or positive - Success.

Example

```
#define RANDOM_BYTE_SZ 32  
  
int      ret;  
CRYPT_RNG_CTX mcRng;  
byte      out[RANDOM_BYTE_SZ];  
  
ret = CRYPT_RNG_Initialize(&mcRng);  
  
ret = CRYPT_RNG_Get(&mcRng, &out[0]);  
  
ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

5.3.7.7.3 CRYPT_RNG_Initialize Function

C

```
int CRYPT_RNG_Initialize(  
    CRYPT_RNG_CTX*  
);
```

Description

This function initializes the context that stores information relative to random number generation.

Preconditions

None.

Parameters

Parameters	Description
rng	Pointer to random number generator context.

Returns

- negative - An error occurred setting up the random number generator.
- 0 - Otherwise.

Example

```
#define RANDOM_BYTE_SZ 32

int          ret;
CRYPT_RNG_CTX mcRng;
byte         out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

5.3.7.8 Compression Functions

5.3.7.8.1 CRYPT_HUFFMAN_Compress Function

C

```
int CRYPT_HUFFMAN_Compress(
    unsigned char*,
    unsigned int,
    const unsigned char*,
    unsigned int,
    unsigned int
);
```

Description

This function compresses a block of data using Huffman encoding.

Preconditions

None.

Parameters

Parameters	Description
out	Pointer to location to store the compressed data.
outSz	Maximum size of the output data in bytes.
in	Point to location of source data.
inSz	Size of the input data in bytes.
flags	Flags to control how compress operates

Returns

- negative - error code
- positive - bytes stored in out buffer

Remarks

Output buffer must be large enough to hold the contents of the operation.

Example

```
const unsigned char text[] = "...";
```

```
unsigned int inSz = sizeof(text);
unsigned int outSz;
unsigned char cBuffer[1024];

int ret;

ret = CRYPT_HUFFMAN_COMPRESS(cBuffer, sizeof(cBuffer), text, inSz, 0);
```

5.3.7.8.2 CRYPT_HUFFMAN_DeCompress Function

C

```
int CRYPT_HUFFMAN_DeCompress(
    unsigned char*,
    unsigned int,
    const unsigned char*,
    unsigned int
);
```

Description

This function decompresses a block of data using Huffman encoding.

Preconditions

None.

Parameters

Parameters	Description
out	Pointer to destination buffer
outSz	Size of destination buffer
in	Pointer to source buffer to decompress
inSz	Size of source buffer to decompress

Returns

- negative - Error code
- positive - Bytes stored in out buffer

Remarks

Output buffer must be large enough to hold the contents of the operation.

Example

```
unsigned char cBuffer[1024];
unsigned char dBuffer[1024];

int ret

ret = CRYPT_HUFFMAN_DeCompress(dBuffer, sizeof(dBuffer), cBuffer, msglen);
```

5.3.7.9 ECC Encryption/Decryption Functions

5.3.7.9.1 CRYPT_ECC_DHE_KeyMake Function

C

```
int CRYPT_ECC_DHE_KeyMake(
    CRYPT_ECC_CTX*,
    CRYPT_RNG_CTX*,
```

```
    int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.2 CRYPT_ECC_DHE_SharedSecretMake Function

C

```
int CRYPT_ECC_DHE_SharedSecretMake(  
    CRYPT_ECC_CTX*,  
    CRYPT_ECC_CTX*,  
    unsigned char*,  
    unsigned int,  
    unsigned int*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.3 CRYPT_ECC_DSA_HashSign Function

C

```
int CRYPT_ECC_DSA_HashSign(  
    CRYPT_ECC_CTX*,  
    CRYPT_RNG_CTX*,  
    unsigned char*,  
    unsigned int,  
    unsigned int*,  
    const unsigned char*,  
    unsigned int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.4 CRYPT_ECC_DSA_HashVerify Function

C

```
int CRYPT_ECC_DSA_HashVerify(  
    CRYPT_ECC_CTX*,  
    const unsigned char*,  
    unsigned int,  
    unsigned char*,  
    unsigned int,  
    int*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.5 CRYPT_ECC_Free Function

C

```
int CRYPT_ECC_Free(  
    CRYPT_ECC_CTX*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.6 CRYPT_ECC_Initialize Function

C

```
int CRYPT_ECC_Initialize(  
    CRYPT_ECC_CTX*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.7 CRYPT_ECC_KeySizeGet Function

C

```
int CRYPT_ECC_KeySizeGet(  
    CRYPT_ECC_CTX*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.8 CRYPT_ECC_PrivateImport Function

C

```
int CRYPT_ECC_PrivateImport(  
    CRYPT_ECC_CTX*,  
    const unsigned char*,  
    unsigned int,  
    const unsigned char*,  
    unsigned int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.9 CRYPT_ECC_PublicExport Function

C

```
int CRYPT_ECC_PublicExport(  
    CRYPT_ECC_CTX*,  
    unsigned char*,  
    unsigned int,  
    unsigned int*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.10 CRYPT_ECC_PublicImport Function

C

```
int CRYPT_ECC_PublicImport(  
    CRYPT_ECC_CTX*,  
    const unsigned char*,  
    unsigned int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.9.11 CRYPT_ECC_SignatureSizeGet Function

C

```
int CRYPT_ECC_SignatureSizeGet(  
    CRYPT_ECC_CTX*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -

Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.10 Triple DES (3DES) Encryption/Decryption Functions

5.3.7.10.1 CRYPT_TDES_CBC_Decrypt Function

C

```
int CRYPT_TDES_CBC_Decrypt(  
    CRYPT_TDES_CTX*,  
    unsigned char*,  
    const unsigned char*,  
    unsigned int  
);
```

Description

This function decrypts a block of data using a Triple DES algorithm.

Preconditions

The context tdes must be set earlier using [CRYPT_TDES_KeySet](#). The input block must be a multiple of 8 bytes long.

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
out	Pointer to output buffer to store the results.
in	Pointer to input buffer for the source of the data.
inSz	Size of the input data buffer.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - Otherwise.

Remarks

Input data must have a length a multiple of 8 bytes. Output data will be zero-padded at the end if the original data was not a multiple of 8 bytes long.

Example

```
CRYPT_TDES_CTX mcDes3;  
int ret;  
byte out1[TDES_SIZE];  
byte out2[TDES_SIZE];  
  
strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 24);  
strncpy((char*)iv, "12345678", 8);  
  
ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);  
  
ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);  
  
ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);
```

```
ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

5.3.7.10.2 CRYPT_TDES_CBC_Encrypt Function

C

```
int CRYPT_TDES_CBC_Encrypt(
    CRYPT_TDES_CTX*,
    unsigned char*,
    const unsigned char*,
    unsigned int
);
```

Description

This function encrypts a block of data using a Triple DES algorithm.

Preconditions

The context tdes must be set earlier using [CRYPT_TDES_KeySet](#). The input block must be a multiple of 8 bytes long.

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
out	Pointer to output buffer to store the results.
in	Pointer to input buffer for the source of the data.
inSz	Size of the input data buffer.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - Otherwise.

Remarks

The input data must be padded at the end with zeros to make the length a multiple of 8.

Example

```
CRYPT_TDES_CTX mcDes3;
int ret;
byte out1[TDES_SIZE];
byte out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 24);
strncpy((char*)iv, "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);
ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);
ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);
ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

5.3.7.10.3 CRYPT_TDES_IvSet Function

C

```
int CRYPT_TDES_IvSet(
    CRYPT_TDES_CTX*,
    const unsigned char*
);
```


Description

This function changes the IV of a TDES context, but leaves the Key alone.

Preconditions

None.

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
iv	Pointer to buffer holding the initialization vector. Must be 8 bytes in size.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - Otherwise.

Remarks

The IV must be 8 bytes long.

Example

```
CRYPT_TDES_CTX mcDes3;
int          ret;
byte         out1[TDES_SIZE];
byte         out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 24);
strncpy((char*)iv, "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);
ret = CRYPT_TDES_IvSet(&mcDes3, iv);
ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);
ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);
ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

5.3.7.10.4 CRYPT_TDES_KeySet Function

C

```
int CRYPT_TDES_KeySet(
    CRYPT_TDES_CTX*,
    const unsigned char*,
    const unsigned char*,
    int
);
```

Description

This function sets the key and initialization vector (IV) for a set of Triple-DES operations.

Preconditions

None.

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.

key	Pointer to buffer holding the key. Must be 24 bytes in size.
iv	Pointer to buffer holding the initialization vector. Must be 8 bytes in size.

5.3.7.11 RSA Encryption/Decryption Functions

5.3.7.11.1 CRYPT_RSA_EncryptSizeGet Function

C

```
int CRYPT_RSA_EncryptSizeGet(  
    CRYPT_RSA_CTX*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.11.2 CRYPT_RSA_Free Function

C

```
int CRYPT_RSA_Free(  
    CRYPT_RSA_CTX*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.11.3 CRYPT_RSA_Initialize Function

C

```
int CRYPT_RSA_Initialize(  
    CRYPT_RSA_CTX*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example**5.3.7.11.4 CRYPT_RSA_PrivateDecrypt Function****C**

```
int CRYPT_RSA_PrivateDecrypt(  
    CRYPT_RSA_CTX*,  
    unsigned char*,  
    unsigned int,  
    const unsigned char*,  
    unsigned int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example**5.3.7.11.5 CRYPT_RSA_PrivateKeyDecode Function****C**

```
int CRYPT_RSA_PrivateKeyDecode(  
    CRYPT_RSA_CTX*,  
    const unsigned char*,  
    unsigned int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.11.6 CRYPT_RSA_PublicEncrypt Function

C

```
int CRYPT_RSA_PublicEncrypt(  
    CRYPT_RSA_CTX*,  
    unsigned char*,  
    unsigned int,  
    const unsigned char*,  
    unsigned int,  
    CRYPT_RNG_CTX*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.11.7 CRYPT_RSA_PublicKeyDecode Function

C

```
int CRYPT_RSA_PublicKeyDecode(  
    CRYPT_RSA_CTX*,  
    const unsigned char*,  
    unsigned int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.12 AES Encryption/Decryption Functions

5.3.7.12.1 CRYPT_AES_CBC_Decrypt Function

C

```
int CRYPT_AES_CBC_Decrypt(  
    CRYPT_AES_CTX*,  
    unsigned char*,  
    const unsigned char*,  

```

```
    unsigned int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.12.2 CRYPT_AES_CBC_Encrypt Function

C

```
int CRYPT_AES_CBC_Encrypt(  
    CRYPT_AES_CTX*,  
    unsigned char*,  
    const unsigned char*,  
    unsigned int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.12.3 CRYPT_AES_CTR_Encrypt Function

C

```
int CRYPT_AES_CTR_Encrypt(  
    CRYPT_AES_CTX*,  
    unsigned char*,  
    const unsigned char*,  
    unsigned int  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Remarks

ctr (counter), use Encrypt both ways with ENCRYPT key setup

Example

5.3.7.12.4 CRYPT_AES_DIRECT_Decrypt Function

C

```
int CRYPT_AES_DIRECT_Decrypt(  
    CRYPT_AES_CTX*,  
    unsigned char*,  
    const unsigned char*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.12.5 CRYPT_AES_DIRECT_Encrypt Function

C

```
int CRYPT_AES_DIRECT_Encrypt(  
    CRYPT_AES_CTX*,  
    unsigned char*,  
    const unsigned char*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.12.6 CRYPT_AES_IvSet Function

C

```
int CRYPT_AES_IvSet(  
    CRYPT_AES_CTX*,  
    const unsigned char*  
);
```

Parameters

Parameters	Description
Param1	Pointer to context which saves state between calls.
Param2	Param3 -
Param4	Param5 -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.12.7 CRYPT_AES_KeySet Function

C

```
int CRYPT_AES_KeySet(  
    CRYPT_AES_CTX*,  
    const unsigned char*,  
    unsigned int,  
    const unsigned char*,  
    int  
);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
key	keyLen -
iv	dir -

Returns

*BAD_FUNC_ARG - An invalid pointer was passed to the function. *0 - Otherwise.

Example

5.3.7.13 MD5 Functions

5.3.7.13.1 CRYPT_MD5_DataAdd Function

C

```
int CRYPT_MD5_DataAdd(  
    CRYPT_MD5_CTX*,  
    const unsigned char*,  
    unsigned int  
);
```

Description

This function updates the hash with the data provided.

Preconditions

The MD5 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX strucure which holds the hash values.
input	Pointer to the data to use to update the hash.
sz	Size of the data (in bytes) of the data to use to update the hash.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in md5 or input.
- 0 - Otherwise.

Remarks

In order to preserve the validity of the MD5 hash, nothing must modify the context holding variable between calls to [CRYPT_MD5_DataAdd](#).

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

5.3.7.13.2 CRYPT_MD5_Finalize Function

C

```
int CRYPT_MD5_Finalize(
    CRYPT_MD5_CTX*,
    unsigned char*
);
```

Description

This function finalizes the hash and puts the result into digest.

Preconditions

The MD5 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX strucure which holds the hash values.
digest	Pointer to byte array to store hash result.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in md5 or digest.
- 0 - Otherwise.

Remarks

In order to preserve the validity of the MD5 hash, nothing must modify the context holding variable between calls to [CRYPT_MD5_DataAdd](#) and [CRYPT_MD5_Finalize](#).

Example

```
CRYPT_MD5_CTX md5;
```



```
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

5.3.7.13.3 CRYPT_MD5_Initialize Function

C

```
int CRYPT_MD5_Initialize(
    CRYPT_MD5_CTX* md5
);
```

Description

This function initializes the internal structures necessary for MD5 hash calculations.

Preconditions

None.

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - Otherwise.

Remarks

All MD5 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

5.3.7.14 Data Types and Constants

5.3.7.14.1 CRYPT_AES_CTX Structure

C

```
struct CRYPT_AES_CTX {
    int holder[69];
};
```

Description

AES

Members

Members	Description
int holder[69];	big enough to hold internal, but check on init

5.3.7.14.2 CRYPT_ECC_CTX Structure**C**

```
struct CRYPT_ECC_CTX {  
    void* holder;  
};
```

Description

ECC

5.3.7.14.3 CRYPT_HMAC_CTX Structure**C**

```
struct CRYPT_HMAC_CTX {  
    long long holder[67];  
};
```

Description

HMAC

Members

Members	Description
long long holder[67];	big enough to hold internal, but check on init

5.3.7.14.4 CRYPT_MD5_CTX Structure**C**

```
struct CRYPT_MD5_CTX {  
    int holder[24];  
};
```

Description

MD5

Members

Members	Description
int holder[24];	big enough to hold internal, but check on init

5.3.7.14.5 CRYPT_RNG_CTX Structure**C**

```
struct CRYPT_RNG_CTX {  
    int holder[66];  
};
```

Description

RNG

Members

Members	Description
int holder[66];	big enough to hold internal, but check on init

5.3.7.14.6 CRYPT_RSA_CTX Structure**C**

```
struct CRYPT_RSA_CTX {  
    void* holder;  
};
```

Description

RSA

5.3.7.14.7 CRYPT_SHA_CTX Structure**C**

```
struct CRYPT_SHA_CTX {  
    int holder[24];  
};
```

Description

SHA

Members

Members	Description
int holder[24];	big enough to hold internal, but check on init

5.3.7.14.8 CRYPT_SHA256_CTX Structure**C**

```
struct CRYPT_SHA256_CTX {  
    int holder[28];  
};
```

Description

SHA-256

Members

Members	Description
int holder[28];	big enough to hold internal, but check on init

5.3.7.14.9 CRYPT_SHA384_CTX Structure**C**

```
struct CRYPT_SHA384_CTX {  
    long long holder[32];  
};
```

Description

SHA-384

Members

Members	Description
long long holder[32];	big enough to hold internal, but check on init

5.3.7.14.10 CRYPT_SHA512_CTX Structure**C**

```
struct CRYPT_SHA512_CTX {  
    long long holder[36];  
};
```

Description

SHA-512

Members

Members	Description
long long holder[36];	big enough to hold internal, but check on init

5.3.7.14.11 CRYPT_TDES_CTX Structure**C**

```
struct CRYPT_TDES_CTX {  
    int holder[100];  
};
```

Description

TDES

Members

Members	Description
int holder[100];	big enough to hold internal, but check on init

5.3.7.14.12 MC_CRYPT_API_H Macro**C**

```
#define MC_CRYPT_API_H
```

Description

Defines Microchip CRYPTO API layer

5.3.8 Files**Files**

Name	Description
crypto.h	Crypto Framework Library header for cryptographic functions.

Description

5.3.8.1 crypto.h

Crypto Framework Library Header

This header file contains function prototypes and definitions of the data types and constants that make up the Cryptographic Framework Library for PIC32 families of Microchip microcontrollers.

Functions

	Name	Description
≡	CRYPT_AES_CBC_Decrypt	
≡	CRYPT_AES_CBC_Encrypt	
≡	CRYPT_AES_CTR_Encrypt	
≡	CRYPT_AES_DIRECT_Decrypt	Direct encryption of one block of data.
≡	CRYPT_AES_DIRECT_Encrypt	Direct encryption of one block of data.
≡	CRYPT_AES_IvSet	
≡	CRYPT_AES_KeySet	
≡	CRYPT_ECC_DHE_KeyMake	
≡	CRYPT_ECC_DHE_SharedSecretMake	
≡	CRYPT_ECC_DSA_HashSign	
≡	CRYPT_ECC_DSA_HashVerify	
≡	CRYPT_ECC_Free	
≡	CRYPT_ECC_Initialize	
≡	CRYPT_ECC_KeySizeGet	
≡	CRYPT_ECC_PrivateImport	
≡	CRYPT_ECC_PublicExport	
≡	CRYPT_ECC_PublicImport	
≡	CRYPT_ECC_SignatureSizeGet	
≡	CRYPT_ERROR_StringGet	
≡	CRYPT_HMAC_DataAdd	Add data to the HMAC calculation.
≡	CRYPT_HMAC_Finalize	Complete the HMAC calculation and get the results.
≡	CRYPT_HMAC_SetKey	Initialize the HMAC context and set the key for the hash.
≡	CRYPT_HUFFMAN_Compress	Compress a block of data.
≡	CRYPT_HUFFMAN_DeCompress	Decompress a block of data.
≡	CRYPT_MD5_DataAdd	Updates the hash with the data provided.
≡	CRYPT_MD5_Finalize	Finalizes the hash and puts the result into digest.
≡	CRYPT_MD5_Initialize	Initializes the internal structures necessary for MD5 hash calculations.
≡	CRYPT_RNG_BlockGenerate	Create several random numbers.
≡	CRYPT_RNG_Get	Get one random number.
≡	CRYPT_RNG_Initialize	Initialize random number generator.
≡	CRYPT_RSA_EncryptSizeGet	
≡	CRYPT_RSA_Free	
≡	CRYPT_RSA_Initialize	
≡	CRYPT_RSA_PrivateDecrypt	
≡	CRYPT_RSA_PrivateKeyDecode	
≡	CRYPT_RSA_PublicEncrypt	

	CRYPT_RSA_PublicKeyDecode	
	CRYPT_SHA_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA_Initialize	Initializes the internal structures necessary for SHA hash calculations.
	CRYPT_SHA256_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA256_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA256_Initialize	Initializes the internal structures necessary for SHA256 hash calculations.
	CRYPT_SHA384_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA384_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA384_Initialize	Initializes the internal structures necessary for SHA384 hash calculations.
	CRYPT_SHA512_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA512_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA512_Initialize	Initializes the internal structures necessary for SHA512 hash calculations.
	CRYPT_TDES_CBC_Decrypt	Decrypt a data block using Triple DES.
	CRYPT_TDES_CBC_Encrypt	Encrypt a data block using Triple DES.
	CRYPT_TDES_IvSet	Set the Initialization Vector (IV) for a Triple DES operation.
	CRYPT_TDES_KeySet	Initialization of Triple DES context.

Macros

	Name	Description
	MC_CRYPT_API_H	Defines Microchip CRYPTO API layer

Structures

	Name	Description
	CRYPT_AES_CTX	AES
	CRYPT_ECC_CTX	ECC
	CRYPT_HMAC_CTX	HMAC
	CRYPT_MD5_CTX	MD5
	CRYPT_RNG_CTX	RNG
	CRYPT_RSA_CTX	RSA
	CRYPT_SHA_CTX	SHA
	CRYPT_SHA256_CTX	SHA-256
	CRYPT_SHA384_CTX	SHA-384
	CRYPT_SHA512_CTX	SHA-512
	CRYPT_TDES_CTX	TDES

Company

Microchip Technology Inc.

5.4 Math Library Help

5.4.1 DSP Fixed-Point Library

5.4.1.1 Introduction

DSP Fixed-Point Library

for

Microchip Microcontrollers

The DSP Fixed-Point Library is available for the PIC32MZ family of microcontrollers. This library was created from optimized assembly routines written specifically for the microAptiv™ core.

Description

The DSP Fixed-Point Library contains building block functions for developing digital signal processing algorithms. The library supports the Q15 and Q31 fractional data formats. The functions are implemented in efficient assembly specifically targeted at the DSP extensions in this core family. The library makes these functions available in a simple C-callable structure.

Functions included in the DSP Fixed-Point Library include complex math, vector math, matrix math, digital filters, and transforms. In many cases, these functions require specific data structures to operate, which are detailed in the header file and examples.

5.4.1.2 Release Notes

MPLAB Harmony Version: v0.70b **DSP Fixed-Point Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

- 98 functions:
 - Vector math
 - Complex math
 - Transforms
 - Digital Filters
 - Matrix math
- 16-bit and 32-bit fixed-point integer functions
- Optimized in high performance assembly

Known Issues:

- Optimized for PIC32MZ microAptiv™ core only
- Will not function with `_Fract` data type

5.4.1.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.4.1.4 Using the Library

This topic describes the basic architecture of the DSP Fixed-Point Library and provides information and examples on how to use it.

Interface Header File: [dsp.h](#), [libq.h](#)

The interface to the DSP Fixed-Point library is defined in the "[dsp.h](#)" header file. Any C language source (.c) file that uses the DSP Fixed-Point library must include "[dsp.h](#)".

Some functions make special use of the optimized fixed-point math library [libq.h](#). For use of those functions, the [libq.h](#) file must also be included in a project. The [libq.h](#) file is also installed with MPLAB Harmony. Specific notes within each function will describe if the function is dependent on the libq library.

Library File: [dsp.a](#), [libq.a](#)

The DSP Fixed-Point library archive (.a) file installed with MPLAB Harmony.

This library is only available in binary form, with prototypes for each function described in the [dsp.h](#) file.

In the case of functions that are supported by libq, the fixed-point math function library, the [libq.a](#) library must also be installed. This library is also available in binary form and is installed with MPLAB Harmony.

5.4.1.4.1 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the DSP Fixed-Point Library.

Library Interface Section	Description
Complex Math Functions	General mathematical operations using a complex structure with the form (a + bi)
Vector Math Functions	Mathematical operations on a array of numbers or vector
Matrix Math Functions	Mathematical operations on a matrix
Digital Filtering Functions	FIR and IIR filtering functions with various architectures
Digital Transform Functions	FFT, Windows and related transform elements
Support Functions	Quick support functions for numerical transform

The DSP Fixed-Point Library uses fixed-point fractional functions to optimize execution speed. These functions limit the accuracy of the calculations to the bits specified for the function. Due to parallelism in some operations, the 16-bit version of the functions are more efficient than their 32-bit counterparts. In many cases both 16-bit and 32-bit functions are available to give the user the choice of balance between speed and functional resolution.

Fractional representation of a real number is given by:

$Qn.m$

where:

- n is the number of data bits to the left of the radix point
- m is the number of data bits to the right of the radix point
- a signed bit is implied, and takes one bit of resolution
- Shorthand may eliminate the leading 0, such as in Q0.15, which may be shortened to Q15, and similarly Q0.31, which is shortened to Q31

$Qn.m$ numerical values are used by the library processing data as integers. In this format the n represents the number of integer bits, and the m represents the number of fractional bits. All values assume a sign bit in the most significant bit. Therefore, the range of the numerical value is:

$$-2^{(n-1)} \text{ to } [2^{(n-1)} - 2^{(-m)}]; \text{ with a resolution of } 2^{(-m)}.$$

A Q16 format number (Q15.16) would range from -32768.0 (0x8000 0000) to 32767.99998474 with a precision of 0.000015259 (or 2^{-16}).

For example, a numerical representation of the number 3.14159 in Q2.13 notation would be:

$$3.14159 * 2^{13} = 25735.9 \Rightarrow 0x6488$$

And converting from the Q7.8 format with the value 0x1D89 would be:

$$0x1D89 / 2^8 = 7561 / 256 \Rightarrow 29.5316, \text{ accurate to } 0.00391$$

A majority of the DSP Fixed-Point Library uses functions with variables in Q15 or Q31 format. Representations of these numbers are given in the Types section of this document, and generally are `int16_t` (for Q15 fractional representation) and `int32_t` (for Q31 fractional representation). This limits the equivalent numerical range to roughly -1.0 to 0.999999999. It is possible to represent other number ranges, but scaling before and after the function call are necessary.

All library functions will saturate the output if the value exceeds the maximum or is lower than the minimum allowable value for that resolution. Some pre-scaling may be necessary to prevent unwanted saturation in functions that may otherwise create calculation errors.

5.4.1.4.2 Table of Library Functions

Family	Function	Definition
Complex Math	Addition	DSP_ComplexAdd32
	Conjugate	DSP_ComplexConj32
	Dot Product	DSP_ComplexDotProd32
	Multiplication	DSP_ComplexMult32
	Scalar Multiplication	DSP_ComplexScalarMult32
	Subtraction	DSP_ComplexSub32
Digital Filter	FIR	DSP_FilterFIR32
	FIR Decimation	DSP_FilterFIRDecim32
	FIR Interpolation	DSP_FilterFIRInterp32
	FIR LMS	DSP_FilterLMS16
	IIR	DSP_FilterIIR16 ; DSP_FilterIIRSetup16
	IIR Biquad	DSP_FilterIIRBQ16 ; DSP_FilterIIRBQ16_fast ; DSP_FilterIIRBQ32
	IIR Biquad Cascade	DSP_FilterIIRBQ16_cascade8 ; DSP_FilterIIRBQ16_cascade8_fast
	IIR Biquad Parallel	DSP_FilterIIRBQ16_parallel8 ; DSP_FilterIIRBQ16_parallel8_fast
Matrix Math	Addition	DSP_MatrixAdd32
	Equality	DSP_MatrixEqual32
	Initialization	DSP_MatrixInit32
	Multiplication	DSP_MatrixMul32
	Scale	DSP_MatrixScale32
	Subtraction	DSP_MatrixSub32
	Transpose	DSP_MatrixTranspose32
Transform	FFT	DSP_TransformFFT16 ; DSP_TransformFFT32
	Setup factors	DSP_TransformFFT16_setup ; DSP_TransformFFT32_setup
	inverse FFT	DSP_TransformIFFT16
	Windows	DSP_TransformWindow_Bart16 ; DSP_TransformWindow_Bart32 ; DSP_TransformWindow_Black16 ; DSP_TransformWindow_Black32 ; DSP_TransformWindow_Cosine16 ; DSP_TransformWindow_Cosine32 ; DSP_TransformWindow_Hamm16 ; DSP_TransformWindow_Hamm32 ; DSP_TransformWindow_Hann16 ; DSP_TransformWindow_Hann32 ; DSP_TransformWindow_Kaiser16 ; DSP_TransformWindow_Kaiser32 ;
	Window Initialization	DSP_TransformWinInit_Bart16 ; DSP_TransformWinInit_Bart32 ; DSP_TransformWinInit_Black16 ; DSP_TransformWinInit_Black32 ; DSP_TransformWinInit_Cosine16 ; DSP_TransformWinInit_Cosine32 ; DSP_TransformWinInit_Hamm16 ; DSP_TransformWinInit_Hamm32 ; DSP_TransformWinInit_Hann16 ; DSP_TransformWinInit_Hann32 ; DSP_TransformWinInit_Kaiser16 ; DSP_TransformWinInit_Kaiser32
Vector Math	Absolute value	DSP_VectorAbs16 ; DSP_VectorAbs32

	Addition	DSP_VectorAdd16 ; DSP_VectorAdd32
	Addition w/ constant	DSP_VectorAddc16 ; DSP_VectorAddc32
	Binary exponent	DSP_VectorBexp16 ; DSP_VectorBexp32
	Log, Exponent, Square root	DSP_VectorExp ; DSP_VectorLog10 ; DSP_VectorLog2 ; DSP_VectorLn ; DSP_VectorSqrt ;
	Multiplication & Division	DSP_VectorMul16 ; DSP_VectorMul32 ; DSP_VectorDotp16 ; DSP_VectorDotp32 ; DSP_VectorMulc16 ; DSP_VectorMulc32 ; DSP_VectorDivC ; DSP_VectorRecip
	Subtraction	DSP_VectorSub16 ; DSP_VectorSub32
	Power (sum of squares)	DSP_VectorSumSquares16 ; DSP_VectorSumSquares32
	Equality check	DSP_VectorChkEqu32
	Maximum	DSP_VectorMax32 ; DSP_VectorMaxIndex32
	Minimum	DSP_VectorMin32 ; DSP_VectorMinIndex32
	Copy & Fill	DSP_VectorCopy ; DSP_VectorFill ; DSP_VectorZeroPad
	Shift	DSP_VectorShift ; DSP_VectorCopyReverse32
	Negate	DSP_VectorNegate
	Statistics	DSP_VectorAutocorr16 ; DSP_VectorMean32 ; DSP_VectorRMS16 ; DSP_VectorStdDev16 ; DSP_VectorVari16 ; DSP_VectorVariance

5.4.1.5 Library Interface

Data Types and Constants

	Name	Description
	biquad16	Q15 biquad
	int16c	Q15 complex number (a + bi)
	int32c	Q31 complex number (a + bi)
	matrix32	Q31 matrix
	PARM_EQUAL_FILTER	IIR BQ filter structure Q15 data, Q31 storage
	PARM_EQUAL_FILTER_16	IIR BQ filter structure Q15
	PARM_EQUAL_FILTER_32	IIR BQ filter structure Q31
	PARM_FILTER_GAIN	filter gain structure
	_PARM_EQUAL_FILTER	IIR BQ filter structure Q15 data, Q31 storage
	_PARM_EQUAL_FILTER_16	IIR BQ filter structure Q15
	_PARM_EQUAL_FILTER_32	IIR BQ filter structure Q31
	MAX16	maximum Q15
	MAX32	maximum Q31
	MIN16	minimum Q15
	MIN32	minimum Q31

Complex Math Functions

	Name	Description
≡	DSP_ComplexAdd32	Calculates the sum of two complex numbers.
≡	DSP_ComplexConj16	Calculates the complex conjugate of a complex number.
≡	DSP_ComplexConj32	Calculates the complex conjugate of a complex number.
≡	DSP_ComplexDotProd32	Calculates the dot product of two complex numbers.
≡	DSP_ComplexMult32	Multiplies two complex numbers.
≡	DSP_ComplexScalarMult32	Multiplies a complex number and a scalar number.
≡	DSP_ComplexSub32	Calculates the difference of two complex numbers.

Digital Filter Functions

	Name	Description
≡	DSP_FilterFIR32	Performs a Finite Impulse Response filter on a vector.
≡	DSP_FilterFIRDecim32	Performs a decimating FIR filter on the input array.
≡	DSP_FilterFIRInterp32	Performs an interpolating FIR filter on the input array.
≡	DSP_FilterIIR16	Performs a single-sample cascaded biquad Infinite Impulse Response (IIR) filter.
≡	DSP_FilterIIRBQ16	Performs a single-pass IIR Biquad Filter.
≡	DSP_FilterIIRBQ16_cascade8	Performs a single IIR Biquad Filter as a cascade of 8 series filters.
≡	DSP_FilterIIRBQ16_cascade8_fast	Performs a single IIR Biquad Filter as a cascade of 8 series filters.
≡	DSP_FilterIIRBQ16_fast	Performs a single-pass IIR Biquad Filter.
≡	DSP_FilterIIRBQ16_parallel8	Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.
≡	DSP_FilterIIRBQ16_parallel8_fast	Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.
≡	DSP_FilterIIRBQ32	Performs a high resolution single-pass IIR Biquad Filter.
≡	DSP_FilterIIRSetup16	Converts biquad structure to coeffs array to set up IIR filter.
≡	DSP_FilterLMS16	Performs a single sample Least Mean Squares FIR Filter.

Matrix Math Functions

	Name	Description
≡	DSP_MatrixAdd32	Addition of two matrices $C = (A + B)$.
≡	DSP_MatrixEqual32	Equality of two matrices $C = (A)$.
≡	DSP_MatrixInit32	Initializes the first N elements of a Matrix to the value num.
≡	DSP_MatrixMul32	Multiplication of two matrices $C = A \times B$.
≡	DSP_MatrixScale32	Scales each element of an input buffer (matrix) by a fixed number.
≡	DSP_MatrixSub32	Subtraction of two matrices $C = (A - B)$.
≡	DSP_MatrixTranspose32	Transpose of a Matrix $C = A (T)$.

Support Functions

	Name	Description
≡	mul16	multiply and shift integer
≡	mul16r	multiply and shift Q15
≡	mul32	multiply and shift Q31
≡	SAT16	saturate both positive and negative Q15
≡	SAT16N	saturate negative Q15
≡	SAT16P	saturate positive Q15

Transform Functions

	Name	Description
	DSP_TransformFFT16	Creates an Fast Fourier Transform (FFT) from a time domain input.
	DSP_TransformFFT16_setup	Creates FFT coefficients for use in the FFT16 function.
	DSP_TransformFFT32	Creates an Fast Fourier Transform (FFT) from a time domain input.
	DSP_TransformFFT32_setup	Creates FFT coefficients for use in the FFT32 function.
	DSP_TransformIFFT16	Creates an Inverse Fast Fourier Transform (FFT) from a frequency domain input.
	DSP_TransformWindow_Bart16	Perform a Bartlett window on a vector.
	DSP_TransformWindow_Bart32	Perform a Bartlett window on a vector.
	DSP_TransformWindow_Black16	Perform a Blackman window on a vector.
	DSP_TransformWindow_Black32	Perform a Blackman window on a vector.
	DSP_TransformWindow_Cosine16	Perform a Cosine (Sine) window on a vector.
	DSP_TransformWindow_Cosine32	Perform a Cosine (Sine) window on a vector.
	DSP_TransformWindow_Hamm16	Perform a Hamming window on a vector.
	DSP_TransformWindow_Hamm32	Perform a Hamming window on a vector.
	DSP_TransformWindow_Hann16	Perform a Hanning window on a vector.
	DSP_TransformWindow_Hann32	Perform a Hanning window on a vector.
	DSP_TransformWindow_Kaiser16	Perform a Kaiser window on a vector.
	DSP_TransformWindow_Kaiser32	Perform a Kaiser window on a vector.
	DSP_TransformWinInit_Bart16	Create a Bartlett window.
	DSP_TransformWinInit_Bart32	Create a Bartlett window.
	DSP_TransformWinInit_Black16	Create a Blackman window.
	DSP_TransformWinInit_Black32	Create a Blackman window.
	DSP_TransformWinInit_Cosine16	Create a Cosine (Sine) window.
	DSP_TransformWinInit_Cosine32	Create a Cosine (Sine) window.
	DSP_TransformWinInit_Hamm16	Create a Hamming window.
	DSP_TransformWinInit_Hamm32	Create a Hamming window.
	DSP_TransformWinInit_Hann16	Create a Hanning window.
	DSP_TransformWinInit_Hann32	Create a Hanning window.
	DSP_TransformWinInit_Kaiser16	Create a Kaiser window.
	DSP_TransformWinInit_Kaiser32	Create a Kaiser window.

Vector Math Functions

	Name	Description
	DSP_VectorAbs16	Calculate the absolute value of a vector.
	DSP_VectorAbs32	Calculate the absolute value of a vector.
	DSP_VectorAdd16	Calculate the sum of two vectors.
	DSP_VectorAdd32	Calculate the sum of two vectors.
	DSP_VectorAddc16	Calculate the sum of two vectors.
	DSP_VectorAddc32	Calculate the sum of two vectors.
	DSP_VectorAutocorr16	Computes the Autocorrelation of a Vector.
	DSP_VectorBexp16	Computes the maximum binary exponent of a vector.
	DSP_VectorBexp32	Computes the maximum binary exponent of a vector.
	DSP_VectorChkEqu32	Compares two input vectors, returns an integer '1' if equal, and '0' if not equal.

	DSP_VectorCopy	Copies the elements of one vector to another.
	DSP_VectorCopyReverse32	Reverses the order of elements in one vector and copies them into another.
	DSP_VectorDivC	Divides the first N elements of inVector by a constant divisor, and stores the result in outVector.
	DSP_VectorDotp16	Computes the dot product of two vectors, and scales the output by a binary factor.
	DSP_VectorDotp32	Computes the dot product of two vectors, and scales the output by a binary factor.
	DSP_VectorExp	Computes the EXP (e^x) of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorFill	Fills an input vector with scalar data.
	DSP_VectorLn	Computes the Natural Log, $\ln(x)$, of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorLog10	Computes the $\log_{10}(x)$, of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorLog2	Computes the $\log_2(x)$ of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorMax32	Returns the maximum value of a vector.
	DSP_VectorMaxIndex32	Returns the index of the maximum value of a vector.
	DSP_VectorMean32	Calculates the mean average of an input vector.
	DSP_VectorMin32	Returns the minimum value of a vector.
	DSP_VectorMinIndex32	Returns the index of the minimum value of a vector.
	DSP_VectorMul16	Multiplication of a series of numbers in one vector to another vector.
	DSP_VectorMul32	Multiplication of a series of numbers in one vector to another vector.
	DSP_VectorMulc16	Multiplication of a series of numbers in one vector to a scalar value.
	DSP_VectorMulc32	Multiplication of a series of numbers in one vector to a scalar value.
	DSP_VectorNegate	Inverses the sign (negates) the elements of a vector.
	DSP_VectorRecip	Computes the reciprocal ($1/x$) of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorRMS16	Computes the root mean square (RMS) value of a vector.
	DSP_VectorShift	Shifts the data index of an input data vector.
	DSP_VectorSqrt	Computes the square root of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorStdDev16	Computes the Standard Deviation of a Vector.
	DSP_VectorSub16	Calculate the difference of two vectors.
	DSP_VectorSub32	Calculate the difference of two vectors.
	DSP_VectorSumSquares16	Computes the sum of squares of a vector, and scales the output by a binary factor.
	DSP_VectorSumSquares32	Computes the sum of squares of a vector, and scales the output by a binary factor.
	DSP_VectorVari16	Computes the variance of N elements of a Vector.
	DSP_VectorVariance	Computes the variance of N elements of inVector.
	DSP_VectorZeroPad	Fills an input vector with zeros.

Description

This section describes the Application Programming Interface (API) functions of the DSP Fixed-Point Library.

Refer to each section for a detailed description.

5.4.1.5.1 Complex Math Functions

5.4.1.5.1.1 DSP_ComplexAdd32 Function

C

```
void DSP_ComplexAdd32(
    int32c * indata1,
    int32c * indata2,
    int32c * Output
);
```

Description

Function DSP_ComplexAdd32:

```
void DSP_ComplexAdd32(int32c *indata1, int32c *indata2, int32c *Output);
```

Calculates the sum of two complex numbers, indata1 and indata2, and stores the complex result in Output. Complex numbers must be in the structural form that includes real and imaginary components. The function saturates the output values if maximum or minimum are exceeded. All values are in Q31 fractional data format. $(a + bi) + (c + di) \Rightarrow (a + c) + (b + d)i$

Preconditions

Complex numbers must be in the [int32c](#) format.

Parameters

Parameters	Description
indata1	pointer to input complex number (int32c)
indata2	pointer to input complex number (int32c)

Returns

pointer to result complex numbers ([int32c](#))

None.

Remarks

None.

Example

```
int32c *res, result;
int32c *input1, *input2;
int32c test_complex_1 = {0x40000000, 0x0CCCCCCC};
// (0.5 + 0.1i)
int32c test_complex_2 = {0x73333333, 0xB3333334};
// (0.9 - 0.6i)
res=&result;
input1=&test_complex_1;
input2=&test_complex_2;

DSP_ComplexAdd32(input1, input2, res);

// result = {0x73333333, 0xC0000000} = (0.9 - 0.5i)
```

5.4.1.5.1.2 DSP_ComplexConj16 Function

C

```
void DSP_ComplexConj16(
    int16c * indata,
    int16c * Output
);
```

Description

DSP_ComplexConj16:

```
void DSP_ComplexConj16(int16c *indata, int16c *Output);
```

Calculates the complex conjugate of Indata, and stores the result in Outdata. Both numbers must be in the complex number data structure which includes real and imaginary components. Values are in Q15 fractional data format. The function will saturate the output if maximum or minimum values are exceeded. $(a + bi) \Rightarrow (a - bi)$

Preconditions

Complex numbers must be in the [int32c](#) format.

Parameters

Parameters	Description
indata	pointer to input complex number (int16c)

Returns

pointer to result complex numbers ([int16c](#))

None.

Remarks

None.

Example

```
int16c *res, result;
int16c *input1;
int16c test_complex_1 = {0x4000, 0x0CCC};
//                      (0.5 + 0.1i)

res=&result;
input1=&test_complex_1;

DSP_ComplexConj16(input1, res);

// result = {0x4000, 0xF334} = (0.5 - 0.1i)
```

5.4.1.5.1.3 DSP_ComplexConj32 Function**C**

```
void DSP_ComplexConj32(
    int32c * indata,
    int32c * Output
);
```

Description

Function DSP_ComplexConj32:

```
void DSP_ComplexConj32(int32c *indata, int32c *Output);
```

Calculates the complex conjugate of indata, and stores the result in Output. Both numbers must be in the complex number data structure, which includes real and imaginary components. Values are in Q31 fractional data format. The function will saturate the output if maximum or minimum values are exceeded. $(a + bi) \Rightarrow (a - bi)$

Preconditions

Complex numbers must be in the [int32c](#) format.

Parameters

Parameters	Description
indata1	pointer to input complex number (int32c)

Returns

pointer to result complex numbers ([int32c](#))

None.

Remarks

None.

Example

```
int32c *res, result;
int32c *input1;
int32c test_complex_1 = {0x40000000, 0x0CCCCC};
//                      (0.5 + 0.1i)

res=&result;
input1=&test_complex_1;

DSP_ComplexConj32(input1, res);

// result = {0x40000000, 0xF3333334} = (0.5 - 0.1i)
```

5.4.1.5.1.4 DSP_ComplexDotProd32 Function**C**

```
void DSP_ComplexDotProd32(
    int32c * indata1,
    int32c * indata2,
    int32c * Output
);
```

Description

Function DSP_ComplexDotProd32:

```
void DSP_ComplexDotProd32(int32c *indata1, int32c *indata2, int32c *Output);
```

Calculates the dot product of two complex numbers, indata1 and indata2, and stores the result in Output. All numbers must be in complex structural format that includes real and imaginary components, and the numbers are in fractional Q31 format. The function will saturate the output if it exceeds maximum or minimum ratings. The formula for the dot product is as follows:
 Output(real) = (Input1.re * Input2.re) + (Input1.im * Input2.im); Output(img) = [(Input1.re * Input2.im) - (Input1.im * Input2.re)]i (a + bi) dot (c + di) => (a * c + b * d) + (a * d - b * c)i

Preconditions

Complex numbers must be in the [int32c](#) format.

Parameters

Parameters	Description
indata1	pointer to input complex number (int32c)
indata2	pointer to input complex number (int32c)

Returns

pointer to result complex numbers ([int32c](#))

None.

Remarks

None.

Example

```
int32c *res, result;
int32c *input1, *input2;
int32c test_complex_1 = {0x40000000, 0x0CCCCCCC};
//                      (0.5 + 0.1i)
int32c test_complex_2 = {0x73333333, 0xB3333334};
//                      (0.9 - 0.6i)
res=&result;
input1=&test_complex_1;
input2=&test_complex_2;

DSP_ComplexDotProd32(input1, input2, res);

// result = {0x31EB851E, 0xCE147AE3} = (0.39 - 0.39i)
```

5.4.1.5.1.5 DSP_ComplexMult32 Function**C**

```
void DSP_ComplexMult32(
    int32c * indata1,
    int32c * indata2,
    int32c * Output
);
```

Description

Function DSP_ComplexMult32:

```
void DSP_ComplexMult32(int32c *indata1, int32c *indata2, int32c *Output);
```

Multiplies two complex numbers, indata1 and indata2, and stores the complex result in Output. All numbers must be in the [int32c](#) complex data structure. All data is in Q31 fractional format. The function will saturate if maximum or minimum values are exceeded. $\text{Output}(\text{real}) = (\text{Input1.re} * \text{Input2.re}) - (\text{Input1.im} * \text{Input2.im})$; $\text{Output}(\text{img}) = [(\text{Input1.re} * \text{Input2.im}) + (\text{Input1.im} * \text{Input2.re})]i$ $(a + bi) \times (c + di) \Rightarrow (a * c - b * d) + (a * d + b * c)i$

Preconditions

Complex numbers must be in the [int32c](#) format.

Parameters

Parameters	Description
indata1	pointer to input complex number (int32c)
indata2	pointer to input complex number (int32c)

Returns

pointer to result complex numbers ([int32c](#))

None.

Remarks

None.

Example

```
int32c *res, result;
int32c *input1, *input2;
int32c test_complex_1 = {0x40000000, 0x0CCCCCCC};
//                      (0.5 + 0.1i)
```

```
int32c test_complex_2 = {0x73333333, 0xB3333334};
//                               (0.9 - 0.6i)
res=&result;
input1=&test_complex_1;
input2=&test_complex_2;

DSP_ComplexMult32(input1, input2, res);

// result = {0x4147AE14, 0xE51EB8551} = (0.51 - 0.21i)
```

5.4.1.5.1.6 DSP_ComplexScalarMult32 Function

C

```
void DSP_ComplexScalarMult32(
    int32c * indata,
    int32_t Scalar,
    int32c * Output
);
```

Description

Function DSP_ComplexScalarMult32:

```
void DSP_ComplexScalarMult32(int32c *indata, int32_t Scalar, int32c *Output);
```

Multiplies a complex number, indata, by a scalar number, Scalar, and stores the result in Output. indata and Output must be in [int32c](#) structure with real and imaginary components. All data must be in the fractional Q31 format. The function will saturate if maximum or minimum values are exceeded. Output(real) = (Input1.re * Scalar); Output(img) = [(Input1.im * Scalar)]i (a + bi) * C => (a * C + b * Ci)

Preconditions

Complex numbers must be in the [int32c](#) format.

Parameters

Parameters	Description
indata	pointer to input complex number (int32c)
Scalar	fractional scalar input value (int32_t)

Returns

pointer to result complex numbers ([int32c](#))

None.

Remarks

None.

Example

```
int32c *res, result;
int32c *input1;
int32_t scalarInput = 0x20000000; // 0.25
int32c test_complex_1 = {0x40000000, 0x0CCCCCCC};
//                               (0.5 + 0.1i)

res=&result;
input1=&test_complex_1;

DSP_ComplexScalarMult32(input1, scalarInput, res);

// result = {0x10000000, 0x03333333} = (0.125 + 0.025i)
```

5.4.1.5.1.7 DSP_ComplexSub32 Function

C

```
void DSP_ComplexSub32(
    int32c * indata1,
    int32c * indata2,
    int32c * Output
);
```

Description

Function DSP_ComplexSub32:

```
void DSP_ComplexSub32(int32c *indata1, int32c *indata2, int32c *Output);
```

Calculates the difference of two complex numbers, indata1 less indata2, and stores the complex result in Output. Both numbers must be in a complex data structure, which includes real and imaginary components. The function saturates the output values if maximum or minimum are exceeded. Real and imaginary components are in the Q31 fractional data format. $(a + bi) - (c + di) \Rightarrow (a - c) + (b - d)i$

Preconditions

Complex numbers must be in the `int32c` format.

Parameters

Parameters	Description
indata1	pointer to input complex number (<code>int32c</code>)
indata2	pointer to input complex number (<code>int32c</code>)

Returns

pointer to result complex numbers (`int32c`)

None.

Remarks

None.

Example

```
int32c *res, result;
int32c *input1, *input2;
int32c test_complex_1 = {0x40000000, 0x0CCCCCCC};
//          (0.5 + 0.1i)
int32c test_complex_2 = {0x73333333, 0xB3333334};
//          (0.9 - 0.6i)
res=&result;
input1=&test_complex_1;
input2=&test_complex_2;

DSP_ComplexSub32(input1, input2, res);

// result = {0xCCCCCCCC, 0x59999998} = (-0.4 + 0.7i)
```

5.4.1.5.2 Digital Filter Functions

5.4.1.5.2.1 DSP_FilterFIR32 Function

C

```
void DSP_FilterFIR32(
```

```

int32_t * outdata,
int32_t * indata,
int32_t * coeffs2x,
int32_t * delayline,
int N,
int K,
int scale
);

```

Description

Function DSP_FilterFIR32:

```
void DSP_FilterFIR32(int32_t *outdata, int32_t *indata, int32_t *coeffs2x, int32_t *delayline, int N, int K, int scale);
```

Performs an FIR filter on the vector indata, and stores the output in the vector outdata. The number of samples processed in the array is given by N. The number of filter taps is given by K. The values are scaled upon input by the binary scaling factor (right shift), scale. The array of 2*K coefficients is contained in the array coeffs2x, where the values are in order b0, b1, b2... and repeated. Lastly the delayline is an array of K values that are initialized to zero and represent previous values. All values are in fractional Q31 data format. The function will saturate results if minimum or maximum values are exceeded.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four. K must be greater than 2 and a multiple of 2. delayline must have K elements, and be initialized to zero. coeffs2x must have 2*K elements.

Parameters

Parameters	Description
outdata	pointer to destination array of elements (int32_t)
indata	pointer to source array of elements (int32_t)
coeffs2x	pointer to an array of coefficients (int32_t)
delayline	pointer to an array of delay variables (int32_t)
N	number of points in the array to process (int) number of samples (int)
K	number of filter taps
scale	binary scaler divisor ($1 / 2^{\text{scale}}$) (int)

Returns

None.

Remarks

Filter coefs must be repeated within the array. The array is twice as large as the number of taps, and the values are repeated in order b0, b1, b2,...bn, b0, b1, b2,... bn. The function updates the delayline array, which must be K elements long. The array should be initialized to zero prior to the first processing. It will contain values for processing cascaded filters within a loop.

Example

```

#define TAPS 4
#define numPOINTS 256

int filterN = numPOINTS;
int filterK = TAPS;
int filterScale = 1; // scale output by 1/2^1 => output * 0.5
int32_t FilterCoefs[TAPS*2] = {0x40000000, 0x20000000, 0x20000000, 0x20000000,
                                0x40000000, 0x20000000, 0x20000000, 0x20000000};
// note repeated filter coefs, A B C D A B C D
//      0.5, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25

int32_t outFilterData[numPOINTS]={0};
int32_t inFilterData[numPOINTS];

```

```

int filterDelayLine[TAPS]={0};

while(true)
{
    // put some data into input array, inFilterData, here //
    DSP_FilterFIR32(outFilterData, inFilterData, FilterCoefs, filterDelayLine,
                    filterN, filterK, filterScale);
}

```

5.4.1.5.2.2 DSP_FilterFIRDecim32 Function

C

```

void DSP_FilterFIRDecim32(
    int32_t * outdata,
    int32_t * indata,
    int32_t * coeffs,
    int32_t * delayline,
    int N,
    int K,
    int scale,
    int rate
);

```

Description

Function DSP_FilterFIRDecim32:

```
void DSP_FilterFIRDecim32(int32_t *outdata, int32_t *indata, int32_t *coeffs, int32_t *delayline, int N, int K, int scale, int rate);
```

Compute a FIR decimation filter on the input vector indata, and store the results to the vector outdata. The total number of output elements is set by N, and therefore the outdata array must be at least N in length. The decimation ratio is given by rate. The input is sampled every integer value of rate, skipping every (rate-1) input samples. The input array must therefore be (rate*N) samples long. The amount of filter taps is specified by K. Coeffs specifies the coefficients array. The delayline array holds delay inputs for calculation, and must be initialized to zero prior to calling the filter. Both coeffs and delayline must be K in length. Scale divides the input by a scaling factor by right shifting the number of bits ($1/2^{\text{scale}}$). All values of input, output, and coeffs are given in Q31 fractional data format. The function will saturate if the output value exceeds the maximum or minimum value.

$$Y = b_0 * X_0 + (b_1 * X(-1)) + (b_2 * X(-2))$$

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. delayline must have K elements, and be initialized to zero. coeffs must have K elements. outdata must have N elements indata must have (N*rate) elements

Parameters

Parameters	Description
outdata	pointer to output array of elements (int32_t)
indata	pointer to input array of elements (int32_t)
coeffs	pointer to an array of coefficients (int32_t)
delayline	pointer to an array of delay variables (int32_t)
N	number of output elements to be processed (int)
K	number of filter taps and coeffs (int)
scale	binary scaler divisor ($1 / 2^{\text{scale}}$) (int)
rate	decimation ratio (int)

Returns

None.

Remarks

Coefs are loaded into the array with corresponding to the least delay first (C0, C(-1), C(-2)). K must be greater than rate. Even while decimating the input stream, every input passes through the delayline. So FIR filters of arbitrary length will give the same output as a non-decimating FIR, just with fewer responses.

Example

```
#define N 8           // number of output samples
#define TAPS 5
#define SKIP 3

int testFilterN = N;      // number of output elements
int testFilterK = TAPS;   // number of taps
int testFilterRate = SKIP; // decimation rate R
int32_t outFiltDataDec[N]={0};
int32_t *inTestFilter[N*SKIP];
int filtScaleNum = 1;      // scale output (1 / 2^n) => Y * 0.5

int32_t filtDelayTest[8]={0}; // always initialize to zero

// get pointer to input buffer here //
inTestFilter = &inputBuffer;

DSP_FilterFIRDecim32(outFiltDataDec, inTestFilter, inTestCoefs,
    filtDelayTest, testFilterN, testFilterK, filtScaleNum, testFilterRate);
```

5.4.1.5.2.3 DSP_FilterFIRInterp32 Function**C**

```
void DSP_FilterFIRInterp32(
    int32_t * outdata,
    int32_t * indata,
    int32_t * coeffs,
    int32_t * delayline,
    int N,
    int K,
    int scale,
    int rate
);
```

Description

Function DSP_FilterFIRInterp32:

```
void DSP_FilterFIRInterp32(int32_t *outdata, int32_t *indata, int32_t *coeffs, int32_t *delayline, int N, int K, int scale, int rate);
```

Perform an interpolating FIR filter on the first N samples of indata, and stores the result in outdata. The number of output elements is N*rate. The number of filter taps, K, must be an even multiple of N. The coefficients array, Coeffs, must be K elements long. The delay line array, delayline, must be K/R elements long, and be initialized to zero. All data elements must be in Q31 fractional data format. Scaling is performed via binary shift on the input equivalent to (1/2^shift). The function will saturate the output if it exceeds maximum or minimum values. The function creates R output values for each input value processed. The delayline of previous values is processed with R elements of the coefficient array. Numerically:

$$Y(1,0) = X(0)*C(0) + X(-1)*C(\text{rate}) + X(-2)*C(2*\text{rate}) \dots Y(1,1) = X(0)*C(1) + X(-1)*C(\text{rate}+1) + X(-2)*C(2*\text{rate} + 1) \dots Y(1,\text{rate}) = X(0)*C(N) + X(-1)*C(\text{rate}+N) + X(-2)*C(2*\text{rate} + N) \dots$$

where output Y corresponds to (input,rate) different outputs, input X has (M/rate) sample delays and C is the coefficient array.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. delayline must have (K/R) elements, and be initialized to zero. K (taps) must be an even multiple of R (rate). outdata must have R*N elements.

Parameters

Parameters	Description
outdata	pointer to output array of elements (int32_t)
indata	pointer to input array of elements (int32_t)
coeffs	pointer to an array of coefficients (int32_t)
delayline	pointer to an array of delay variables (int32_t)
N	number of output elements to be processed (int)
K	number of filter taps and coeffs (int)
scale	binary scaler divisor ($1 / 2^{\text{scale}}$) (int)
rate	decimation ratio (int)

Returns

None.

Remarks

The function processes each input (rate) times. With each pass, coefficients are offset so that (K/rate) multiply accumulate cycles occur.

Example

```
// interpret evenly 1/3 spaced values
#define N 4           // number of output samples
#define TAPS 6
#define INTERP 3

int ifiltN = N;
int ifiltK = TAPS; // k must be an even multiple of R
int ifiltR = INTERP;

int32_t ifiltOut[N*INTERP]={0};
int32_t ifiltDelay[2]={0}; // must be initialized to zero
int ifiltScale = 0;        // no scaling

int32_t ifiltCoefsThirds[TAPS]={0x2AAAAAA9, 0x55555555,0x7FFFFFFE,
                                0x55555555,0x2AAAAAA9,0x00000000};
//                                0.333333, 0.6666667, 0.9999999, 0.6666667, 0.3333333, 0

int32_t ifiltInput[N]={0x0CCCCCD, 0x19999999, 0x26666666, 0x33333333};
//                                0.1, 0.2, 0.3, 0.4

DSP_FilterFIRInterp32(ifiltOut, ifiltInput, ifiltCoefsThirds, ifiltDelay,
                    ifiltN, ifiltK, ifiltScale, ifiltR);

// ifiltOut = {0x04444444, 0x08888889, 0x0CCCCCD, 0x11111111, 0x15555555, 0x19999999,
//             0x1DDDDDDD, 0x22222221, 0x26666665, 0x2AAAAAA9,0x2EEEEEEE, 0x33333332}
// = 0.0333, 0.0667, 0.1, 0.1333, 0.1667, 0.2, 0.2333, 0.2667, 0.3, 0.3333, 0.3667, 0.4
```

5.4.1.5.2.4 DSP_FilterIIR16 Function**C**

```
int16_t DSP_FilterIIR16(
    int16_t in,
    int16_t * coeffs,
    int16_t * delayline,
    int B,
    int scale
);
```


Description

Function DSP_FilterIIR16:

```
int16_t DSP_FilterIIR16(int16_t in, int16_t *coeffs, int16_t *delayline, int B, int scale);
```

Performs a single element cascaded biquad IIR filter on the input, in. The filter contains B number of biquad sections, and cascades the output of one to the input of the next. B must be greater than 2 and a multiple of 2. The int16_t output generated by the function is the computation from the final biquad stage. Delay pipeline array delayline must contain 2*B values and be initialized to zero prior to use. The coefficient array must contain 4*B elements, and must be set up in order of biquad a1, a2, b1, b2. A binary (right shift) factor, scale, will scale the output equivalent to $(1/2^{\text{scale}})$. All numerical values must be in Q15 fractional data format. The function will saturate values if maximum or minimum values are exceeded.

$$Y = X_0 + (b_1 * X(-1)) + (b_2 * X(-2)) + (a_1 * Y(-1)) + (a_2 * Y(-2))$$

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. B must be greater than 2 and a multiple of 2. delayline must have 2*B elements, and be initialized to zero. coeffs must have 4*B elements.

Parameters

Parameters	Description
in	input data element X (int16_t)
coeffs	pointer to an array of coefficients (int16_t)
delayline	pointer to an array of delay variables (int16_t)
B	number of cascaded biquad filter groups to process (int)
scale	binary scaler divisor ($1 / 2^{\text{scale}}$) (int)

Returns

Sample output Y (int16_t)

Example

```
#define B 8           // use * biquad filters in cascade

int dataSamples = 256;
int i, j;
biquad16 bquad[B];
int16_t coeffs[4*B]= {0};
int16_t delaylines[2*B]= {0};
int16_t Y, X;
int scaleBquad = 1; // scale output (1 / 2^n) => Y * 0.5

// do something to set up coeffs, for instance this example //

for (j=0; jRemarks:Filter coeffs must be stored within the array as a1, a2, b1, b2, a1, a2, b1,
b2, in order of biquads form input to output. A function to translate the coeffs from biquad
structure to coeffs is available in DSP_FilterIIRSetup16. The function updates the delayline
array, which must be 2*B elements long. The array should be initialized to zero prior to the
first processing. It will contain values for processing cascaded filters within a loop.
```

5.4.1.5.2.5 DSP_FilterIIRBQ16 Function**C**

```
int16_t DSP_FilterIIRBQ16(
    int16_t Xin,
    PARM_EQUAL_FILTER * pFilter
);
```

Description

Function DSP_FilterIIRBQ16:

```
int16_t DSP_FilterIIRBQ16(int16_t Xin, PARM_EQUAL_FILTER *pFilter);
```

Calculates a single pass IIR biquad filter on Xin, and delivers the result as a 16-bit output. All math is performed using 32 bit instructions, with results truncated to 16-bits for the output. The delay register is stored as a 32-bit value for subsequent functions. All values are fractional Q15 and Q31, see data structure for specifics.

$$Y = X(0)*b_0 + (b_1 * X(-1)) + (b_2 * X(-2)) - (a_1 * Y(-1)) - (a_2 * Y(-2))$$

Preconditions

Delay register values should be initialized to zero.

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

Returns

Sample output Y (int16_t)

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Example

```
PARM_EQUAL_FILTER *ptrFilterEQ;
PARM_EQUAL_FILTER FilterEQ;
uint16_t DataIn, DataOut;
ptrFilterEQ = &FilterEQ;

// 48KHz sampling; 1 KHz bandpass filter; Q=0.9
// divide by 2 and convert to Q15
// b0 = 0.06761171785499065
// b1 = 0
// b2 = -0.06761171785499065
// a1 = -1.848823142275648
// a2 = 0.8647765642900187

// note all coefs are half value of original design, gain handled in algorithm
ptrFiltEQ32->b[0]=0x0453;      // feed forward b0 coef
ptrFiltEQ32->b[1]=0;           // feed forward b1 coef
ptrFiltEQ32->b[2]=0xFBAD;      // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
ptrFiltEQ32->a[0]=0x89AD;      // feedback a1 coef
ptrFiltEQ32->a[1]=0x3758;      // feedback a2 coef

for (i=0;i<256;i++)
{
    // *** get some input data here
    DataIn32 = three_hundred_hz[i];

    DataOut = DSP_FilterIIRBQ16(DataIn, ptrFilterEQ);

    // *** do something with the DataOut here
}
```

5.4.1.5.2.6 DSP_FilterIIRBQ16_cascade8 Function

C

```
int16_t DSP_FilterIIRBQ16_cascade8(
    int16_t Xin,
    PARM_EQUAL_FILTER * pFilter_Array
);
```

Description

Function DSP_FilterIIRBQ16_cascade8:

```
int16_t DSP_FilterIIRBQ16_cascade8(int16_t Xin, PARM_EQUAL_FILTER *pFilter_Array);
```

Calculates a single pass IIR biquad cascade filter on Xin, and delivers the result as a 16-bit output. The cascade of filters is 8 unique biquad filters arranged in series such that the output of one is provided as the input to the next. A unique filter coefficient set is provided to each, and 32 bit delay lines are maintained for each. All math is performed using 32 bit instructions, which results truncated to 16-bits for the output. Global gain values are available on the output. Fracgain is a Q15 fractional gain value and expgain is a binary shift gain value. The combination of the two can be utilized to normalize the output as desired. All values are fractional Q15 and Q31, see data structure for specifics.

$Y = Y7 <- Y6 <- Y5 <- Y4 <- Y3 <- Y2 <- Y1 <- Y0$ where each Y_n filter element represents a unique IIR biquad: $Y_n = Y(n-1)*b_0 + (b_1 * Y(n-2)) + (b_2 * Y(n-3)) - (a_1 * Y_n(-1)) - (a_2 * Y_n(-2))$ and: for Y_0 ; $Y(n-1) = Xin(0)$

Preconditions

Delay register values should be initialized to zero.

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

Returns

Sample output Y (int16_t)

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Example

```
PARM_EQUAL_FILTER filtArray[8];
uint16_t dataY, dataX;

// example to use 2 filter blocks as notch filters
// fill entire Filter Array with coefs
for (i=0;i<8;i++)
{
    filtArray[i].Z[0]=0;
    filtArray[i].Z[1]=0;

    // note all coefs are half value of original design, gain handled in algorithm
    // all pass
    filtArray[i].b[0]=0x4000;
    filtArray[i].b[1]=0;           // feed forward b1 coef
    filtArray[i].b[2]=0;         // feed forward b2 coef

    filtArray[i].a[0]=0;         // feedback a1 coef
    filtArray[i].a[1]=0;         // feedback a2 coef
}
```

```

}
// Unique filters for example
// 10KHz notch filter -- divide coefs by 2
b0 = 0.5883783602332997
b1 = -0.17124071441396285
b2 = 0.5883783602332997
a1 = -0.17124071441396285
a2 = 0.1767567204665992

// note all coefs are half value of original design, gain handled in algorithm
filtArray[3].b[0]=0x25a7;           // feed forward b0 coef
filtArray[3].b[1]=0xf508;           // feed forward b1 coef
filtArray[3].b[2]=0x25a7;           // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
filtArray[3].a[0]=0xf508;           // feedback a1 coef
filtArray[3].a[1]=0x0b4f;           // feedback a2 coef

// 1 KHz notch filter -- divide coefs by 2
b0 = 0.9087554064944908
b1 = -1.7990948352036205
b2 = 0.9087554064944908
a1 = -1.7990948352036205
a2 = 0.8175108129889816

// note all coefs are half value of original design, gain handled in algorithm
filtArray[7].b[0]=0x3a29;           // feed forward b0 coef
filtArray[7].b[1]=0x8cdc;           // feed forward b1 coef
filtArray[7].b[2]=0x3a29;           // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
filtArray[7].a[0]=0x8cdc;           // feedback a1 coef
filtArray[7].a[1]=0x3452;           // feedback a2 coef

for (i=0;i<256;i++)
{
    // *** get input data here
    dataX = compound_300_1K_hz16[i];

    dataY = DSP_FilterIIRBQ16_cascade8(dataX, filtArray);

    // *** do something with the DataY here
}

```

5.4.1.5.2.7 DSP_FilterIIRBQ16_cascade8_fast Function

C

```

int16_t DSP_FilterIIRBQ16_cascade8_fast(
    int16_t Xin,
    PARM_EQUAL_FILTER_16 * pFilter_Array
);

```

Description

Function DSP_FilterIIRBQ16_cascade8_fast:

```
int16_t DSP_FilterIIRBQ16_cascade8_fast(int16_t Xin, PARM\_EQUAL\_FILTER\_16 *pFilter_Array);
```

Calculates a single pass IIR biquad cascade filter on Xin, and delivers the result as a 16-bit output. The cascade of filters is 8 unique biquad filters arranged in series such that the output of one is provided as the input to the next. A unique filter coefficient set is provided to each, and 16 bit delay lines are maintained for each. All math is performed using 16 bit instructions, which results rounded to 16-bits for the output. All values are fractional Q15, see data structure for specifics. The function will saturate the output should it exceed maximum or minimum values.

$Y = Y7 <- Y6 <- Y5 <- Y4 <- Y3 <- Y2 <- Y1 <- Y0$ where each Y_n filter element represents a unique IIR biquad: $Y_n = Y(n-1)*b0 + (b1 * Y(n-2)) + (b2 * Y(n-3)) - (a1 * Y_n(-1)) - (a2 * Y_n(-2))$ and: for $Y0$; $Y(n-1) = X_{in}(0)$

Preconditions

Delay register values should be initialized to zero.

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

Returns

Sample output Y (int16_t)

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Example

```

PARM_EQUAL_FILTER_16 filtArray[8];
uint16_t dataY, dataX;

// example to use 2 filter blocks as notch filters
// fill entire Filter Array with coefs
for (i=0;i<8;i++)
{
    filtArray[i].Z[0]=0;
    filtArray[i].Z[1]=0;

    // note all coefs are half value of original design, gain handled in algorithm
    // all pass
    filtArray[i].b[0]=0x4000;
    filtArray[i].b[1]=0;           // feed forward b1 coef
    filtArray[i].b[2]=0;         // feed forward b2 coef

    filtArray[i].a[0]=0;         // feedback a1 coef
    filtArray[i].a[1]=0;         // feedback a2 coef
}

// Unique filters for example
// 10KHz notch filter -- divide coefs by 2
b0 = 0.5883783602332997
b1 = -0.17124071441396285
b2 = 0.5883783602332997
a1 = -0.17124071441396285
a2 = 0.1767567204665992

// note all coefs are half value of original design, gain handled in algorithm
filtArray[3].b[0]=0x25a7;       // feed forward b0 coef
filtArray[3].b[1]=0xf508;       // feed forward b1 coef
filtArray[3].b[2]=0x25a7;       // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
filtArray[3].a[0]=0xf508;       // feedback a1 coef
filtArray[3].a[1]=0x0b4f;       // feedback a2 coef

// 1 KHz notch filter -- divide coefs by 2
b0 = 0.9087554064944908
b1 = -1.7990948352036205
b2 = 0.9087554064944908

```

```

a1 = -1.7990948352036205
a2 = 0.8175108129889816

// note all coefs are half value of original design, gain handled in algorithm
filtArray[7].b[0]=0x3a29;           // feed forward b0 coef
filtArray[7].b[1]=0x8cdc;           // feed forward b1 coef
filtArray[7].b[2]=0x3a29;           // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
filtArray[7].a[0]=0x8cdc;           // feedback a1 coef
filtArray[7].a[1]=0x3452;           // feedback a2 coef

for (i=0;i<256;i++)
{
    // *** get input data here
    dataX = compound_300_1K_hz16[i];

    dataY = DSP_FilterIIRBQ16_cascade8_fast(dataX, filtArray);

    // *** do something with the DataY here
}

```

5.4.1.5.2.8 DSP_FilterIIRBQ16_fast Function

C

```

int16_t DSP_FilterIIRBQ16_fast(
    int16_t Xin,
    PARM_EQUAL_FILTER_16 * pFilter
);

```

Description

Function DSP_FilterIIRBQ16_fast:

int16_t DSP_FilterIIRBQ16_fast(int16_t Xin, [PARM_EQUAL_FILTER_16](#) *pFilter);

Calculates a single pass IIR biquad filter on Xin, and delivers the result as a 16-bit output. All math is performed using 16 bit instructions, with results rounded to 16-bits for the output. The delay register is stored as a 16-bit value for subsequent functions. The function will saturate the results if maximum or minimum fractional values are exceeded. All values are fractional Q15 format.

$$Y = X(0)*b_0 + (b_1 * X(-1)) + (b_2 * X(-2)) - (a_1 * Y(-1)) - (a_2 * Y(-2))$$

Preconditions

Delay register values should be initialized to zero.

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

Returns

Sample output Y (int16_t)

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Example

```
PARM_EQUAL_FILTER_16 *ptrFilterEQ;
```

```

PARM_EQUAL_FILTER_16 FilterEQ;
uint16_t DataIn, DataOut;
ptrFilterEQ = &FilterEQ;

// 48KHz sampling; 1 KHz bandpass filter; Q=0.9
// divide by 2 and convert to Q15
// b0 = 0.06761171785499065
// b1 = 0
// b2 = -0.06761171785499065
// a1 = -1.848823142275648
// a2 = 0.8647765642900187

// note all coefs are half value of original design, gain handled in algorithm
ptrFiltEQ32->b[0]=0x0453;      // feed forward b0 coef
ptrFiltEQ32->b[1]=0;          // feed forward b1 coef
ptrFiltEQ32->b[2]=0xFBAD;     // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
ptrFiltEQ32->a[0]=0x89AD;     // feedback a1 coef
ptrFiltEQ32->a[1]=0x3758;     // feedback a2 coef

for (i=0;i<256;i++)
{
    // *** get some input data here
    DataIn32 = three_hundred_hz[i];

    DataOut = DSP_FilterIIRBQ16_fast(DataIn, ptrFilterEQ);

    // *** do something with the DataOut here
}

```

5.4.1.5.2.9 DSP_FilterIIRBQ16_parallel8 Function

C

```

int16_t DSP_FilterIIRBQ16_parallel8(
    int16_t Xin,
    PARM_EQUAL_FILTER * pFilter
);

```

Description

Function DSP_FilterIIRBQ16_parallel8:

int16_t DSP_FilterIIRBQ16_parallel8(int16_t Xin, [PARM_EQUAL_FILTER](#) *pFilter);

Calculates a 8 parallel, single pass IIR biquad filters on Xin, sums the result and delivers the result as a 16-bit output. All math is performed using 32 bit instructions, which results truncated to 16-bits for the output. The delay register is stored as a 32-bit value for subsequent functions. Output is tuned by 2 multiplier factors. First each parallel section has a fractional gain (attenuation) that enables individual scaling of that section. Second, a global binary (log2N) gain is applied to the result. The combination of gain factors enable both gain and attenuation. All values are fractional Q15 and Q31, see data structure for specifics.

$Y = Y7/8 + Y6/8 + Y5/8 + Y4/8 + Y3/8 + Y2/8 + Y1/8 + Y0/8$ where each Y_n filter element represents a unique IIR biquad: $Y_n = X(0)*b0 + (b1 * X(n-1)) + (b2 * X(n-2)) - (a1 * Yn(-1)) - (a2 * Yn(-2))$

Preconditions

Delay register values should be initialized to zero. The sum of all fracgain should be ≤ 1

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

Returns

Sample output Y (int16_t)

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Example

```

PARAM_EQUAL_FILTER filtArrayPara[8];
uint16_t dataY, dataX;

// fill entire Filter Array with coefs
for (i=0;i<8;i++)
{
    filtArrayPara[i].Z[0]=0;
    filtArrayPara[i].Z[1]=0;

    filtArrayPara[i].G.fracGain = 0x7FFF;           // gain = 1 default
    filtArrayPara[i].G.expGain = 1;                // == 2^N; gain of 2

    // note all coefs are half value of original design, gain handled in algorithm
    // none pass -- default
    filtArrayPara[i].b[0]=0;           // feed forward b0 coef
    filtArrayPara[i].b[1]=0;           // feed forward b1 coef
    filtArrayPara[i].b[2]=0;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArrayPara[i].a[0]=0;           // feedback a1 coef
    filtArrayPara[i].a[1]=0;           // feedback a2 coef
}

// 1K bandpass Q=0.9
filtArrayPara[7].G.fracGain = 0x4000; // gain = 0.5 because using 2 outputs
// note all coefs are half value of original design, gain handled in algorithm
filtArrayPara[7].b[0]=0x04ad;
filtArrayPara[7].b[1]=0;           // feed forward b1 coef
filtArrayPara[7].b[2]=0xfb53;      // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
filtArrayPara[7].a[0]=0x8a90;      // feedback a1 coef
filtArrayPara[7].a[1]=0x36a4;      // feedback a2 coef

// 300 Hz bandpass Q=0.9
filtArrayPara[6].G.fracGain = 0x1000; // gain = 0.125 as an example
// note all coefs are half value of original design, gain handled in algorithm
filtArrayPara[6].b[0]=0x017b;      // feed forward b0 coef
filtArrayPara[6].b[1]=0;           // feed forward b1 coef
filtArrayPara[6].b[2]=0xfe85;      // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
filtArrayPara[6].a[0]=0x8316;      // feedback a1 coef
filtArrayPara[6].a[1]=0x3d08;      // feedback a2 coef

for (i=0;i<256;i++)
{
    // *** get input data here
    dataX = compound_300_1K_hz16[i];

    dataY = DSP_FilterIIRBQ16_cascade8_fast(dataX, filtArray);
}

```



```
// *** do something with the DataY here
}
```

5.4.1.5.2.10 DSP_FilterIIRBQ16_parallel8_fast Function

C

```
int16_t DSP_FilterIIRBQ16_parallel8_fast(
    int16_t Xin,
    PARM_EQUAL_FILTER_16 * pFilter
);
```

Description

Function DSP_FilterIIRBQ16_parallel8_fast:

```
int16_t DSP_FilterIIRBQ16_parallel8_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter);
```

Calculates a 8 parallel, single pass IIR biquad filters on Xin, sums the result and delivers the result as a 16-bit output. All math is performed using 16 bit instructions, which results rounded to 16-bits for the output. The delay register is stored as a 16-bit value for subsequent functions. Output is tuned by 2 multiplier factors. First each parallel section has a fractional gain (attenuation) that enables individual scaling of that section. Second, a global binary (log2N) gain is applied to the result. The combination of gain factors enable both gain and attenuation. All values are fractional Q15. The function will round outputs and saturate if maximum or minimum values are exceeded.

$Y = Y7/8 + Y6/8 + Y5/8 + Y4/8 + Y3/8 + Y2/8 + Y1/8 + Y0/8$ where each Y_n filter element represents a unique IIR biquad: $Y_n = X(0)*b_0 + (b_1 * X(n-1)) + (b_2 * X(n-2)) - (a_1 * Y_n(-1)) - (a_2 * Y_n(-2))$

Preconditions

Delay register values should be initialized to zero. The sum of all fracgain should be ≤ 1

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

Returns

Sample output Y (int16_t)

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Example

```
PARM_EQUAL_FILTER_16 filtArrayPara[8]; // note change in data structure
uint16_t dataY, dataX;

// fill entire Filter Array with coefs
for (i=0;i<8;i++)
{
    filtArrayPara[i].Z[0]=0;
    filtArrayPara[i].Z[1]=0;

    filtArrayPara[i].G.fracGain = 0x7FFF; // gain = 1 default
    filtArrayPara[i].G.expGain = 1;       // log2N; gain of 2

    // note all coefs are half value of original design, gain handled in algorithm
    // none pass -- default
    filtArrayPara[i].b[0]=0; // feed forward b0 coef
```

```

    filtArrayPara[i].b[1]=0;           // feed forward b1 coef
    filtArrayPara[i].b[2]=0;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArrayPara[i].a[0]=0;           // feedback a1 coef
    filtArrayPara[i].a[1]=0;           // feedback a2 coef
}

// 1K bandpass Q=0.9
filtArrayPara[7].G.fracGain = 0x4000; // gain = 0.5 because using 2 outputs
// note all coefs are half value of original design, gain handled in algorithm
filtArrayPara[7].b[0]=0x04ad;
filtArrayPara[7].b[1]=0;               // feed forward b1 coef
filtArrayPara[7].b[2]=0xfb53;          // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
filtArrayPara[7].a[0]=0x8a90;          // feedback a1 coef
filtArrayPara[7].a[1]=0x36a4;          // feedback a2 coef

// 300 Hz bandpass Q=0.9
filtArrayPara[6].G.fracGain = 0x1000; // gain = 0.125 as an example
// note all coefs are half value of original design, gain handled in algorithm
filtArrayPara[6].b[0]=0x017b;          // feed forward b0 coef
filtArrayPara[6].b[1]=0;               // feed forward b1 coef
filtArrayPara[6].b[2]=0xfe85;          // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
filtArrayPara[6].a[0]=0x8316;          // feedback a1 coef
filtArrayPara[6].a[1]=0x3d08;          // feedback a2 coef

for (i=0;i<256;i++)
{
    // *** get input data here
    dataX = compound_300_1K_hz16[i];

    dataY = DSP_FilterIIRBQ16_cascade8_fast(dataX, filtArray);

    // *** do something with the DataY here
}

```

5.4.1.5.2.11 DSP_FilterIIRBQ32 Function

C

```

int32_t DSP_FilterIIRBQ32(
    int32_t Xin,
    PARM_EQUAL_FILTER_32 * pFilter
);

```

Description

Function DSP_FilterIIRBQ32:

```
int32_t DSP_FilterIIRBQ32(int32_t Xin, PARM_EQUAL_FILTER_32 *pFilter);
```

Calculates a single pass IIR biquad filter on Xin, and delivers the result as a 16-bit output. All math is performed using 32 bit instructions, with results truncated to 32-bits for the output. The delay register is stored as a 32-bit value for subsequent functions. All values are fractional Q31, see data structure for specifics.

$$Y = X(0)*b_0 + (b_1 * X(-1)) + (b_2 * X(-2)) - (a_1 * Y(-1)) - (a_2 * Y(-2))$$

Preconditions

Delay register values should be initialized to zero.

Parameters

Parameters	Description
Xin	input data element X (int32_t)
pFilter	pointer to high resolution filter coef and delay structure

Returns

Sample output Y (int32_t)

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Example

```

PARAM_EQUAL_FILTER_32 *ptrFiltEQ32;
PARAM_EQUAL_FILTER_32 FilterEQ32;
int32_t DataIn32, DataOut32;

ptrFiltEQ32 = &FilterEQ32;

ptrFiltEQ32->Z[0]=0;
ptrFiltEQ32->Z[1]=0;

// 1000 Hz Q= 0.9 BP filter design, 44.1K sampling
//
// b0 = 0.07311778239751009 forward
// b1 = 0
// b2 = -0.07311778239751009
// a1 = -1.8349811166056893 back
// a2 = 0.8537644352049799

// note all coefs are half value of original design, gain handled in algorithm
ptrFiltEQ32->b[0]=0x04ADF635; // feed forward b0 coef
ptrFiltEQ32->b[1]=0; // feed forward b1 coef
ptrFiltEQ32->b[2]=0xFB5209CB; // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
ptrFiltEQ32->a[0]=0x8A8FAB5D; // feedback a1 coef
ptrFiltEQ32->a[1]=0x36A41395; // feedback a2 coef

for (i=0;i<256;i++)
{
    // *** get input data here
    DataIn32 = three_hundred_hz[i];

    DataOut32 = DSP_FilterIIRBQ32(DataIn32, ptrFiltEQ32);

    // *** do something with the DataOut32 here
}

```

5.4.1.5.2.12 DSP_FilterIIRSetup16 Function**C**

```

void DSP_FilterIIRSetup16(
    int16_t * coeffs,
    biquad16 * bq,
    int B
);

```

Description

Function DSP_FilterIIRSetup16:

```
void DSP_FilterIIRSetup16(int16_t *coeffs, biquad16 *bq, int B);
```

Converts an array of biquad coefficients, bq, into an linear array of coefficients, coeffs. The output array must be 4*B elements long. The number of biquads in the resulting factor is given by B. All numerical values must be in Q15 fractional data format.

Preconditions

coeffs must have 4*B elements.

Parameters

Parameters	Description
coeffs	pointer to an array of coefficients (int16_t)
bq	pointer to array of biquad structure filter coefs (biquad16)
B	number of cascaded biquad filter groups to process (int)

Returns

None.

Remarks

None.

Example

see DSP_FilterIIR16 [for](#) example.

5.4.1.5.2.13 DSP_FilterLMS16 Function**C**

```
int16_t DSP_FilterLMS16(
    int16_t in,
    int16_t ref,
    int16_t * coeffs,
    int16_t * delayline,
    int16_t * error,
    int K,
    int16_t mu
);
```

Description

Function DSP_FilterLMS16:

```
int16_t DSP_FilterLMS16(int16_t in, int16_t ref, int16_t *coeffs, int16_t *delayline, int16_t *error, int K, int16_t mu);
```

Computes an LMS adaptive filter on the input in. Filter output of the FIR is given as a 16 bit value. The filter target is ref, and the calculation difference between the output and the target is error. The filter adapts its coefficients, coefs, on each pass. The number of coefficients (filter taps) is given by the value K. The delayline array should be initialized to zero prior to calling the filter for the first time, and have K elements. The value mu is the rate at which the filter adapts. All values are Q15 fractional numbers. The function will saturate the output if it exceeds maximum or minimum values. The LMS will adapt its coefs to attempt to drive the output value toward the ref value. The rate of adaption on each pass depends on mu and the error from the previous calculation.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. delayline must have (K) elements, and be initialized to zero. K (taps) must be a multiple of 4, and >= 8. mu must be positive.

Parameters

Parameters	Description
in	input data value (int16_t)
ref	target output value (int16_t)
coeffs	pointer to an array of coefficients (int16_t)
delayline	pointer to an array of delay variables (int16_t)
error	output minus reference (int16_t)
K	number of filter taps and coeffs (int)
mu	adaption rate (int16_t)

Returns

(int16_t) - FIR filter output

Remarks

Filter coefs may start at random or zero value, but convergence is dependent on the amount of update required.

Example

```
#define lmsTAPS 8

int16_t lmsOut;
int lmsTaps = lmsTAPS;

int16_t lmsCoefs[lmsTAPS]={0x5000, 0x4000,0x3000, 0x2000,0x1000, 0x0000,0xF000, 0xE000};
int16_t lmsDelay[lmsTAPS]={0};
int16_t *ptrLMSError;
int16_t lmsError = 0x0200;
int16_t inVal=0;

int16_t refVal = 0x0CCC; // some target value = 0.1
int16_t lmsAdapt = 0x3000;

ptrLMSError = &lmsError;

for (i=0;i<200;i++)
{
    // get some input value here //
    if (i < 100)
    {
        inVal = 0x4233;
    }
    else
    {
        inVal = 0xCF10;
    }

    lmsOut = DSP_FilterLMS16(inVal, refVal, lmsCoefs, lmsDelay,
        ptrLMSError, lmsTaps, lmsAdapt);
}
```

5.4.1.5.3 Matrix Math Functions**5.4.1.5.3.1 DSP_MatrixAdd32 Function****C**

```
void DSP_MatrixAdd32(
    matrix32 * resMat,
    matrix32 * srcMat1,
```

```
    matrix32 * srcMat2
);
```

Description

Function DSP_MatrixAdd32:

```
void DSP_MatrixAdd32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);
```

Vector summation of two matrices, where both have 32-bit integer elements. The resulting output will saturate if element addition exceeds **MAX32** or **MIN32**.

Preconditions

Both matrices must be equivalent in rows and columns. Both Matrices must be set into structure (ROWS, COLUMNS, vector_pointer).

Parameters

Parameters	Description
resMat	pointer to new sum Matrix C (*int32_t)
srcMat1	pointer to the Matrix A structure (*int32_t)
srcMat2	pointer to the Matrix B structure (*int32_t)

Returns

None.

Remarks

Execution Time (cycles): 225 cycles + 23 / matrix_element. The function will saturate the output value if it exceeds maximum limits per element.

Example

```
#define ROW 2
#define COL 2

matrix32 *resMat, *srcMat1, *srcMat2;
int32_t result[ROW*COL];

int32_t matA[ROW*COL] = {1,2,3,4};
int32_t matB[ROW*COL] = {2,4,6,8};

matrix32 mat, mat2, mat3;
resMat=&mat;
srcMat1=&mat2;
srcMat2=&mat3;

srcMat1->row=ROW;
srcMat1->col=COL;
srcMat1->pMatrix=matA;

srcMat2->col=COL;
srcMat2->row=ROW;
srcMat2->pMatrix=matB;

resMat->row=ROW;
resMat->col=COL;
resMat->pMatrix=result;

DSP_MatrixAdd32(resMat, srcMat1, srcMat2);

// result[i] = matA[i] + matB[i] = {3,6,9,0xA}
```

5.4.1.5.3.2 DSP_MatrixEqual32 Function

C

```
void DSP_MatrixEqual32(
    matrix32 * resMat,
    matrix32 * srcMat
);
```

Description

Function DSP_MatrixEqual32:

```
void DSP_MatrixEqual32(matrix32 *resMat, matrix32 *srcMat);
```

Vector copy of all elements from one matrix to another. C is a duplicate of A.

Preconditions

None.

Parameters

Parameters	Description
resMat	pointer to completed new Matrix C (*int32_t)
srcMat	pointer to the Matrix A structure (*int32_t)

Returns

None.

Remarks

Execution Time (cycles): 163 cycles + 12 / matrix_element.

Example

```
#define ROW 2
#define COL 2

matrix32 *resMat, *srcMat1, *srcMat2;
int32_t result[ROW*COL];

int32_t matA[ROW*COL] = {5,2,-3,8};

matrix32 mat, mat2;
resMat=&mat;
srcMat1=&mat2;

srcMat1->row=ROW;
srcMat1->col=COL;
srcMat1->pMatrix=matA;

resMat->row=ROW;
resMat->col=COL;
resMat->pMatrix=result;

DSP_MatrixEqual32(resMat, srcMat1, srcMat2);

// result[i] = matA[i] = {5, 2, -3, 8}
```

5.4.1.5.3.3 DSP_MatrixInit32 Function

C

```
void DSP_MatrixInit32(
    int32_t * data_buffer,
```

```

    int N,
    int32_t num
);

```

Description

Function DSP_MatrixInit32:

```
void DSP_MatrixInit32(int32_t *data_buffer, int N, int32_t num);
```

Copy the value num into the first N Matrix elements of data_buffer.

Preconditions

data_buffer must be predefined to be equal to or greater than N elements. N must be a factor of four, or it will truncate to the nearest factor of four.

Parameters

Parameters	Description
data_buffer	pointer to the Matrix to be initialized (int32_t[M*N])
N	number of elements to be initialized (int32_t)
num	value to be initialized into the matrix (int32_t)

Returns

None.

Remarks

None.

Example

```

#define ROW 3
#define COL 3

int32_t numElements = 4; // multiple of 4
int valueElements = -1;

int32_t matA[ROW*COL] = {5,2,-3,8,4,2,-6,8,9};

DSP_MatrixInit32(matA, numElements, valueElements);

// matA[i] = {-1,-1,-1,-1,4,2,-6,8,9}

```

5.4.1.5.3.4 DSP_MatrixMul32 Function

C

```

void DSP_MatrixMul32(
    matrix32 * resMat,
    matrix32 * srcMat1,
    matrix32 * srcMat2
);

```

Description

Function DSP_MatrixMul32:

```
void DSP_MatrixMul32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);
```

Multiplication of two matrices, with inputs and outputs being in fractional Q31 numerical format. The output elements will saturate if the dot product exceeds maximum or minimum fractional values.

Preconditions

Matrices must be aligned such that columns of A = rows of B. resMat must have the format of rows of A, columns of B. All

Matrices must be set into structure (ROWS, COLUMNS, vector_pointer).

Parameters

Parameters	Description
resMat	pointer to different Matrix C structure (*int32_t)
srcMat1	pointer to the Matrix A structure (*int32_t)
srcMat2	pointer to the Matrix B structure (*int32_t)

Returns

None.

Remarks

Execution Time (cycles): 319 cycles + 38 / output matrix_element. The function will saturate the output value if it exceeds maximum limits per element.

Example

```
#define ROW1 3
#define COL1 2
#define ROW2 2
#define COL2 2

matrix32 *resMat, *srcMat1, *srcMat2;
int32_t result[ROW1*COL2];

int32_t test_MatrixA[ROW1*COL1]=
{
    0x40000000,0x20000000,    // 0.5,  0.25
    0xD999999A,0x4CCCCCCC,    // -0.3, 0.6
    0xC0000000,0x0CCCCCDD    // -0.5  0.1
};

int32_t test_MatrixB[ROW2*COL2]=
{
    0x40000000,0x20000000,    // 0.5,  0.25
    0x0CCCCCDD,0xCCCCCDD    // 0.1, -0.4
};

matrix32 mat, mat2, mat3;
resMat=&mat;
srcMat1=&mat2;
srcMat2=&mat3;

srcMat1->row=ROW1;
srcMat1->col=COL1;
srcMat1->pMatrix=test_MatrixA;

srcMat2->col=COL2;
srcMat2->row=ROW2;
srcMat2->pMatrix=test_MatrixB;

resMat->row=ROW1; // note resulting matrix MUST have ROW1 & COL2 format
resMat->col=COL2;
resMat->pMatrix=result;

DSP_MatrixMul32(resMat, srcMat1, srcMat2);

// result[] = matA[] x matB[] =
// { 0x23333333, 0x03333333 // 0.275, 0.025
//   0xF47AE147, 0xD7AE147B // -0.9, -0.315
//   0xE147AE14, 0xEAE147AE } // -0.24, -0.165
```

5.4.1.5.3.5 DSP_MatrixScale32 Function

C

```
void DSP_MatrixScale32(
    int32_t * data_buffer,
    int N,
    int32_t num
);
```

Description

Function DSP_MatrixScale32:

```
void DSP_MatrixScale32(int32_t *data_buffer, int N, int32_t num);
```

Multiply the first N elements of an input buffer by a fixed scalar num. The resulting value is stored back into the input buffer. N number total samples of the input buffer are processed. All values are in Q31 fractional integer format. The result of calculations is saturated to the [MAX32](#) or [MIN32](#) value if exceeded.

Preconditions

data_buffer must be predefined to be equal to or greater than N elements. N must be a factor of four, or will truncate to the nearest factor of four.

Parameters

Parameters	Description
data_buffer	pointer to the Matrix to be initialized (int32_t[M*N])
N	number of elements to be initialized (int)
num	value to be initialized into the matrix (int32_t)

Returns

None.

Remarks

Execution time (cycles): 190 + 9 cycles / element, typical.

Example

```
int32_t numScale = 0x40000000; // 0.5
int valN = 12;
int32_t inputBufScale[12] = {0x40000000, 0x40000000, 0x20000000, 0x20000000,
    0x19999999, 0xCCCCCCCD, 0xF3333333, 0x80000000,
    0x7FFFFFFF, 0x00000000, 0x40000000, 0x70000000 };
// 0.5, 0.5, 0.25, 0.25, 0.25, 0.2, -0.4, -0.1, -1, 1, 0, 0.5, 0.875

DSP_MatrixScale32(inputBufScale, valN, numScale);

// inputBufScale[i] = {0x20000000, 0x20000000, 0x10000000, 0x10000000,
// 0x0CCCCCCC, 0xE6666666, 0xF9999999, 0xC0000000,
// 0x3FFFFFFF, 0x00000000, 0x20000000, 0x38000000}
// 0.25, 0.25, 0.125, 0.125, 0.1, -0.2, -0.05, -0.5, 0.5, 0, 0.25, 0.4375
```

5.4.1.5.3.6 DSP_MatrixSub32 Function

C

```
void DSP_MatrixSub32(
    matrix32 * resMat,
    matrix32 * srcMat1,
    matrix32 * srcMat2
);
```

Description

Function DSP_MatrixSub32:

```
void DSP_MatrixSub32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);
```

Vector subtraction of two matrices, where both have 32-bit integer elements. The resulting output will saturate if element addition exceeds **MAX32** or **MIN32**.

Preconditions

Both matrices must be equivalent in rows and columns. All Matrices must be set into structure (ROWS, COLUMNS, vector_pointer)

Parameters

Parameters	Description
resMat	pointer to different Matrix C structure (*int32_t)
srcMat1	pointer to the Matrix A structure (*int32_t)
srcMat2	pointer to the Matrix B structure (*int32_t)

Returns

None.

Remarks

Execution Time (cycles): 222 cycles + 21 / matrix_element. The function will saturate the output value if it exceeds maximum limits per element.

Example

```
#define ROW 2
#define COL 2

matrix32 *resMat, *srcMat1, *srcMat2;
int32_t result[ROW*COL];

int32_t matA[ROW*COL] = {5, 2, -3, 8};
int32_t matB[ROW*COL] = {2, 2, 2, 2};

matrix32 mat, mat2, mat3;
resMat=&mat;
srcMat1=&mat2;
srcMat2=&mat3;

srcMat1->row=ROW;
srcMat1->col=COL;
srcMat1->pMatrix=matA;

srcMat2->col=COL;
srcMat2->row=ROW;
srcMat2->pMatrix=matB;

resMat->row=ROW;
resMat->col=COL;
resMat->pMatrix=result;

DSP_MatrixSub32(resMat, srcMat1, srcMat2);

// result[i] = matA[i] - matB[i] = {3, 0, -5, 6}
```

5.4.1.5.3.7 DSP_MatrixTranspose32 Function

C

```
void DSP_MatrixTranspose32(
    matrix32 * desMat,
    matrix32 * srcMat
);
```

Description

Function DSP_MatrixTranspose32:

```
void DSP_MatrixTranspose32(matrix32 *desMat, matrix32 *srcMat);
```

Transpose of rows and columns of a matrix.

Preconditions

Matrix definitions for ROWS and COLS must be transposed prior to the function call.

Parameters

Parameters	Description
desMat	pointer to transposed new Matrix C (*int32_t)
srcMat	pointer to the Matrix A structure (*int32_t)

Returns

None.

Remarks

Execution Time (cycles): 210 cycles + 10 / matrix_element.

Example

```
#define ROW 3
#define COL 4

matrix32 *resMat, *srcMat1;
int32_t result[ROW*COL];

int32_t matA[ROW*COL] = { 1,  2,  3,  4,
                          5,  6,  7,  8,
                          -1, -3, -5, -7};

matrix32 mat, mat2;
resMat=&mat;
srcMat1=&mat2;

srcMat1->row=ROW;
srcMat1->col=COL;
srcMat1->pMatrix=matA;

resMat->row=COL; // note the shift in columns and rows
resMat->col=ROW;
resMat->pMatrix=result;

DSP_MatrixTranspose32(resMat, srcMat1);

// result[] = matA(T)[] = {  1,  5, -1,
//                           2,  6, -3,
//                           3,  7, -5,
//                           4,  8, -7}
```

5.4.1.5.4 Transform Functions

5.4.1.5.4.1 DSP_TransformFFT16 Function

C

```
void DSP_TransformFFT16(
    int16c * dout,
    int16c * din,
    int16c * twiddles,
    int16c * scratch,
    int log2N
);
```

Description

Function DSP_TransformFFT16:

```
void DSP_TransformFFT16(int16c *dout, int16c *din, int16c *twiddles, int16c *scratch, int log2N);
```

Performs an complex FFT on the input, din, and stores the complex result in dout. Performs $2^{\log_2 N}$ point calculation, and the working buffer scratch as well as the input and output must be $2^{\log_2 N}$ in length. Coefficient twiddle factors come from twiddles, and may be loaded with the use of [DSP_TransformFFT16_setup](#). All values are 16 bit (Q15) fractional.

Preconditions

din, dout, twiddles and scratch must have N elements N is calculated as $2^{\log_2 N}$ $\log_2 N$ must be ≥ 3 FFT factors must be calculated in advance, use [DSP_TransformFFT16_setup](#)

Parameters

Parameters	Description
dout	pointer to complex output array (int16c)
din	pointer to complex input array (int16c)
twiddles	pointer to an complex array of factors (int16c)
scratch	pointer to a complex scratch pad buffer (int16c)
log2N	binary exponent of number of samples (int)

Returns

None.

Remarks

Scratch must be declared but need not be initialized. Din may be aided with a window function prior to calling the FFT, but is not required. Din is a complex number array, but may be loaded solely with real numbers if no phase information is available.

Example

```
int log2N = 8; // log2(256) = 8
int fftSamples = 256;

int16c *fftDin;
int16c fftDout[fftSamples];
int16c scratch[fftSamples];
int16c fftCoefs[fftSamples];

int16c *fftc;
fftc = &fftCoefs;

DSP_TransformFFT16_setup(fftc, log2N); // call setup function

while (1)
```

```

{
    fftDin = &ffftin_8Khz_long_window16;  // get 256 point complex data

    DSP_TransformFFT16(fftDout, fftDin, fftc, scratch, log2N);

    // do something with the output, fftDout
};

```

5.4.1.5.4.2 DSP_TransformFFT16_setup Function

C

```

void DSP_TransformFFT16_setup(
    int16c * twiddles,
    int log2N
);

```

Description

Function DSP_TransformFFT16_setup:

```
void DSP_TransformFFT16_setup(int16c *twiddles, int log2N);
```

Calculates the N twiddle factors required to operate the FFT16 function. These factors are done in serial fashion, and require considerable processing power. Ideally this function would be run only once prior to an ongoing FFT, and the results held in a buffer.

Preconditions

twiddles must be N in length N is calculated ($2^{\log 2N}$)

Parameters

Parameters	Description
twiddles	pointer to a complex array of factors (int16c)
log2N	binary exponent of number of data points (int)

Returns

None.

Remarks

This function is of considerable length and executed in C. It is recommended it only be called once for any given FFT length in time sensitive applications.

Example

see DSP_TransformFFT16 [for](#) example.

5.4.1.5.4.3 DSP_TransformFFT32 Function

C

```

void DSP_TransformFFT32(
    int32c * dout,
    int32c * din,
    int32c * twiddles,
    int32c * scratch,
    int log2N
);

```

Description

Function DSP_TransformFFT32:

```
void DSP_TransformFFT32(int32c *dout, int32c *din, int32c *twiddles, int32c *scratch, int log2N);
```

Performs an complex FFT on the input, din, and stores the complex result in dout. Performs $2^{\log_2 N}$ point calculation, and the working buffer scratch as well as the input and output must be $2^{\log_2 N}$ in length. Coefficient twiddle factors come from twiddles, and may be loaded with the use of [DSP_TransformFFT16_setup](#). All values are 16 bit (Q31) fractional.

Preconditions

din, dout, twiddles and scratch must have N elements N is calculated as $2^{(\log_2 N)}$ $\log_2 N$ must be ≥ 3 FFT factors must be calculated in advance, use [DSP_TransformFFT32_setup](#)

Parameters

Parameters	Description
dout	pointer to complex output array (int32c)
din	pointer to complex input array (int32c)
twiddles	pointer to an complex array of FFT factors (int32c)
scratch	pointer to a complex scratch pad buffer (int32c)
log2N	binary exponent of number of samples (int)

Returns

None.

Remarks

Scratch must be declared but need not be initialized. Din may be aided with a window function prior to calling the FFT, but is not required. Din is a complex number array, but may be loaded solely with real numbers if no phase information is available.

Example

```
int log2N = 8; // log2(256) = 8
int fftSamples = 256;

int32c *fftDin;
int32c  fftDout[fftSamples];
int32c  scratch[fftSamples];
int32c  fftCoefs[fftSamples];

int32c *fftc;
fftc = &fftCoefs;

DSP_TransformFFT32_setup(fftc, log2N); // call setup function

while (1)
{
    fftDin = &fftin_5Khz_long_window32; // get 256 point complex data

    DSP_TransformFFT32(fftDout, fftDin, fftc, scratch, log2N);

    // do something with the output, fftDout
};
```

5.4.1.5.4.4 DSP_TransformFFT32_setup Function

C

```
void DSP_TransformFFT32_setup(
    int32c * twiddles,
    int log2N
);
```

Description

Function DSP_TransformFFT32_setup:

```
void DSP_TransformFFT32_setup(int32c *twiddles, int log2N);
```

Calculates the N FFT twiddle factors required to operate the FFT32 function. These factors are done in serial fashion, and require considerable processing power. Ideally this function would be run only once prior to an ongoing FFT, and the results held in a buffer.

Preconditions

twiddles must be N in length N is calculated ($2^{\log_2 N}$)

Parameters

Parameters	Description
twiddles	pointer to a complex array of coefficients (int32c)
log2N	binary exponent of number of data points (int)

Returns

None.

Remarks

This function is of considerable length and executed in C. It is recommended it only be called once for any given FFT length in time sensitive applications.

Example

see `DSP_TransformFFT32` [for](#) example.

5.4.1.5.4.5 DSP_TransformIFFT16 Function

C

```
void DSP_TransformIFFT16(
    int16c * dout,
    int16c * din,
    int16c * twiddles,
    int16c * scratch,
    int log2N
);
```

Description

Function DSP_TransformIFFT16:

```
void DSP_TransformIFFT16(int16c *dout, int16c *din, int16c *twiddles, int16c *scratch, int log2N);
```

Performs an complex Inverse FFT on the input, din, and stores the complex result in dout. Performs $2^{\log_2 N}$ point calculation, and the working buffer scratch as well as the input and output must be $2^{\log_2 N}$ in length. Coefficient twiddle factors come from twiddles, and may be loaded with the use of [DSP_TransformFFT16_setup](#). All values are 16 bit (Q15) fractional.

Preconditions

din, dout, twiddles and scratch must have N elements N is calculated as $2^{(\log_2 N)}$ $\log_2 N$ must be ≥ 3 FFT factors must be calculated in advance, use [DSP_TransformFFT16_setup](#)

Parameters

Parameters	Description
dout	pointer to complex output array (int16c)
din	pointer to complex input array (int16c)
twiddles	pointer to an complex array of factors (int16c)
scratch	pointer to a complex scratch pad buffer (int16c)
log2N	binary exponent of number of samples (int)

Returns

None.

Remarks

Scratch must be declared but need not be initialized. Din may be aided with a window function prior to calling the FFT, but is not required. A very similar function to the FFT is executed for the inverse FFT. This requires twiddle factors set in advance with the same method as used in the FFT. Complex conjugate and scaling are handled within the algorithm. The output is scaled using binary shifting based on log2N. Since the algorithm reduces the output by a scale factor of log2N, the resolution is reduced proportionally to the number of data points.

Example

```
int ilog2N = 10; // log2(64) = 6; log2(256) = 8; log2(1024) = 10;
int ifftSamples = pow(2,ilog2N);

int16c *ifftDin;
int16c ifftDout[ifftSamples];
int16c iscratch[ifftSamples];
int16c ifftCoefs[ifftSamples];
int16c ifftTimeOut[ifftSamples];

// set up twiddle factors, these are used for both FFT and iFFT

int16c *ifftc;
ifftc = &ifftCoefs;
DSP_TransformFFT16_setup( ifftc, ilog2N); // call to coef setup

// in this example, we take an FFT of an original time domain (sine wave)
// the output of the FFT is used as the input of the iFFT for comparison

ifftDin = &fftin_800hz_verylong16;
DSP_TransformFFT16(ifftDout, ifftDin, ifftc, iscratch, ilog2N);

// ifftDout = frequency domain output, complex number array

DSP_TransformIFFT16(ifftTimeOut, ifftDout, ifftc, iscratch, ilog2N);

// do something with the output, fftTimeOut, time domain
```

5.4.1.5.4.6 DSP_TransformWindow_Bart16 Function**C**

```
void DSP_TransformWindow_Bart16(
    int16_t * OutVector,
    int16_t * InVector,
    int N
);
```

Description

Function DSP_TransformWindow_Bart16:

```
void DSP_TransformWindow_Bart16(int16_t *OutVector, int16_t *InVector, int N);
```

Compute a Bartlett (Triangle) Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Bartlett Window follows the equation:

Window(n) = 1 - (abs(2*n - N)/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int16_t OutVector16[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector16[i]= 0x4000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Bart16(OutVector16, InVector16, WindowN);
// OutWindow = 0x0000, 0x1000, 0x2000, 0x3000, 0x4000, 0x3000, 0x2000, 0x1000
```

5.4.1.5.4.7 DSP_TransformWindow_Bart32 Function**C**

```
void DSP_TransformWindow_Bart32(
    int32_t * OutVector,
    int32_t * InVector,
    int N
);
```

Description

Function DSP_TransformWindow_Bart32:

```
void DSP_TransformWindow_Bart32(int32_t *OutVector, int32_t *InVector, int N);
```

Compute a Bartlett (Triangle) Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Bartlett Window follows the equation:

$Window(n) = 1 - (abs(2*n - N)/N)$ where n is the window sample number N is the total number of samples The functional output computes $WinVector(n) = Window(n) * InVector(n)$

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int32_t OutVector32[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector32[i]= 0x40000000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Bart32(OutVector32, InVector32, WindowN);
// OutWindow = 0x0, 0x10000000, 0x20000000, 0x30000000, 0x40000000,
//              0x30000000, 0x20000000, 0x10000000
```

5.4.1.5.4.8 DSP_TransformWindow_Black16 Function**C**

```
void DSP_TransformWindow_Black16(
    int16_t * OutVector,
    int16_t * InVector,
    int N
);
```

Description

Function DSP_TransformWindow_Black16:

```
void DSP_TransformWindow_Black16(int16_t *OutVector, int16_t *InVector, int N);
```

Compute a Blackman Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Blackman Window follows the equation:

$Window(n) = 0.42659 - 0.49656 * \cos(2\pi n/(N-1)) + 0.076849 * \cos(4\pi n/(N-1))$ where n is the window sample number N is the total number of samples The functional output computes $WinVector(n) = Window(n) * InVector(n)$

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int16_t OutVector16[8]={0};
int WindowN = 8;
```

```

for (i=0;i<WindowN;i++)
{
    InVector16[i]= 0x4000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Black16(OutVector16, InVector16, WindowN);
// OutWindow = 0x0071, 0x0665, 0x1DF1, 0x3B00, 0x3B00, 0x1DF1, 0x0665, 0x0071

```

5.4.1.5.4.9 DSP_TransformWindow_Black32 Function

C

```

void DSP_TransformWindow_Black32(
    int32_t * OutVector,
    int32_t * InVector,
    int N
);

```

Description

Function DSP_TransformWindow_Black32:

```
void DSP_TransformWindow_Black32(int32_t *OutVector, int32_t *InVector, int N);
```

Compute a Blackman Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Blackman Window follows the equation:

Window(n) = 0.42659 - 0.49656 * COS(2*Pi*n/(N-1)) + 0.076849 * COS(4*Pi*n/(N-1)) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```

int32_t OutVector32[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector32[i]= 0x40000000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Black32(OutVector32, InVector32, WindowN);
// OutWindow = 0x0070B490, 0x06649680, 0x1DF13240, 0x3B003D80, 0x3B003D80,
//              0x1DF13240, 0x06649680, 0x0070B490

```

5.4.1.5.4.10 DSP_TransformWindow_Cosine16 Function

C

```
void DSP_TransformWindow_Cosine16(
    int16_t * OutVector,
    int16_t * InVector,
    int N
);
```

Description

Function DSP_TransformWindow_Cosine16:

```
void DSP_TransformWindow_Cosine16(int16_t *OutVector, int16_t *InVector, int N);
```

Compute a Cosine Window on the first N samples of the input vector, InVector. The output is stored in OutWindow. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Cosine Window follows the equation:

Window(n) = SIN(Pi*n/(N-1)) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int16_t OutVector16[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector16[i]= 0x4000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Cosine16(OutVector16, InVector16, WindowN);
// OutWindow = 0x0000, 0x1BC5, 0x320A, 0x3E65, 0x3E65, 0x320A, 0x1BC5, 0x0071
```

5.4.1.5.4.11 DSP_TransformWindow_Cosine32 Function

C

```
void DSP_TransformWindow_Cosine32(
    int32_t * OutVector,
    int32_t * InVector,
    int N
);
```

Description

Function DSP_TransformWindow_Cosine32:

```
void DSP_TransformWindow_Cosine32(int32_t *OutVector, int32_t *InVector, int N);
```

Compute a Cosine Window on the first N samples of the input vector, InVector. The output is stored in OutWindow. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Cosine Window follows the equation:

$Window(n) = \sin(\pi n / (N-1))$ where n is the window sample number N is the total number of samples The functional output computes $WinVector(n) = Window(n) * InVector(n)$

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int32_t OutVector32[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector32[i]= 0x40000000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Cosine32(OutVector32, InVector32, WindowN);
// OutWindow = 0x00000000, 0x1BC4C060, 0x32098700, 0x3E653800, 0x3E653800,
//              0x32098700, 0x1BC4C060, 0x00000000
```

5.4.1.5.4.12 DSP_TransformWindow_Hamm16 Function

C

```
void DSP_TransformWindow_Hamm16(
    int16_t * OutVector,
    int16_t * InVector,
    int N
);
```

Description

Function DSP_TransformWindow_Hamm16:

```
void DSP_TransformWindow_Hamm16(int16_t *OutVector, int16_t *InVector, int N);
```

Compute a Hamming Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Hamming Window follows the equation:

$Window(n) = 0.54 - 0.46 * \cos(2 * \pi * n / N)$ where n is the window sample number N is the total number of samples The functional output computes $WinVector(n) = Window(n) * InVector(n)$

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int16_t OutVector16[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector16[i]= 0x4000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Hamm16(OutVector16, InVector16, WindowN);
// OutWindow = 0x051F, 0x0DBE, 0x228F, 0x3761, 0x4000, 0x3761, 0x228F, 0x0DBE
```

5.4.1.5.4.13 DSP_TransformWindow_Hamm32 Function**C**

```
void DSP_TransformWindow_Hamm32(
    int32_t * OutVector,
    int32_t * InVector,
    int N
);
```

Description

Function DSP_TransformWindow_Hamm32:

```
void DSP_TransformWindow_Hamm32(int32_t *OutVector, int32_t *InVector, int N);
```

Compute a Hamming Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Hamming Window follows the equation:

$Window(n) = 0.54 - 0.46 * \cos(2 * \pi * n / N)$ where n is the window sample number N is the total number of samples The functional output computes $WinVector(n) = Window(n) * InVector(n)$

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int32_t OutVector32[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector32[i]= 0x40000000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Hamm32(OutVector32, InVector32, WindowN);
// OutWindow = 0x051EB860, 0x0DBE26C0, 0x228F5C40, 0x37609200, 0x40000000,
//              0x37609200, 0x228F5C40, 0x0DBE26C0
```

5.4.1.5.4.14 DSP_TransformWindow_Hann16 Function**C**

```
void DSP_TransformWindow_Hann16(
    int16_t * OutVector,
    int16_t * InVector,
    int N
);
```

Description

Function DSP_TransformWindow_Hann16:

```
void DSP_TransformWindow_Hann16(int16_t *OutVector, int16_t *InVector, int N);
```

Compute a Hanning Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Hanning Window follows the equation:

Window(n) = 0.5 - 0.5 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int16_t OutVector16[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector16[i]= 0x4000; // constant 0.5 for functional testing
}
```



```

}

    DSP_TransformWindow_Hann16(OutVector16, InVector16, WindowN);
    // OutWindow = 0x0000, 0x095F, 0x2000, 0x36A1, 0x4000, 0x36A1, 0x2000, 0x095F

```

5.4.1.5.4.15 DSP_TransformWindow_Hann32 Function

C

```

void DSP_TransformWindow_Hann32(
    int32_t * OutVector,
    int32_t * InVector,
    int N
);

```

Description

Function DSP_TransformWindow_Hann32:

```
void DSP_TransformWindow_Hann32(int32_t *OutVector, int32_t *InVector, int N);
```

Compute a Hanning Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Hanning Window follows the equation:

Window(n) = 0.5 - 0.5 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```

int32_t OutVector32[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector32[i]= 0x40000000; // constant 0.5 for functional testing
}

    DSP_TransformWindow_Hann32(OutVector32, InVector32, WindowN);
    // OutWindow = 0x00000000, 0x095F61C0, 0x20000000, 0x36A09E80, 0x40000000,
    //              0x36A09E80, 0x20000000, 0x095F61C0

```

5.4.1.5.4.16 DSP_TransformWindow_Kaiser16 Function

C

```

void DSP_TransformWindow_Kaiser16(
    int16_t * OutVector,
    int16_t * InVector,

```

```
    int N
);
```

Description

Function DSP_TransformWindow_Kaiser16:

```
void DSP_TransformWindow_Kaiser16(int16_t *OutVector, int16_t *InVector, int N);
```

Compute a Kaiser Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Hanning Window follows the equation:

Window(n) = 0.402 - 0.498 * COS(2*Pi*n/N) + 0.098 * cos(4*Pi*n/N) + 0.001 * cos(6*Pi*n/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int16_t OutVector16[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector16[i]= 0x4000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Kaiser16(OutVector16, InVector16, WindowN);
// OutWindow = 0x0031, 0x0325, 0x1375, 0x304F, 0x3FCF, 0x304F, 0x1375, 0x0325
```

5.4.1.5.4.17 DSP_TransformWindow_Kaiser32 Function

C

```
void DSP_TransformWindow_Kaiser32(
    int32_t * OutVector,
    int32_t * InVector,
    int N
);
```

Description

Function DSP_TransformWindow_Kaiser32:

```
void DSP_TransformWindow_Kaiser32(int32_t *OutVector, int32_t *InVector, int N);
```

Compute a Kaiser Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Hanning Window follows the equation:

$\text{Window}(n) = 0.402 - 0.498 * \cos(2\pi n/N) + 0.098 * \cos(4\pi n/N) + 0.001 * \cos(6\pi n/N)$ where n is the window sample number N is the total number of samples The functional output computes $\text{WinVector}(n) = \text{Window}(n) * \text{InVector}(n)$

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Example

```
int32_t OutVector32[8]={0};
int WindowN = 8;

for (i=0;i<WindowN;i++)
{
    InVector32[i]= 0x40000000; // constant 0.5 for functional testing
}

DSP_TransformWindow_Kaiser32(OutVector32, InVector32, WindowN);
// OutWindow = 0x003126F6, 0x032555C8, 0x1374BCA0, 0x304F66C0, 0x3FCED900,
//              0x304F66C0, 0x1374BCA0, 0x032555C8
```

5.4.1.5.4.18 DSP_TransformWinInit_Bart16 Function

C

```
void DSP_TransformWinInit_Bart16(
    int16_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Bart16:

```
void DSP_TransformWinInit_Bart16(int16_t *OutWindow, int N);
```

Create a N -element Bartlett (Triangle) Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Bartlett Window follows the equation:

$\text{Window}(n) = 1 - (\text{abs}(2*n - N)/N)$ where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example**5.4.1.5.4.19 DSP_TransformWinInit_Bart32 Function****C**

```
void DSP_TransformWinInit_Bart32(
    int32_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Bart32:

```
void DSP_TransformWinInit_Bart32(int32_t *OutWindow, int N);
```

Create a N-element Bartlett (Triangle) Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values Q31 fractional format. The Bartlett Window follows the equation:

Window(n) = 1 - (abs(2*n - N)/N) where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example**5.4.1.5.4.20 DSP_TransformWinInit_Black16 Function****C**

```
void DSP_TransformWinInit_Black16(
```

```

    int16_t * OutWindow,
    int N
);

```

Description

Function DSP_TransformWinInit_Black16:

```
void DSP_TransformWinInit_Black16(int16_t *OutWindow, int N);
```

Create a N-element Blackman Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are Q16 fractional format. The Blackman Window follows the equation:

Window(n) = 0.42659 - 0.49656 * COS(2*Pi*n/(N-1)) + 0.076849 * COS(4*Pi*n/(N-1)) where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example

5.4.1.5.4.21 DSP_TransformWinInit_Black32 Function

C

```

void DSP_TransformWinInit_Black32(
    int32_t * OutWindow,
    int N
);

```

Description

Function DSP_TransformWinInit_Black32:

```
void DSP_TransformWinInit_Black32(int32_t *OutWindow, int N);
```

Create a N-element Blackman Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are Q31 fractional format. The Blackman Window follows the equation:

Window(n) = 0.42659 - 0.49656 * COS(2*Pi*n/(N-1)) + 0.076849 * COS(4*Pi*n/(N-1)) where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example**5.4.1.5.4.22 DSP_TransformWinInit_Cosine16 Function****C**

```
void DSP_TransformWinInit_Cosine16(
    int16_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Cosine16:

```
void DSP_TransformWinInit_Cosine16(int16_t *OutWindow, int N);
```

Create a N-element Cosine Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Cosine Window follows the equation:

Window(n) = SIN(Pi*n/(N-1)) where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done

during recurring loop processing.

Example

5.4.1.5.4.23 DSP_TransformWinInit_Cosine32 Function

C

```
void DSP_TransformWinInit_Cosine32(
    int32_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Cosine32:

```
void DSP_TransformWinInit_Cosine32(int32_t *OutWindow, int N);
```

Create a N-element Cosine Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q31 fractional format. The Cosine Window follows the equation:

Window(n) = SIN(Pi*n/(N-1)) where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example

5.4.1.5.4.24 DSP_TransformWinInit_Hamm16 Function

C

```
void DSP_TransformWinInit_Hamm16(
    int16_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Hamm16:

```
void DSP_TransformWinInit_Hamm16(int16_t *OutWindow, int N);
```

Create a N-element Hamming Window, and store the output to OutWindow. Operations are performed at higher resolution

floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Hamming Window follows the equation:

$\text{Window}(n) = 0.54 - 0.46 * \cos(2\pi n/N)$ where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example

5.4.1.5.4.25 DSP_TransformWinInit_Hamm32 Function

C

```
void DSP_TransformWinInit_Hamm32(
    int32_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Hamm32:

```
void DSP_TransformWinInit_Hamm32(int32_t *OutWindow, int N);
```

Create a N -element Hamming Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q31 fractional format. The Hamming Window follows the equation:

$\text{Window}(n) = 0.54 - 0.46 * \cos(2\pi n/N)$ where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example**5.4.1.5.4.26 DSP_TransformWinInit_Hann16 Function****C**

```
void DSP_TransformWinInit_Hann16(
    int16_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Hann16:

```
void DSP_TransformWinInit_Hann16(int16_t *OutWindow, int N);
```

Create a N-element Hanning Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Hanning Window follows the equation:

Window(n) = 0.5 - 0.5 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example**5.4.1.5.4.27 DSP_TransformWinInit_Hann32 Function****C**

```
void DSP_TransformWinInit_Hann32(
    int32_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Hann32:

```
void DSP_TransformWinInit_Hann32(int32_t *OutWindow, int N);
```

Create a N-element Hanning Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q31 fractional format. The Hanning Window follows the equation:

Window(n) = 0.5 - 0.5 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example**5.4.1.5.4.28 DSP_TransformWinInit_Kaiser16 Function****C**

```
void DSP_TransformWinInit_Kaiser16(
    int16_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Kaiser16:

```
void DSP_TransformWinInit_Kaiser16(int16_t *OutWindow, int N);
```

Create a N-element Kaiser Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Kaiser Window follows the equation:

Window(n) = 0.402 - 0.498 * COS(2*Pi*n/N) + 0.098 * cos(4*Pi*n/N) + 0.001 * cos(6*Pi*n/N) where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example**5.4.1.5.4.29 DSP_TransformWinInit_Kaiser32 Function****C**

```
void DSP_TransformWinInit_Kaiser32(
    int32_t * OutWindow,
    int N
);
```

Description

Function DSP_TransformWinInit_Kaiser32:

```
void DSP_TransformWinInit_Kaiser32(int32_t *OutWindow, int N);
```

Create a N-element Kaiser Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q31 fractional format. The Kaiser Window follows the equation:

Window(n) = 0.402 - 0.498 * COS(2*Pi*n/N) + 0.098 * cos(4*Pi*n/N) + 0.001 * cos(6*Pi*n/N) where n is the window sample number N is the total number of samples

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Example**5.4.1.5.5 Vector Math Functions****5.4.1.5.5.1 DSP_VectorAbs16 Function****C**

```
void DSP_VectorAbs16(
    int16_t * outdata,
    int16_t * indata,
    int N
);
```

Description

Function DSP_VectorAbs16:

```
void DSP_VectorAbs16(int16_t *outdata, int16_t *indata, int N);
```

Computes the absolute value of each element of indata and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q15 fractional format. outdata[i] filled with the absolute value of elements of indata

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufTest[16] = {-5, 2, -3, 4, -1, 0, -2, -8, -21, 21, 10, 100, 200, 127, -127, -2};
int Num = 8;

pOutdata = &outVal;

DSP_VectorAbs16(pOutdata, inBufTest, Num);

// outVal[i] = {5, 2, 3, 4, 1, 0, 2, 8}
```

5.4.1.5.5.2 DSP_VectorAbs32 Function**C**

```
void DSP_VectorAbs32(
    int32_t * outdata,
    int32_t * indata,
```

```
    int N
);
```

Description

Function DSP_VectorAbs32:

```
void DSP_VectorAbs32(int32_t *outdata, int32_t *indata, int N);
```

Computes the absolute value of each element of indata and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of abs(indata[i])

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata	pointer to input array of 16-bit elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```
int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufTest[16] = {-5, 2, -3, 4, -1, 0, -2, -8, -21, 21, 10, 100, 200, 127, -127, -2};
int Num = 8;

pOutdata = &outVal;

DSP_VectorAbs32(pOutdata, inBufTest, Num);

// outVal[i] = {5, 2, 3, 4, 1, 0, 2, 8}
```

5.4.1.5.5.3 DSP_VectorAdd16 Function

C

```
void DSP_VectorAdd16(
    int16_t * outdata,
    int16_t * indata1,
    int16_t * indata2,
    int N
);
```

Description

Function DSP_VectorAdd16:

```
void DSP_VectorAdd16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);
```

Computes the sum value of each element of indata1 + indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in the Q15 fractional format. outdata[i] filled with N elements of indata1[i] + indata2[i]

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of

eight.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata1	pointer to input array of 16-bit elements (int16_t)
indata2	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
int16_t inBuf2[16]=    { 1,2, 3,4, 5,6, 7, 8, 9, 10,-1,-100,-127,127,-7, 0};
int Num = 8;

pOutdata = &outVal;

DSP_VectorAdd16(pOutdata, inBufTest, inBuf2, Num);

// outVal[i] = inBufTest[i] + inBuf2[i] = {-4,4,0,8,4,6,5,0}
```

5.4.1.5.5.4 DSP_VectorAdd32 Function

C

```
void DSP_VectorAdd32(
    int32_t * outdata,
    int32_t * indata1,
    int32_t * indata2,
    int N
);
```

Description

Function DSP_VectorAdd32:

```
void DSP_VectorAdd32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);
```

Computes the sum value of each element of indata1 + indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of indata1[i] + indata2[i]

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata1	pointer to input array of 16-bit elements (int32_t)
indata2	pointer to input array of 16-bit elements (int32_t)
N	number of samples

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```
int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
int32_t inBuf2[16]=    { 1,2, 3,4, 5,6, 7, 8, 9, 10,-1,-100,-127,127,-7, 0};
int Num = 8;

pOutdata = &outVal;

DSP_VectorAdd32(pOutdata, inBufTest, inBuf2, Num);

// outVal[i] = inBufTest[i] + inBuf2[i] = {-4,4,0,8,4,6,5,0}
```

5.4.1.5.5.5 DSP_VectorAddc16 Function**C**

```
void DSP_VectorAddc16(
    int16_t * outdata,
    int16_t * indata,
    int16_t c,
    int N
);
```

Description

Function DSP_VectorAddc16:

```
void DSP_VectorAddc16(int16_t *outdata, int16_t *indata, int16_t c, int N);
```

Computes the sum value of each element of (indata + c) and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q15 fractional format. outdata[i] filled with N elements of indata[i] + c

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata	pointer to input array of 16-bit elements (int16_t)
c	constant value added to all indata1 elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
```

```

int16_t inBufTest[16] = {-5, 2, -3, 4, -1, 0, -2, -8, -21, 21, 10, 100, 200, 127, -127, -2};
int16_t constValue = 3;
int Num = 8;

pOutdata = &outVal;

DSP_VectorAddc16(pOutdata, inBufTest, constValue, Num);

// outVal[i] = inBufTest[i] + constValue = {-2, 5, 0, 7, 2, 3, 1, -5}

```

5.4.1.5.5.6 DSP_VectorAddc32 Function

C

```

void DSP_VectorAddc32(
    int32_t * outdata,
    int32_t * indata,
    int32_t c,
    int N
);

```

Description

Function DSP_VectorAddc32:

```
void DSP_VectorAddc32(int32_t *outdata, int32_t *indata, int32_t c, int N);
```

Computes the sum value of each element of (indata + c) and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of indata1[i] + c

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata	pointer to input array of 16-bit elements (int32_t)
c	constant value added to all indata1 elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```

int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufTest[16] = {-5, 2, -3, 4, -1, 0, -2, -8, -21, 21, 10, 100, 200, 127, -127, -2};
int32_t constValue = 3;
int Num = 8;

pOutdata = &outVal;

DSP_VectorAddc32(pOutdata, inBufTest, constValue, Num);

// outVal[i] = inBufTest[i] + constValue = {-2, 5, 0, 7, 2, 3, 1, -5}

```


5.4.1.5.5.7 DSP_VectorAutocorr16 Function

C

```
void DSP_VectorAutocorr16(
    int16_t * outCorr,
    int16_t * inVector,
    int N,
    int K
);
```

Description

Function DSP_VectorAutocorr16:

```
void DSP_VectorAutocorr16(int16_t *outCorr, int16_t *inVector, int N, int K);
```

Calculates the autocorrelation, with a lag of 1 to K, on the first N elements of inVector and returns the 16-bit scalar result in outCorr. The autocorrelation is calculated from other statistical equations including mean and variance. While in some cases these equations exist inside the DSP library, the functions are executed in a serial fashion within this code to provide enhanced performance. The unbiased function has the form -

mean (M) = $\text{sum}[0..N-1](x(n) / N)$ variance (V) = $\text{sum}[0..N-1]((x(n) - M)^2) / (N-1)$ autocovariance (ACV)[k] = $\text{sum}[0..(N-k)]((x(n) - M) * (x(n+k) - M) / (N-k))$ autocorrelation (AC)[k] = CV[k] / V where N is the number of vector elements, n is the index of those elements x(n) is a single element in the input vector M is the mean of the N elements of the vector k is the lag or series index

The output of the function will return K elements, and the outCorr array should be sized to [accept](#) those 16-bit results. The outputs correspond to k=1, k=2, ..., k=K delay states. The function returns a 16-bit value in rounded, saturated Q15 format.

Input values of the vector and output scalar value is Q15 fractional format. This format has data that ranges from -1 to 1, and has internal saturation limits of those same values. Some care has been taken to reduce the impact of saturation by adding processing steps to effectively complete the processing in blocks. However, in some extreme cases of data variance it is still possible to reach the saturation limits.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. outCorr must be an array with at least K values. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
outCorr	pointer to output array (int16_t)
inVector	pointer to source array of elements (int16_t)
N	number of samples (int)
K	lag value, number of output elements (int)

Returns

None.

Remarks

This function is optimized with microMIPS and M14KCe ASE DSP instructions. This function is dependent on the LIBQ library, and the [_LIBQ_Q16Div](#) specifically.

Example

```
int autocvN = 16; // N value, number of samples in array
int autocvLag = 4; // Lag value, output shifts to observe
int16_t outAC[16]={0};
int16_t CVIn16[16] = {0x1999, 0xD99A, 0x1000, 0x6000, 0x1999, 0x1999, 0x2666,
    0x3333, 0x1000, 0x6000, 0x1999, 0x1999, 0x2666, 0x3333, 0x1999, 0x0CCC};
```

```
// = { .2, -.3, .125, .75, .2, .2, .3, .4, .125, .75, .2, .2, .3, .4, .2, .1};
DSP_VectorAutocorr16(outAC, CVIn16, autocvN, autocvLag);
// outAC = {0xF406, D46C, 0x098F, 0x191A}
//      = -0.093567, -0.34045, 0.07468, 0.19611
```

5.4.1.5.5.8 DSP_VectorBexp16 Function

C

```
int DSP_VectorBexp16(
    int16_t * DataIn,
    int N
);
```

Description

Function DSP_VectorBexp16:

```
int DSP_VectorBexp16(int16_t *DataIn, int N);
```

Calculates the maximum binary exponent on the first N elements of the input vector DataIn, and stores the integer result. The returned value represents the potential binary scaling of the vector, and may be used with other functions that auto scale their output without saturation. Inputs are given in Q15 fractional data format.

Preconditions

N must be a multiple of 2 and greater or equal to 2.

Parameters

Parameters	Description
DataIn	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

Returns

Binary exponent [log2 multiplier] (int)

Remarks

None.

Example

```
int valN = 4;
int16_t dummy16[valN]={0x3004, 0x00CC, 0xFC04, 0xFFFF};
//                      0.375, 0.0062, -0.0311, -0.00049
int answer;

answer = DSP_VectorBexp16(dummy16, valN);

// answer = 1, maximum binary gain is 2.
```

5.4.1.5.5.9 DSP_VectorBexp32 Function

C

```
int DSP_VectorBexp32(
    int32_t * DataIn,
    int N
);
```

Description

Function DSP_VectorBexp32:

```
int DSP_VectorBexp32(int32_t *DataIn, int N);
```

Calculates the maximum binary exponent on the first N elements of the input vector DataIn, and stores the integer result. The returned value represents the potential binary scaling of the vector, and may be used with other functions that auto scale their output without saturation. Inputs are given in Q31 fractional data format.

Preconditions

None.

Parameters

Parameters	Description
DataIn	pointer to input array of 16-bit elements (int32_t)
N	number of samples (int)

Returns

Binary exponent [log2 multiplier] (int)

Remarks

None.

Example

```
int valN=4;
int32_t datInput32[4]={0xFF000000, 0x07000000,0x000CCCCC, 0x08000000};
//                                -0.007183, 0.054688, 0.0003906, 0.0625
int answer32;

answer32 = DSP_VectorBexp32(datInput32, valN);

// answer = 3, maximum binary gain is 8.
```

5.4.1.5.5.10 DSP_VectorChkEqu32 Function

C

```
int DSP_VectorChkEqu32(
    int32_t * indata1,
    int32_t * indata2,
    int N
);
```

Description

Function DSP_VectorChkEqu32:

```
int DSP_VectorChkEqu32(int32_t *indata1, int32_t *indata2, int N);
```

Compares the first N values of indata1 to the same elements of indata2. The comparison requires that all numbers be in Q31 fractional data format. Returns the integer value '1' if all numbers are equal, and '0' if they are not equal. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
indata1	pointer to input array 1 of elements (int32_t)
indata2	pointer to input array 2 of elements (int32_t)

N	number of samples (int)

Returns

(int) - '1' if vectors are equal, '0' if vectors are not equal

Remarks

None.

Example

```
int outCheck;
int Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//          1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int32_t inBufTestB[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x80000000, 0x40000000, 0x7FFFFFFF, 0x20000000};
//          1,      -1,      0.9,      0.8,      -1,      0.5,      1,      0.25

outCheck = DSP_VectorChkEqu32(inBufTestA, inBufTestB, Num);

// outCheck = 1 // true for first 4 numbers of series
```

5.4.1.5.5.11 DSP_VectorCopy Function

C

```
void DSP_VectorCopy(
    int32_t * outdata,
    int32_t * indata,
    int N
);
```

Description

Function DSP_VectorCopy:

void DSP_VectorCopy(int32_t *outdata, int32_t *indata, int N);

Fills the first N values of an input vector outdata with the the elements from indata. N must be a multiple of four and greater than or equal to four or it will be truncated to the nearest multiple of four. The vector result and the scalar value to fill are both Q31 fractional format.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
outdata	pointer to destination array of values (int32_t)
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

None.

Example

```

int Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//           1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int32_t inBufTestB[8]={0x00000000, 0x7FFFFFFF, 0x40000000, 0x0CCCCCCC,
                       0x40000000, 0x60000000, 0x80000000, 0x20000000};
//           0,      1,      0.5,      0.1,      0.75,      0.5,      -1,      0.25

DSP_VectorCopy(inBufTestA, inBufTestB, Num);

// inBufTestA = {0x00000000, 0x7FFFFFFF, 0x40000000, 0x0CCCCCCC,
//               0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334} // first 4 values copied

```

5.4.1.5.5.12 DSP_VectorCopyReverse32 Function**C**

```

void DSP_VectorCopyReverse32(
    int32_t * outdata,
    int32_t * indata,
    int N
);

```

Description

Function DSP_VectorCopyReverse32:

```
void DSP_VectorCopyReverse32(int32_t *outdata, int32_t *indata, int N);
```

Fills the first N values of an input vector Outdata with the the reverse elements from INDATA. N must be a multiple of 4 and greater than 4 or will be truncated to the nearest multiple of 4. The vectors are both Q31 fractional format.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
outdata	pointer to destination array of values (int32_t)
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

None.

Example

```

int Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//           1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int32_t inBufTestB[8]={0x00000000, 0x7FFFFFFF, 0x40000000, 0x0CCCCCCC,
                       0x40000000, 0x60000000, 0x80000000, 0x20000000};
//           0,      1,      0.5,      0.1,      0.75,      0.5,      -1,      0.25

DSP_VectorCopyReverse32(inBufTestA, inBufTestB, Num);

// inBufTestA = {0x0CCCCCCC, 0x40000000, 0x7FFFFFFF, 0x00000000,

```

```
//      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334}
// first 4 values copied reverse order
```

5.4.1.5.5.13 DSP_VectorDivC Function

C

```
void DSP_VectorDivC(
    _Q16 * outVector,
    _Q16 * inVector,
    _Q16 divisor,
    int N
);
```

Description

Function DSP_VectorDivC:

```
void DSP_VectorDivC(_Q16 *outVector, _Q16 *inVector, _Q16 divisor, int N);
```

Divides each element of the first N elements of inVector by a constant, divisor. The output is stored to outVector. Both vectors and the scalar are [_Q16](#) format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. If values exceed maximum or minimum they will saturate to the maximum or minimum respectively.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Div function from the LIBQ library. That library must be compiled as part of the project.

Parameters

Parameters	Description
outdata	pointer to desination array of elements (_Q16)
indata	pointer to source array of elements (_Q16)
divisor	scalar divisor for the input vector (_Q16)
N	number of samples (int)

Returns

None.

Remarks

This function uses the Microchip PIC32MZ LIBQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Div function see the LIBQ documentation for [LIBQ_Q16Div](#).

Example

```
int divNum = 4;
_Q16 divScalar = 0x00020000; // 2.0
_Q16 inDivVec[8] = {0x08000000, 0xfffc0000, 0x00024000, 0x00100000, 0x00038000,
    0x00400000, 0xfffe0000, 0x00058000};
//      2048, -4, 2.25, 16, 3.5, 64, -2, 5.5
_Q16 outDivVec[8] = {0};

DSP_VectorDivC(outDivVec, inDivVec, divScalar, divNum);

// outDivVec = 0x04000000, 0xFFFFE0000, 0x00012000, 0x00080000, 0, 0, 0, 0
//              1024.0,      -2.0,      1.125,      16.0, 0, 0, 0, 0
```

5.4.1.5.5.14 DSP_VectorDotp16 Function

C

```
int16_t DSP_VectorDotp16(
    int16_t * indata1,
```

```

    int16_t * indata2,
    int N,
    int scale
);

```

Description

Function DSP_VectorDotp16:

```
int16_t DSP_VectorDotp16(int16_t *indata1, int16_t *indata2, int N, int scale);
```

Calculates the dot product of two input vectors, and scales the output. Function will saturate if it exceeds maximum or minimum values. Scaling is done by binary shifting, after accumulation in a 32 bit register. All calculations are done in Q15 fractional format. return = $1/(2^{\text{scale}}) * \text{sum}(\text{indata1}[i] * \text{indata2}[i])$

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Parameters

Parameters	Description
indata1	pointer to input array of 16-bit elements (int16_t)
indata2	pointer to input array of 16-bit elements (int16_t)
scale	number of bits to shift return right (int)
N	number of samples (int)

Returns

int16_t - scaled output of calculation, Q15 format

Remarks

This must be assembled with .set microMIPS.

Example

```

int16_t inBufMultA[8]={0x7FFF, 0x8000, 0x7333, 0x6666, 0x1999, 0x4000, 0x7FFF, 0xB334};
//      1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int16_t inBufMultB[8]={0x0CCD, 0x0CCD, 0x4000, 0xC000, 0xE667, 0x4000, 0x0000, 0x0CCD};
//      0.1,      0.1,      0.5,      -0.5,      -0.2,      0.5,      0,      0.1
int Num = 8;
int scaleVal = 2;
int16_t outScalar;

int Num = 8;

outScalar = DSP_VectorDotp16(inBufMultA, inBufMultB, Num, scaleVal);

// outScalar = 1/(2^scaleVal)*(inBufMultA[] dot inBufMultB[]) =
// (1/4) * (0.1 + -0.1 + 0.45 + -0.4 + -0.04 + 0.25 + 0 + -0.06) = 0.25 * 0.20 = 0.05
// = (int16_t)0x0666

```

5.4.1.5.5.15 DSP_VectorDotp32 Function

C

```

int32_t DSP_VectorDotp32(
    int32_t * indata1,
    int32_t * indata2,
    int N,
    int scale
);

```

Description

Function DSP_VectorDotp32:

```
int32_t DSP_VectorDotp32(int32_t *indata1, int32_t *indata2, int N, int scale);
```

Calculates the dot product of two input vectors, and scales the output. Function will saturate if it exceeds maximum or minimum values. Scaling is done by binary shifting, after calculation of the result. All calculations are done in Q31 fractional format. return = $1/(2^{\text{scale}}) * \text{sum}(\text{indata1}[i] * \text{indata2}[i])$

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
indata1	pointer to input array of 16-bit elements (int32_t)
indata2	pointer to input array of 16-bit elements (int32_t)
scale	number of bits to shift return right (int)
N	number of samples (int)

Returns

int16_t - scaled output of calculation, Q31 format

Remarks

This must be assembled with .set microMIPS.

Example

```
int32_t inBufMultA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//          1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int32_t inBufMultB[8]={0x0CCCCCD, 0x0CCCCCD, 0x40000000, 0xC0000000,
                      0xE6666667, 0x40000000, 0x00000000, 0x0CCCCCD};
//          0.1,      0.1,      0.5,     -0.5,     -0.2,      0.5,      0,      0.1
int Num = 8;
int scaleVal = 2;
int32_t outScalar;

int Num = 8;

outScalar = DSP_VectorDotp32(inBufMultA, inBufMultB, Num, scaleVal);

// outScalar = 1/(2^scaleVal)*(inBufMultA[] dot inBufMultB[]) =
// (1/4) * (0.1 + -0.1 + 0.45 + -0.4 + -0.04 + 0.25 + 0 + -0.06) = 0.25 * 0.20 = 0.05
// = (int32_t)0x06666666
```

5.4.1.5.5.16 DSP_VectorExp Function**C**

```
void DSP_VectorExp(
    _Q16 * outVector,
    _Q16 * inVector,
    int N
);
```

Description

Function DSP_VectorExp:

```
void DSP_VectorExp(_Q16 *outVector, _Q16 *inVector, int N);
```


Computes the Exp value, e to the power of X , on the first N elements of `inVector`. The output is stored to `outVector`. Both vectors are `_Q16` format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Preconditions

The pointers `outdata` and `indata` must be aligned on 4-byte boundaries. This function uses the Exp and Div functions from the LIBQ library. That library must be compiled as part of the project.

Parameters

Parameters	Description
<code>outdata</code>	pointer to desination array of elements (<code>_Q16</code>)
<code>indata</code>	pointer to source array of elements (<code>_Q16</code>)
<code>N</code>	number of samples (int)

Returns

None.

Remarks

Inclusion of the LIBQ header file and library is mandatory to use this function.

Example

```
int expNum = 4;
_Q16 inExpVec[8] = {0x00010000, 0xffff0000, 0x00020000, 0x00030000, 0x00038000,
                    0x00040000, 0xfffe0000, 0x00058000};
//          1.0,  -1.0,  2.0,  3.0,   3.5,  4.0,  -2.0,  5.5
_Q16 outExpVec[8] = {0};

DSP_VectorExp(outExpVec, inExpVec, expNum);

// outExpVec = 0x0002B7E1, 0x00005E2D, 0x00076399, 0x001415E6, 0, 0, 0, 0
//          2.71828,  0.26787,   7.3891,  20.0855, 0, 0, 0, 0
```

5.4.1.5.5.17 DSP_VectorFill Function

C

```
void DSP_VectorFill(
    int32_t * indata,
    int32_t data,
    int N
);
```

Description

Function `DSP_VectorFill`:

```
void DSP_VectorFill(int32_t *indata, int32_t data, int N);
```

Fills the first N values of an input vector `indata` with the value `data`. N must be a multiple of four and greater than or equal to four or it will be truncated to the nearest multiple of four. The vector result and the scalar value to fill are both Q31 fractional format.

Preconditions

The pointers `outdata` and `indata` must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
<code>indata</code>	pointer to source array of elements (int32_t)

data	scalar value to fill the array (int32_t)
N	number of samples (int)

Returns

None.

Remarks

None.

Example

```
int32_t fillValue = 0x3FFFFFFF;
int Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//          1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6

DSP_VectorFill(inBufTestA, fillValue, Num);

// inBufTestA = {0x3FFFFFFF, 0x3FFFFFFF, 0x3FFFFFFF, 0x3FFFFFFF,
//               0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334} // first 4 values filled
```

5.4.1.5.5.18 DSP_VectorLn Function**C**

```
void DSP_VectorLn(
    _Q4_11 * outVector,
    _Q16 * inVector,
    int N
);
```

Description

Function DSP_VectorLn:

```
void DSP_VectorLn(_Q4_11 *outVector, _Q16 *inVector, int N);
```

Computes the Ln(x) value, on the first N elements of inVector. The output is stored to outVector. Input vector is Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. The output vector is reduced resolution Q4.11 format, which is a 16-bit integer format with 11 bits representing the fractional resolution. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Ln function from the LIBQ library. That library must be compiled as part of the project.

Parameters

Parameters	Description
outdata	pointer to desination array of elements (<u>Q16</u>)
indata	pointer to source array of elements (<u>Q4_11</u>)
N	number of samples (int)

Returns

None.

Remarks

This function uses the Microchip PIC32MZ LIBQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Ln function see the LIBQ documentation for

[_LIBQ_Q4_11_In_Q16](#). A negative number input will return a saturated negative value (0x8000).

Example

```
int lnNum = 4;
_Q16 inLnVal[8] = {0x40000000, 0xffff0000, 0x00020000, 0x00100000, 0x00038000,
                  0x00400000, 0xfffe0000, 0x00058000};
//      16384.0,  -1.0,  2.0,  16.0,   3.5,   64.0,  -2.0,  5.5
_Q4_11 outLnVal[8] = {0};

DSP_VectorLn(outLnVal, inLnVal, lnNum);

// outLnVal =      0x4DA2,      0x8000,   0x058C,   0x162E,  0,  0,  0,  0
//              9.704,  sat negative,  0.6934,   2.772,  0,  0,  0,  0
```

5.4.1.5.5.19 DSP_VectorLog10 Function

C

```
void DSP_VectorLog10(
    _Q3_12 * outVector,
    _Q16 * inVector,
    int N
);
```

Description

Function DSP_VectorLog10:

```
void DSP_VectorLog10(_Q3_12 *outVector, _Q16 *inVector, int N);
```

Computes the Log10(x) value, on the first N elements of inVector. The output is stored to outVector. Input vector is [_Q16](#) format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. The output vector is reduced resolution Q3.12 format, which is a 16-bit integer format with 12 bits representing the fractional resolution. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Log10 function from the LIBQ library. That library must be compiled as part of the project.

Parameters

Parameters	Description
outdata	pointer to desination array of elements (_Q16)
indata	pointer to source array of elements (_Q3_12)
N	number of samples (int)

Returns

None.

Remarks

This function uses the Microchip PIC32MZ LIBQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Log10 function see the LIBQ documentation for [_LIBQ_Q3_12_log10_Q16](#). A negative number input will return a saturated negative value (0x8000).

Example

```
int logNum = 4;
_Q16 inLogVal[8] = {0x40000000, 0xffff0000, 0x00020000, 0x00100000, 0x00038000,
                  0x00400000, 0xfffe0000, 0x00058000};
//      16384.0,  -1.0,  2.0,  16.0,   3.5,   64.0,  -2.0,  5.5
_Q3_12 outLogVal[8] = {0};
```

```
DSP_VectorLog10(outLogVal, inLogVal, logNum);

// outLogVal =      0x436E,      0x8000,      0x04D1,      0x1344, 0, 0, 0, 0
//              4.2144,  sat negative,  0.3010,      1.2041, 0, 0, 0, 0
```

5.4.1.5.5.20 DSP_VectorLog2 Function

C

```
void DSP_VectorLog2(
    _Q5_10 * outVector,
    _Q16 * inVector,
    int N
);
```

Description

Function DSP_VectorLog2:

```
void DSP_VectorLog2(_Q5_10 *outVector, _Q16 *inVector, int N);
```

Computes the Log2 value, where $\log_2(x) = \ln(x) * \log_2(e)$, on the first N elements of inVector. The output is stored to outVector. Input vector is Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. The output vector is reduced resolution Q5.10 format, which is a 16-bit integer format with 10 bits representing the fractional resolution. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Log2 function from the LIBQ library. That library must be compiled as part of the project.

Parameters

Parameters	Description
outdata	pointer to desination array of elements (<u>Q16</u>)
indata	pointer to source array of elements (<u>Q5_10</u>)
N	number of samples (int)

Returns

None.

Remarks

This function uses the Microchip PIC32MZ LIBQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Log2 function see the LIBQ documentation for LIBQ_Q5_10_log2_Q16. A negative number input will return a saturated negative value (0x8000).

Example

```
int log2Num = 4;
_Q16 inLog2Val[8] = {0x40000000, 0xffff0000, 0x00020000, 0x00030000, 0x00038000,
                    0x00040000, 0xfffe0000, 0x00058000};
//      16384.0,  -1.0,   2.0,   3.0,   3.5,   4.0,  -2.0,   5.5
_Q5_10 outLog2Val[8] = {0};

DSP_VectorLog2(outLog2Val, inLog2Val, log2Num);

// outLog2Val =      0x3800,      0x8000,      0x0400,      0x0657, 0, 0, 0, 0
//              14.0,  sat negative,      1.0,      1.585, 0, 0, 0, 0
```

5.4.1.5.5.21 DSP_VectorMax32 Function

C

```
int32_t DSP_VectorMax32(
```

```

    int32_t * indata,
    int N
);

```

Description

Function DSP_VectorMax32:

```
int32_t DSP_VectorMax32(int32_t *indata, int N);
```

Returns the highest value of the first N elements of the vector indata. The comparison requires that all numbers be in Q31 fractional data format. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and multiple of four.

Parameters

Parameters	Description
indata	pointer to input array of elements (int32_t)
N	number of samples (int)

Returns

(int32_t) - maximum value within the vector, Q31 format

Remarks

None.

Example

```

int32_t outCheck;
int Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//          1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6

outCheck = DSP_VectorMax32(inBufTestA, Num);

// outCheck = 0x7FFFFFFF // first 4 values

```

5.4.1.5.5.22 DSP_VectorMaxIndex32 Function

C

```

int DSP_VectorMaxIndex32(
    int32_t * indata,
    int N
);

```

Description

Function DSP_VectorMaxIndex32:

```
int DSP_VectorMaxIndex32(int32_t *indata, int N);
```

Returns the index of the highest value of the first N elements of the vector indata. The comparison requires that all numbers be in Q31 fractional data format. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of

four.

Parameters

Parameters	Description
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

Returns

int - index of the position of the maximum array element

Remarks

Index values range from 0 .. (n-1).

Example

```
int    indexValue;
int    Num = 8;
int32_t inBufTestA[8]={0x3FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//           0.5,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6

indexValue = DSP_VectorMaxIndex32(inBufTestA, Num);

// returnValue = 6 (position corresponding to 0x7FFFFFFF)
```

5.4.1.5.5.23 DSP_VectorMean32 Function

C

```
int32_t DSP_VectorMean32(
    int32_t * indata1,
    int N
);
```

Description

Function DSP_VectorMean32:

```
int32_t DSP_VectorMean32(int32_t *indata, int N);
```

Calculates the mean average of the first N elements of the vector indata. The values of indata1 are in Q31 fractional format. The value N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

Returns

int32_t - mean average value of the vector

Remarks

None.

Example

```
int32_t    returnValue;
```

```

int      Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//           1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6

returnValue = DSP_VectorMean32(inBufTestA, Num);

// returnValue = 0x36666666 = (1-1+0.9+0.8)/4 = 0.425

```

5.4.1.5.5.24 DSP_VectorMin32 Function

C

```

int32_t DSP_VectorMin32(
    int32_t * input,
    int N
);

```

Description

Function DSP_VectorMin32:

```
int32_t DSP_VectorMin32(int32_t *indata, int N);
```

Returns the lowest value of the first N elements of the vector indata. The comparison requires that all numbers be in Q31 fractional data format. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and multiple of four.

Parameters

Parameters	Description
indata	pointer to input array of elements (int32_t)
N	number of samples (int)

Returns

(int32_t) - minimum value within the vector, Q31 format

Remarks

None.

Example

```

int32_t outCheck;
int      Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//           1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6

outCheck = DSP_VectorMin32(inBufTestA, Num);

// outCheck = 0x80000000 // first 4 values

```

5.4.1.5.5.25 DSP_VectorMinIndex32 Function

C

```

int DSP_VectorMinIndex32(
    int32_t * indata,
    int N
);

```

Description

Function DSP_VectorMinIndex32:

```
int DSP_VectorMinIndex32(int32_t *indata, int N);
```

Returns the relative position index of the lowest value of the first N elements of the vector indata. The comparison requires that all numbers be in Q31 fractional data format. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

Returns

int32_t - mean average value of the vector

Remarks

Index values range from 0 .. (n-1).

Example

```
int    indexValue;
int    Num = 8;
int32_t inBufTestA[8]={0x3FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//           0.5,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6

indexValue = DSP_VectorMinIndex32(inBufTestA, Num);

// return value = 1 (position corresponding to 0x80000000)
```

5.4.1.5.5.26 DSP_VectorMul16 Function**C**

```
void DSP_VectorMul16(
    int16_t * outdata,
    int16_t * indata1,
    int16_t * indata2,
    int N
);
```

Description

Function DSP_VectorMul16:

```
void DSP_VectorMul16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);
```

Multiplies the value of each element of indata1 * indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in the Q15 fractional format. outdata[i] filled with N elements of indata1[i] * indata2[i]

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata1	pointer to input array of 16-bit elements (int16_t)
indata2	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufMultA[8]={0x7FFF, 0x8000, 0x7333, 0x6666, 0x1999, 0x4000, 0x7FFF, 0xB334};
//      1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int16_t inBufMultB[8]={0x0CCD, 0x0CCD, 0x4000, 0xC000, 0xE667, 0x4000, 0x0000, 0x0CCD};
//      0.1,      0.1,      0.5,      -0.5,      -0.2,      0.5,      0,      0.1
int Num = 8;

pOutdata = &outVal;

DSP_VectorMul16(pOutdata, inBufMultA, inBufMultB, Num);

// outVal[i] = inBufTest[i] * inBuf2[i] =
//      {0x0CCD, 0xF333, 0x399A, 0xCCCD, 0xFAE2, 0x2000, 0x0000, 0xF852}
//      0.1,      -0.1,      0.45,      -0.4,      -0.04,      0.25,      0,      -0.06
```

5.4.1.5.5.27 DSP_VectorMul32 Function**C**

```
void DSP_VectorMul32(
    int32_t * outdata,
    int32_t * indata1,
    int32_t * indata2,
    int N
);
```

Description

Function DSP_VectorMul32:

```
void DSP_VectorMul32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);
```

Multiplies the value of each element of indata1 * indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in the Q31 fractional format. outdata[i] filled with N elements of indata1[i] * indata2[i]

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata1	pointer to input array of 16-bit elements (int32_t)

indata2	pointer to input array of 16-bit elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```
int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufMultA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//          1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int32_t inBufMultB[8]={0x0CCCCCD, 0x0CCCCCD, 0x40000000, 0xC0000000,
                      0xE6666667, 0x40000000, 0x00000000, 0x0CCCCCD};
//          0.1,      0.1,      0.5,      -0.5,      -0.2,      0.5,      0,      0.1
int Num = 8;

pOutdata = &outVal;

DSP_VectorMul32(pOutdata, inBufMultA, inBufMultB, Num);

// outVal[i] = inBufTest[i] * inBuf2[i] =
// {0x0CCCCCD, 0xF3333333, 0x3999999A, 0xCCCCCD, 0xFAE147AE,
//   0x20000000, 0x00000000, 0xF851EB86}
// 0.1,      -0.1,      0.45,      -0.4,      -0.04,      0.25,      0,      -0.06
```

5.4.1.5.5.28 DSP_VectorMulc16 Function**C**

```
void DSP_VectorMulc16(
    int16_t * outdata,
    int16_t * indata,
    int16_t c,
    int N
);
```

Description

Function DSP_VectorMulc16:

```
void DSP_VectorMulc16(int16_t *outdata, int16_t *indata, int16_t c, int N);
```

Multiplies the value of each element of indata1 * c and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q15 fractional format. outdata[i] filled with N elements of indata[i] * c

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata	pointer to input array of 16-bit elements (int16_t)
c	scalar multiplicand (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufMultA[8]={0x7FFF, 0x8000, 0x7333, 0x6666, 0x1999, 0x4000, 0x0000, 0xB334};
//      1,      -1,      0.9,      0.8,      0.2,      0.5,      0,      -0.6
int16_t constValue = 0x4000;
int Num = 8;

pOutdata = &outVal;

DSP_VectorMulc16(pOutdata, inBufMultA, constValue, Num);

// outVal[i] = inBufTest[i] * constValue =
//      {0x4000, 0xC000, 0x399A, 0x3333, 0x1999, 0x2000, 0x0000, 0xD99A}
//      0.5,      -0.5,      0.45,      0.4,      0.1,      0.25,      0,      -0.3
```

5.4.1.5.5.29 DSP_VectorMulc32 Function**C**

```
void DSP_VectorMulc32(
    int32_t * outdata,
    int32_t * indata,
    int32_t c,
    int N
);
```

Description

Function DSP_VectorMulc32:

```
void DSP_VectorMulc32(int32_t *outdata, int32_t *indata, int32_t c, int N);
```

Multiples the value of each element of indata * c and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of indata[i] * c

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata	pointer to input array of 16-bit elements (int32_t)
c	scalar multiplicand (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```

int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufMultA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x00000000, 0xB3333334};
//      1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int32_t constValue = 0x4000;
int Num = 8;

pOutdata = &outVal;

DSP_VectorMulc32(pOutdata, inBufMultA, constValue, Num);

// outVal[i] = inBufTest[i] * constValue =
//      {0x40000000, 0xC0000000, 0x3999999A, 0x33333333, 0x19999999,
//      0x20000000, 0x00000000, 0xD999999A}
//      0.5,      -0.5,      0.45,      0.4,      0.1,      0.25,      0,      -0.3

```

5.4.1.5.5.30 DSP_VectorNegate Function**C**

```

void DSP_VectorNegate(
    int32_t * outdata,
    int32_t * indata,
    int N
);

```

Description

Function DSP_VectorNegate:

```
void DSP_VectorNegate(int32_t *outdata, int32_t *indata, int N);
```

Sign inversion of the first N values of an indata are assigned to outdata. N must be a multiple of four and greater than or equal to four or it will be truncated to the nearest multiple of four. The vector result and the scalar value to fill are both Q31 fractional format.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
outdata	pointer to desination array of elements (int32_t)
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

None.

Example

```

int Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//      1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int32_t outBufTest[8]={0x0CCCCCD, 0x0CCCCCD, 0x40000000, 0xC0000000,

```

```

//          0xE6666667, 0x40000000, 0x00000000, 0x0CCCCCD};
//          0.1,    0.1,    0.5,   -0.5,   -0.2,    0.5,    0,    0.1;

DSP_VectorNegate(outBufTest, inBufTestA, Num);

// inBufTestA = {0x80000000, 0x7FFFFFFF, 0x8CCCCCD, 0x9999999A,
//               0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334} // first 4 values neg
//               -1,    1,   -0.9,   -0.8,   -0.2,    0.5,    0,    0.1

```

5.4.1.5.5.31 DSP_VectorRecip Function

C

```

void DSP_VectorRecip(
    _Q16 * outVector,
    _Q16 * inVector,
    int N
);

```

Description

Function DSP_VectorRecip:

```
void DSP_VectorRecip(_Q16 *outVector, _Q16 *inVector, int N);
```

Computes the reciprocal (1/x) on the first N elements of inVector. The output is stored to outVector. Both vectors are Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. If values exceed maximum or minimum they will saturate to the maximum or minimum respectively.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Div function from the LIBQ library. That library must be compiled as part of the project.

Parameters

Parameters	Description
outdata	pointer to desination array of elements (<u>Q16</u>)
indata	pointer to source array of elements (<u>Q16</u>)
N	number of samples (int)

Returns

None.

Remarks

This function uses the Microchip PIC32MZ LIBQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Div function see the LIBQ documentation for LIBQ_Q16Div. A value of zero in the array will not cause an error, but will return 0.

Example

```

int recNum = 4;
_Q16 inRecVec[8] = {0x08000000, 0xfffc0000, 0x00020000, 0x00100000, 0x00038000,
                   0x00400000, 0xfffe0000, 0x00058000};
//          2048.0, -4.0, 2.0, 16.0, 3.5, 64.0, -2.0, 5.5
_Q16 outRecVec[8] = {0};

DSP_VectorRecip(outRecVec, inRecVec, recNum);

// outRecVec = 0x00000020, 0xFFFFC000, 0x00008000, 0x00001000, 0, 0, 0, 0
//          0.000488,   -0.25,         0.5,       0.0625, 0, 0, 0, 0

```

5.4.1.5.5.32 DSP_VectorRMS16 Function

C

```
int16_t DSP_VectorRMS16(
    int16_t * inVector,
    int N
);
```

Description

Function DSP_VectorRMS16:

```
int16_t DSP_VectorRMS16(int16_t *inVector, int N);
```

Computes the root mean square value of the first N values of inVector. Both input and output are Q15 fractional values. The function will saturate if maximum or minimum values are exceeded.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and multiple of four.

Parameters

Parameters	Description
indata	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

Returns

int16_t - RMS function output, Q15 format

Remarks

This function is optimized with microMIPS and M14KCe ASE DSP instructions. This function is dependent on the LIBQ library, and uses the [_LIBQ_Q16Sqrt](#) external function call.

Example

```
int16_t vecRMSIn[32]={0x1999, 0xD99A, 0x4000, 0x2666,0x1999,0x1999,0x2666, 0x3333};
// 0.2, -0.3, 0.5, 0.3, 0.2, 0.2, 0.3, 0.4
int16_t RMSOut=0;
int Nrms = 8;

RMSOut = DSP_VectorRMS16(vecRMSIn, Nrms);

// RMSOut = 0x287C (= 0.31628)
```

5.4.1.5.5.33 DSP_VectorShift Function

C

```
void DSP_VectorShift(
    int32_t * outdata,
    int32_t * indata,
    int N,
    int shift
);
```

Description

Function DSP_VectorShift:

```
void DSP_VectorShift(int32_t *outdata, int32_t *indata, int N, int shift);
```

Shifts N data elements of indata to outdata, with an index change of shift. The amount of data shifted includes zero padding for

the first (shift) elements if shift is positive. The vector size of indata and outdata need not be the same, however, N must not exceed either array size.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must not exceed the amount of elements in the source array. shift must not exceed the number of elements in the destination array.

Parameters

Parameters	Description
outdata	pointer to desination array of elements (int32_t)
indata	pointer to source array of elements (int32_t)
N	number of samples (int)
shift	number of indexes to shift (int)

Returns

None.

Remarks

Destination array values shift to left (relative to the input vector) when shift is positive (back filled with zeros) and shift to the right when shift is negative. The total amount of values copied to the destination array is the length of N less the shift amount.

Example

```
int shiftValue = 3;
int Num = 8;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//           1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int32_t inBufTestB[8]={0x80000000, 0x7FFFFFFF, 0x40000000, 0x0CCCCCCC,
                      0x40000000, 0x60000000, 0x80000000, 0x20000000};
//           -1,      1,      0.5,      0.1,      0.75,      0.5,      -1,      0.25

DSP_VectorShift(inBufTestA, inBufTestB, Num, shiftVal);

// inBufTestA = {0x00000000, 0x00000000, 0x00000000, 0x80000000,
//               0x7FFFFFFF, 0x40000000, 0x0CCCCCCC, 0x40000000} // shifted 3 positive
```

5.4.1.5.5.34 DSP_VectorSqrt Function

C

```
void DSP_VectorSqrt(
    _Q16 * outVector,
    _Q16 * inVector,
    int N
);
```

Description

Function DSP_VectorSqrt:

```
void DSP_VectorSqrt(_Q16 *outVector, _Q16 *inVector, int N);
```

Computes the Sqrt(x) on the first N elements of inVector. The output is stored to outVector. Both vectors are Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Sqrt function from the LIBQ library. That library must be compiled as part of the project.

Parameters

Parameters	Description
outdata	pointer to desination array of elements (_Q16)
indata	pointer to source array of elements (_Q16)
N	number of samples (int)

Returns

None.

Remarks

This function uses the Microchip PIC32MZ LIBQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Sqrt function see the LIBQ documentation for [_LIBQ_Q16Sqrt](#). A negative number input will return a saturated value (0x00FFFFxx).

Example

```
int sqrtNum = 4;
_Q16 inSqrtVec[8] = {0x40000000, 0xffff0000, 0x00020000, 0x00100000, 0x00038000,
                    0x00400000, 0xfffe0000, 0x00058000};
//      16384.0,  -1.0,  2.0,  16.0,   3.5,   64.0,  -2.0,  5.5
_Q16 outSqrtVec[8] = {0};

DSP_VectorSqrt(outSqrtVec, inSqrtVec,  sqrtNum);

// outSqrtVec =  0x00800000, 0x00FFFF80, 0x00016A0A, 0x00040000, 0, 0, 0, 0
//              128.0,  sat negative,  1.41422,    4.0,    0, 0, 0, 0
```

5.4.1.5.5.35 DSP_VectorStdDev16 Function**C**

```
int16_t DSP_VectorStdDev16(
    int16_t * inVector,
    int N
);
```

Description

Function DSP_VectorStdDev16:

```
int16_t DSP_VectorStdDev16(int16_t *inVector, int N);
```

Calculates the standard deviation on the first N elements of inVector and returns the 16-bit scalar result. The standard deviation is the square root of the variance, which is a measure of the delta from mean values. The mean value of the vector is computed in the process. The function has the form -

$$\text{StdDev} = \text{SQRT}(\text{sum}[0..N]((x(i) - M(N))^2) / (N-1))$$
 where N is the number of vector elements x(i) is a single element in the vector M(N) is the mean of the N elements of the vector

Input values of the vector and output scalar value is Q15 fractional format. This format has data that ranges from -1 to 1, and has internal saturation limits of those same values. Some care has been taken to reduce the impact of saturation by adding processing steps to effectively complete the processing in blocks. However, in some extreme cases of data variance it is still possible to reach the saturation limits.

Preconditions

The pointers inVector must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four. Dependent on use of the LIBQ library.

Parameters

Parameters	Description
inVector	pointer to source array of elements (int16_t)
N	number of samples (int)

Returns

int16_t - Standard Deviation of N selected elements

Remarks

The input vector elements number, N, must be at least 4 and a multiple of 4. This function is optimized with microMIPS and M14KCe ASE DSP instructions. This function is dependent on the LIBQ library, and the [_LIBQ_Q16Sqrt](#) specifically.

Example

```
int16_t vecStdDevIn[32]={0x4000, 0xD99A, 0x1000, 0x6000,0x1999,0x1999,0x2666, 0x3333};
// .2, -.3, .125, .75, .2, .2, .3, .4
int16_t StdDevOut, Var16Out;
int Nstdev = 4;

StdDevOut = DSP_VectorStdDev16(vecStdDevIn, Nstdev);
// StdDevOut = 0x3A9E (= 0.45797)
```

5.4.1.5.36 DSP_VectorSub16 Function**C**

```
void DSP_VectorSub16(
    int16_t * outdata,
    int16_t * indata1,
    int16_t * indata2,
    int N
);
```

Description

Function DSP_VectorSub16:

```
void DSP_VectorSub16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);
```

Computes the difference value of each element of indata1 - indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q15 fractional format. outdata[i] filled with N elements of indata1[i] - indata2[i]

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata1	pointer to input array of 16-bit elements (int16_t)
indata2	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```

int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
int16_t inBuf2[16]=     { 1,2, 3,4, 5,6, 7, 8, 9, 10,-1,-100,-127,127,-7, 0};
int Num = 8;

pOutdata = &outVal;

DSP_VectorSub16(pOutdata, inBufTest, inBuf2, Num);

// outVal[i] = inBufTest[i] - inBuf2[i] = {-6,0,-6,0,-6,-6,-9,-16}

```

5.4.1.5.5.37 DSP_VectorSub32 Function**C**

```

void DSP_VectorSub32(
    int32_t * outdata,
    int32_t * indata1,
    int32_t * indata2,
    int N
);

```

Description

Function DSP_VectorSub32:

```
void DSP_VectorSub32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);
```

Computes the difference value of each element of indata1 - indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of indata1[i] - indata2[i]

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata1	pointer to input array of 16-bit elements (int32_t)
indata2	pointer to input array of 16-bit elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

This must be assembled with .set microMIPS.

Example

```

int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
int32_t inBuf2[16]=     { 1,2, 3,4, 5,6, 7, 8, 9, 10,-1,-100,-127,127,-7, 0};
int Num = 8;

pOutdata = &outVal;

DSP_VectorSub32(pOutdata, inBufTest, inBuf2, Num);

```

```
// outVal[i] = inBufTest[i] - inBuf2[i] = {-6,0,-6,0,-6,-6,-9,-16}
```

5.4.1.5.5.38 DSP_VectorSumSquares16 Function

C

```
int16_t DSP_VectorSumSquares16(
    int16_t * indata,
    int N,
    int scale
);
```

Description

Function DSP_VectorSumSquares16:

```
int16_t DSP_VectorSumSquares16(int16_t *indata, int N, int scale);
```

Calculates the sum of the squares of each element of an input vector, and scales the output. Function will saturate if it exceeds maximum or minimum values. Scaling is done by binary shifting, after accumulation in a 32 bit register. All calculations are done in Q15 fractional format. return = $1/(2^{\text{scale}}) * \text{sum}(\text{indata}[i]^2)$

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Parameters

Parameters	Description
indata	pointer to input array of 16-bit elements (int16_t)
scale	number of bits to shift return right (int)
N	number of samples (int)

Returns

int16_t - scaled output of calculation, Q15 format

Remarks

This must be assembled with .set microMIPS.

Example

```
int16_t inBufMultA[8]={0x7FFF, 0x8000, 0x7333, 0x6666, 0x1999, 0x4000, 0x7FFF, 0xB334};
//                      1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int Num = 8;
int scaleVal = 3;
int16_t outScalar;

outScalar = DSP_VectorSumSquares16(inBufMultA, Num, scaleVal);

// outScalar = 1/(2^scaleVal)* sum(inBufMultA[i]^2) =
// (1/8) * (1 + 1 + 0.81 + 0.64 + 0.04 + 0.25 + 1 + 0.36) = 0.125 * 5.1 = 0.6375
// = (int16_t)0x5199
```

5.4.1.5.5.39 DSP_VectorSumSquares32 Function

C

```
int32_t DSP_VectorSumSquares32(
    int32_t * indata,
    int N,
    int scale
);
```

Description

Function DSP_VectorSumSquares32:

```
int32_t DSP_VectorSumSquares32(int32_t *indata, int N, int scale);
```

Calculates the sum of the squares of each element of an input vector, and scales the output. The function will saturate if it exceeds maximum or minimum values. Scaling is done by binary shifting, after calculation of the results. All calculations are done in Q31 fractional format. $\text{return} = 1/(2^{\text{scale}}) * \text{sum}(\text{indata}[i]^2)$

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and multiple of four.

Parameters

Parameters	Description
indata	pointer to input array of 16-bit elements (int32_t)
scale	number of bits to shift return right (int)
N	number of samples (int)

Returns

int32_t - scaled output of calculation, Q15 format

Remarks

This must be assembled with .set microMIPS.

Example

```
int32_t inBufMultA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x00000000, 0xB3333334};
//          1,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6
int Num = 8;
int scaleVal = 3;
int32_t outScalar;

outScalar = DSP_VectorSumSquares32(inBufMultA, Num, scaleVal);

// outScalar = 1/(2^scaleVal)* sum(inBufMultA[i]^2) =
// (1/8) * (1 + 1 + 0.81 + 0.64 + 0.04 + 0.25 + 1 + 0.36) = 0.125 * 5.1 = 0.6375
// = (int32_t)0x51999999
```

5.4.1.5.5.40 DSP_VectorVari16 Function**C**

```
int16_t DSP_VectorVari16(
    int16_t * inVector,
    int N
);
```

Description

Function DSP_VectorVari16:

```
int16_t DSP_VectorVari16(int16_t *inVector, int N);
```

Calculates the variance on the first N elements of inVector and returns the 16-bit scalar result. The variance is a measure of the delta from mean values, and the mean value of the vector is computed in the process. The function has the form -

$\text{var} = \text{sum}[0..N]((x(i) - M(N))^2) / (N-1)$ where N is the number of vector elements x(i) is a single element in the vector M(N) is the mean of the N elements of the vector

Input values of the vector and output scalar value is Q15 fractional format. This format has data that ranges from -1 to 1, and has internal saturation limits of those same values. Some care has been taken to reduce the impact of saturation by adding processing steps to effectively complete the processing in blocks. However, in some extreme cases of data variance it is still possible to reach the saturation limits.

Preconditions

The pointers inVector must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
inVector	pointer to source array of elements (int16_t)
N	number of samples (int)

Returns

int16_t - Variance of N selected elements

Remarks

The input vector elements number, N, must be at least 4 and a multiple of 4. This function is optimized with microMIPS and M14KCe ASE DSP instructions.

Example

```
int16_t vecStDevIn[32]={0x4000, 0xD99A, 0x1000, 0x6000,0x1999,0x1999,0x2666, 0x3333};
// .2, -.3, .125, .75, .2, .2, .3, .4
int16_t Var16Out;
int Nvar = 4;

Var16Out= DSP_VectorVari16(vecStDevIn, Nvar); // 16-bit variance function
// Var16Out = 0x1AD8 (= 0.20974)
```

5.4.1.5.5.41 DSP_VectorVariance Function

C

```
int32_t DSP_VectorVariance(
    int32_t * inVector,
    int N
);
```

Description

Function DSP_VectorVariance:

```
int32_t DSP_VectorVariance(int32_t *inVector, int N);
```

Calculates the variance on the first N elements of inVector and returns the 32-bit scalar result. The variance is a measure of the delta from mean values, and the mean value of the vector is computed in the process. The function has the form -

$$\text{var} = \text{sum}[0..N]((x(i) - M(N))^2) / (N-1)$$
 where N is the number of vector elements x(i) is a single element in the vector M(N) is the mean of the N elements of the vector

Input values of the vector and output scalar value is Q31 fractional format. This format has data that ranges from -1 to 1, and has internal saturation limits of those same values. Some care has been taken to reduce the impact of saturation by adding processing steps to effectively complete the processing in blocks. However, in some extreme cases of data variance it is still possible to reach the saturation limits.

Preconditions

The pointers inVector must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
inVector	pointer to source array of elements (int32_t)
N	number of samples (int)

Returns

int32_t - Variance of N selected elements

Remarks

The input vector elements number, N, must be at least 4 and a multiple of 4. This function is optimized with microMIPS and M14KCe ASE DSP instructions.

Example

```
int varN = 8;

int32_t inVarVec[8] = {0xE6666667, 0x40000000, 0x40000000, 0x0CCCCCCC,
                      0x00000000, 0x59999999, 0x20000000, 0xC0000000};
//      -0.2, 0.5, 0.5, 0.1, 0, 0.7, 0.25, -0.5
int32_t outVar = 0;

outVar = DSP_VectorVariance(inVarVec, varN);

// outVar == 0x1490D2A6, = 0.1606696
```

5.4.1.5.5.42 DSP_VectorZeroPad Function**C**

```
void DSP_VectorZeroPad(
    int32_t * indata,
    int N
);
```

Description

Function DSP_VectorZeroPad:

```
void DSP_VectorZeroPad(int32_t *indata, int N);
```

Fills the first N values of an input vector indata with the value zero. N must be a multiple of four and greater than or equal to four or it will be truncated to the nearest multiple of four. The vector result is in Q31 fractional format.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Parameters

Parameters	Description
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

Returns

None.

Remarks

None.

Example

```

int Num = 4;
int32_t inBufTestA[8]={0x3FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                      0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//          0.5,      -1,      0.9,      0.8,      0.2,      0.5,      1,      -0.6

DSP_VectorZeroPad(inBufTestA, Num);

// inBufTestA = {0x00000000, 0x00000000, 0x00000000, 0x00000000,
//               0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//               0,        0,        0,        0,        0.2,      0.5,      1,      -0.6

```

5.4.1.5.6 Support Functions**5.4.1.5.6.1 mul16 Function****C**

```

static inline int16_t mul16(
    int16_t a,
    int16_t b
);

```

Description

multiply and shift integer

5.4.1.5.6.2 mul16r Function**C**

```

static inline int16_t mul16r(
    int16_t a,
    int16_t b
);

```

Description

multiply and shift Q15

5.4.1.5.6.3 mul32 Function**C**

```

static inline int32_t mul32(
    int32_t a,
    int32_t b
);

```

Description

multiply and shift Q31

5.4.1.5.6.4 SAT16 Function**C**

```

static inline int32_t SAT16(
    int32_t x
);

```

Description

saturate both positive and negative Q15

5.4.1.5.6.5 SAT16N Function**C**

```
static inline int32_t SAT16N(
    int32_t x
);
```

Description

saturate negative Q15

5.4.1.5.6.6 SAT16P Function**C**

```
static inline int32_t SAT16P(
    int32_t x
);
```

Description

saturate positive Q15

5.4.1.5.7 Data Types and Constants**5.4.1.5.7.1 biquad16 Structure****C**

```
typedef struct {
    int16_t a1;
    int16_t a2;
    int16_t b1;
    int16_t b2;
} biquad16;
```

Description

Q15 biquad

Members

Members	Description
int16_t a1;	feedback delay 1 coef
int16_t a2;	feedback delay 2 coef
int16_t b1;	feedforward delay 1 coef
int16_t b2;	feedforward delay 1 coef

5.4.1.5.7.2 int16c Structure**C**

```
typedef struct {
    int16_t re;
    int16_t im;
} int16c;
```


Description

Q15 complex number ($a + bi$)

Members

Members	Description
int16_t re;	real portion (a)
int16_t im;	imaginary portion (b)

5.4.1.5.7.3 int32c Structure**C**

```
typedef struct {
    int32_t re;
    int32_t im;
} int32c;
```

Description

Q31 complex number ($a + bi$)

Members

Members	Description
int32_t re;	real portion (a)
int32_t im;	imaginary portion (b)

5.4.1.5.7.4 matrix32 Structure**C**

```
typedef struct {
    int32_t row;
    int32_t col;
    int32_t * pMatrix;
} matrix32;
```

Description

Q31 matrix

Members

Members	Description
int32_t row;	matrix rows
int32_t col;	matrix columns
int32_t * pMatrix;	matrix pointer to data

5.4.1.5.7.5 PARM_EQUAL_FILTER Structure**C**

```
typedef struct _PARM_EQUAL_FILTER {
    PARM_FILTER_GAIN G;
    int16_t log2Alpha;
    int16_t b[3];
    int16_t a[2];
    int32_t Z[2];
} PARM_EQUAL_FILTER;
```

Description

IIR BQ filter structure Q15 data, Q31 storage

Members

Members	Description
PARM_FILTER_GAIN G;	Filter max gain multiplier
int16_t log2Alpha;	coefficient scaling bit shift value
int16_t b[3];	Feedforward Coefficients, Q15 format
int16_t a[2];	Feedback Coefficients, Q15 format
int32_t Z[2];	Filter memory, should be initialized to zero.

5.4.1.5.7.6 PARM_EQUAL_FILTER_16 Structure**C**

```
typedef struct _PARM_EQUAL_FILTER_16 {
    PARM_FILTER_GAIN G;
    int16_t log2Alpha;
    int16_t b[3];
    int16_t a[2];
    int16_t Z[2];
} PARM_EQUAL_FILTER_16;
```

Description

IIR BQ filter structure Q15

Members

Members	Description
PARM_FILTER_GAIN G;	Filter max gain multiplier
int16_t log2Alpha;	coefficient scaling bit shift value
int16_t b[3];	Feedforward Coefficients, Q15 format
int16_t a[2];	Feedback Coefficients, Q15 format
int16_t Z[2];	Filter memory, should be initialized to zero.

5.4.1.5.7.7 PARM_EQUAL_FILTER_32 Structure**C**

```
typedef struct _PARM_EQUAL_FILTER_32 {
    PARM_FILTER_GAIN G;
    int log2Alpha;
    int32_t b[3];
    int32_t a[2];
    int32_t Z[2];
} PARM_EQUAL_FILTER_32;
```

Description

IIR BQ filter structure Q31

Members

Members	Description
PARM_FILTER_GAIN G;	Filter max gain multiplier
int log2Alpha;	coefficient scaling bit shift value
int32_t b[3];	Feedforward Coefficients, Q15 format

int32_t a[2];	Feedback Coefficients, Q15 format
int32_t Z[2];	Filter memory, should be initialized to zero.

5.4.1.5.7.8 PARM_FILTER_GAIN Structure

C

```
typedef struct {
    int16_t fracGain;
    int16_t expGain;
} PARM_FILTER_GAIN;
```

Description

filter gain structure

Members

Members	Description
int16_t fracGain;	Q15 fractional filter gain
int16_t expGain;	log2N (binary) filter gain

5.4.1.5.7.9 MAX16 Macro

C

```
#define MAX16 ((int16_t) 0x7FFF) // maximum Q15
```

Description

maximum Q15

5.4.1.5.7.10 MAX32 Macro

C

```
#define MAX32 ((int32_t) 0x7FFFFFFF) // maximum Q31
```

Description

maximum Q31

5.4.1.5.7.11 MIN16 Macro

C

```
#define MIN16 ((int16_t) 0x8000) // minimum Q15
```

Description

minimum Q15

5.4.1.5.7.12 MIN32 Macro

C

```
#define MIN32 ((int32_t) 0x80000000) // minimum Q31
```

Description

minimum Q31

5.4.1.6 Files

Files

Name	Description
dsp.h	DSP functions for the PIC32MZ device family

Description

5.4.1.6.1 dsp.h

Digital Signal Processing (DSP) Library

The DSP Library provides functions supplied by MIPS and Microchip that are optimized for performance on the microAptiv(TM) core of the PIC32MZ family of microcontrollers. The library provides advanced mathematical operations for complex numbers, vector and matrix mathematics, digital filtering and transforms.

All functions are implemented in efficient assembly with C-callable prototypes. In some cases both 16-bit and 32-bit functions are supplied to enable the user with a choice of resolution and performance.

For most functions, input and output data is represented by 16-bit fractional numbers in Q15 format, which is the most commonly used data format for signal processing. Some functions use other data formats internally for increased precision of intermediate results.

The Q15 data type used by the DSP functions is specified as `int16_t` in the C header file that is supplied with the library. Note that within C code, care must be taken to avoid confusing fixed-point values with integers. To the C compiler, objects declared with `int16_t` type are integers, not fixed-point, and all arithmetic operations performed on those objects in C will be done as integers. Fixed-point values have been declared as `int16_t` only because the standard C language does not include intrinsic support for fixed-point data types.

Some functions also have versions operating on 32-bit fractional data in Q31 format. These functions operate similarly to their 16-bit counterparts. However, it should be noted that the 32-bit functions do not benefit much from the SIMD capabilities offered by DSP ASE. Thus, the performance of the 32-bit functions is generally reduced compared to the performance of the corresponding 16-bit functions.

Signed fixed point types are defined as follows:

`Qn.m` where:

- `n` is the number of data bits to the left of the radix point
- `m` is the number of data bits to the right of the radix point
- a signed bit is implied

Unique variable types for fractional representation are also defined:

Exact Name # Bits Required Type Q0.15 (Q15) 16 `int16_t` Q0.31 (Q31) 32 `int32_t`

Table of DSP Library functions:

Complex Math:

```
void DSP\_ComplexAdd32(int32c *indata1, int32c *indata2, int32c *Output);  
void DSP\_ComplexConj16(int16c *indata, int16c *Output);  
void DSP\_ComplexConj32(int32c *indata, int32c *Output);
```

```
void DSP_ComplexDotProd32(int32c *indata1, int32c *indata2, int32c *Output);
```

```
void DSP_ComplexMult32(int32c *indata1, int32c *indata2, int32c *Output);
```

```
void DSP_ComplexScalarMult32(int32c *indata, int32_t Scalar, int32c *Output);
```

```
void DSP_ComplexSub32(int32c *indata1, int32c *indata2, int32c *Output);
```

Filter Functions:

```
void DSP_FilterFIR32(int32_t *outdata, int32_t *indata, int32_t *coeffs2x, int32_t *delayline, int N, int K, int scale);
```

```
void DSP_FilterFIRDecim32(int32_t *outdata, int32_t *indata, int32_t *coeffs, int32_t *delayline, int N, int K, int scale, int rate);
```

```
int16_t DSP_FilterIIR16(int16_t in, int16_t *coeffs, int16_t *delayline, int B, int scale);
```

```
void DSP_FilterIIRSetup16(int16_t *coeffs, biquad16 *bq, int B);
```

```
void DSP_FilterFIRInterp32(int32_t *outdata, int32_t *indata, int32_t *coeffs, int32_t *delayline, int N, int K, int scale, int rate);
```

```
int16_t DSP_FilterLMS16(int16_t in, int16_t ref, int16_t *coeffs, int16_t *delayline, int16_t *error, int K, int16_t mu);
```

```
int16_t DSP_FilterIIRBQ16_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter);
```

```
int16_t DSP_FilterIIRBQ16(int16_t Xin, PARM_EQUAL_FILTER *pFilter);
```

```
int32_t DSP_FilterIIRBQ32(int32_t Xin, PARM_EQUAL_FILTER_32 *pFilter);
```

```
int16_t DSP_FilterIIRBQ16_cascade8(int16_t Xin, PARM_EQUAL_FILTER *pFilter_Array);
```

```
int16_t DSP_FilterIIRBQ16_cascade8_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter_Array);
```

```
int16_t DSP_FilterIIRBQ16_parallel8(int16_t Xin, PARM_EQUAL_FILTER *pFilter);
```

```
int16_t DSP_FilterIIRBQ16_parallel8_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter);
```

Matrix Functions:

```
void DSP_MatrixAdd32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);
```

```
void DSP_MatrixEqual32(matrix32 *resMat, matrix32 *srcMat);
```

```
void DSP_MatrixInit32(int32_t *data_buffer, int32_t N, int32_t num);
```

```
void DSP_MatrixMul32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);
```

```
void DSP_MatrixScale32(int32_t *data_buffer, int32_t N, int32_t num);
```

```
void DSP_MatrixSub32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);
```

```
void DSP_MatrixTranspose32(matrix32 *desMat, matrix32 *srcMat);
```

Transforms:

```
void DSP_TransformFFT16(int16c *dout, int16c *din, int16c *twiddles, int16c *scratch, int log2N);
```

```
void DSP_TransformIFFT16(int16c *dout, int16c *din, int16c *twiddles, int16c *scratch, int log2N);
```

```
void DSP_TransformFFT16_setup(int16c *twiddles, int log2N);
```

```
void DSP_TransformFFT32(int32c *dout, int32c *din, int32c *twiddles, int32c *scratch, int log2N);
```

```
void DSP_TransformFFT32_setup(int32c *twiddles, int log2N);
```

```
void DSP_TransformWindow_Bart16(int16_t *OutVector, int16_t *InVector, int N);
```

```
void DSP_TransformWindow_Bart32(int32_t *OutVector, int32_t *InVector, int N);
```

```
void DSP_TransformWindow_Black16(int16_t *OutVector, int16_t *InVector, int N);
```

```
void DSP\_TransformWindow\_Black32(int32_t *OutVector, int32_t *InVector, int N);
void DSP\_TransformWindow\_Cosine16(int16_t *OutVector, int16_t *InVector, int N);
void DSP\_TransformWindow\_Cosine32(int32_t *OutVector, int32_t *InVector, int N);
void DSP\_TransformWindow\_Hamm16(int16_t *OutVector, int16_t *InVector, int N);
void DSP\_TransformWindow\_Hamm32(int32_t *OutVector, int32_t *InVector, int N);
void DSP\_TransformWindow\_Hann16(int16_t *OutVector, int16_t *InVector, int N);
void DSP\_TransformWindow\_Hann32(int32_t *OutVector, int32_t *InVector, int N);
void DSP\_TransformWindow\_Kaiser16(int16_t *OutVector, int16_t *InVector, int N);
void DSP\_TransformWindow\_Kaiser32(int32_t *OutVector, int32_t *InVector, int N);
void DSP\_TransformWinInit\_Bart16(int16_t *OutWindow, int N);
void DSP\_TransformWinInit\_Bart32(int32_t *OutWindow, int N);
void DSP\_TransformWinInit\_Black16(int16_t *OutWindow, int N);
void DSP\_TransformWinInit\_Black32(int32_t *OutWindow, int N);
void DSP\_TransformWinInit\_Cosine16(int16_t *OutWindow, int N);
void DSP\_TransformWinInit\_Cosine32(int32_t *OutWindow, int N);
void DSP\_TransformWinInit\_Hamm16(int16_t *OutWindow, int N);
void DSP\_TransformWinInit\_Hamm32(int32_t *OutWindow, int N);
void DSP\_TransformWinInit\_Hann16(int16_t *OutWindow, int N);
void DSP\_TransformWinInit\_Hann32(int32_t *OutWindow, int N);
void DSP\_TransformWinInit\_Kaiser16(int16_t *OutWindow, int N);
void DSP\_TransformWinInit\_Kaiser32(int32_t *OutWindow, int N);
```

Vector Math:

```
void DSP\_VectorAbs16(int16_t *outdata, int16_t *indata, int N);
void DSP\_VectorAbs32(int32_t *outdata, int32_t *indata, int N);
void DSP\_VectorAdd16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);
void DSP\_VectorAdd32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);
void DSP\_VectorAddc16(int16_t *outdata, int16_t *indata, int16_t c, int N);
void DSP\_VectorAddc32(int32_t *outdata, int32_t *indata, int32_t c, int N);
void DSP\_VectorAutocorr16(int16_t *outCorr, int16_t *inVector, int N, int K);
int DSP\_VectorBexp16(int16_t *DataIn, int N);
int DSP\_VectorBexp32(int32_t *DataIn, int N);
int DSP\_VectorChkEqu32(int32_t *indata1, int32_t *indata2, int N);
void DSP\_VectorCopy(int32_t *outdata, int32_t *indata, int N);
void DSP\_VectorCopyReverse32(int32_t *outdata, int32_t *indata, int N);
void DSP\_VectorDivC(\_Q16 *outVector, \_Q16 *inVector, \_Q16 divisor, int N);
```

```

int16_t DSP\_VectorDotp16(int16_t *indata1, int16_t *indata2, int N, int scale);
int32_t DSP\_VectorDotp32(int32_t *indata1, int32_t *indata2, int N, int scale);
void DSP\_VectorExp(\_Q16 *outVector, \_Q16 *inVector, int N);
void DSP\_VectorFill(int32_t *indata, int32_t data, int N);
void DSP\_VectorLog10(\_Q3\_12 *outVector, \_Q16 *inVector, int N);
void DSP\_VectorLog2(\_Q5\_10 *outVector, \_Q16 *inVector, int N);
void DSP\_VectorLn(\_Q4\_11 *outVector, \_Q16 *inVector, int N);
int32_t DSP\_VectorMax32(int32_t *indata, int N);
int DSP\_VectorMaxIndex32(int32_t *indata, int N);
int32_t DSP\_VectorMean32(int32_t *indata, int N);
int32_t DSP\_VectorMin32(int32_t *input, int N);
int DSP\_VectorMinIndex32(int32_t *indata, int N);
void DSP\_VectorMul16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);
void DSP\_VectorMul32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);
void DSP\_VectorMulc16(int16_t *outdata, int16_t *indata, int16_t c, int N);
void DSP\_VectorMulc32(int32_t *outdata, int32_t *indata, int32_t c, int N);
void DSP\_VectorNegate(int32_t *outdata, int32_t *indata, int N);
void DSP\_VectorRecip(\_Q16 *outVector, \_Q16 *inVector, int N);
int16_t DSP\_VectorRMS16(int16_t *inVector, int N);
void DSP\_VectorShift(int32_t *outdata, int32_t *indata, int N, int shift);
int16_t DSP\_VectorStdDev16(int16_t *inVector, int N);
void DSP\_VectorSqrt(\_Q16 *outVector, \_Q16 *inVector, int N);
void DSP\_VectorSub16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);
void DSP\_VectorSub32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);
int16_t DSP\_VectorSumSquares16(int16_t *indata, int N, int scale);
int32_t DSP\_VectorSumSquares32(int32_t *indata, int N, int scale);
int16_t DSP\_VectorVari16(int16_t *inVector, int N);
int32_t DSP\_VectorVariance(int32_t *inVector, int N);
void DSP\_VectorZeroPad(int32_t *indata, int N);

```

Functions

	Name	Description
≡	DSP_ComplexAdd32	Calculates the sum of two complex numbers.
≡	DSP_ComplexConj16	Calculates the complex conjugate of a complex number.
≡	DSP_ComplexConj32	Calculates the complex conjugate of a complex number.
≡	DSP_ComplexDotProd32	Calculates the dot product of two complex numbers.
≡	DSP_ComplexMult32	Multiplies two complex numbers.

	DSP_ComplexScalarMult32	Multiplies a complex number and a scalar number.
	DSP_ComplexSub32	Calculates the difference of two complex numbers.
	DSP_FilterFIR32	Performs a Finite Impulse Response filter on a vector.
	DSP_FilterFIRDecim32	Performs a decimating FIR filter on the input array.
	DSP_FilterFIRInterp32	Performs an interpolating FIR filter on the input array.
	DSP_FilterIIR16	Performs a single-sample cascaded biquad Infinite Impulse Response (IIR) filter.
	DSP_FilterIIRBQ16	Performs a single-pass IIR Biquad Filter.
	DSP_FilterIIRBQ16_cascade8	Performs a single IIR Biquad Filter as a cascade of 8 series filters.
	DSP_FilterIIRBQ16_cascade8_fast	Performs a single IIR Biquad Filter as a cascade of 8 series filters.
	DSP_FilterIIRBQ16_fast	Performs a single-pass IIR Biquad Filter.
	DSP_FilterIIRBQ16_parallel8	Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.
	DSP_FilterIIRBQ16_parallel8_fast	Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.
	DSP_FilterIIRBQ32	Performs a high resolution single-pass IIR Biquad Filter.
	DSP_FilterIIRSetup16	Converts biquad structure to coeffs array to set up IIR filter.
	DSP_FilterLMS16	Performs a single sample Least Mean Squares FIR Filter.
	DSP_MatrixAdd32	Addition of two matrices $C = (A + B)$.
	DSP_MatrixEqual32	Equality of two matrices $C = (A)$.
	DSP_MatrixInit32	Initializes the first N elements of a Matrix to the value num.
	DSP_MatrixMul32	Multiplication of two matrices $C = A \times B$.
	DSP_MatrixScale32	Scales each element of an input buffer (matrix) by a fixed number.
	DSP_MatrixSub32	Subtraction of two matrices $C = (A - B)$.
	DSP_MatrixTranspose32	Transpose of a Matrix $C = A^T$.
	DSP_TransformFFT16	Creates an Fast Fourier Transform (FFT) from a time domain input.
	DSP_TransformFFT16_setup	Creates FFT coefficients for use in the FFT16 function.
	DSP_TransformFFT32	Creates an Fast Fourier Transform (FFT) from a time domain input.
	DSP_TransformFFT32_setup	Creates FFT coefficients for use in the FFT32 function.
	DSP_TransformIFFT16	Creates an Inverse Fast Fourier Transform (FFT) from a frequency domain input.
	DSP_TransformWindow_Bart16	Perform a Bartlett window on a vector.
	DSP_TransformWindow_Bart32	Perform a Bartlett window on a vector.
	DSP_TransformWindow_Black16	Perform a Blackman window on a vector.
	DSP_TransformWindow_Black32	Perform a Blackman window on a vector.
	DSP_TransformWindow_Cosine16	Perform a Cosine (Sine) window on a vector.
	DSP_TransformWindow_Cosine32	Perform a Cosine (Sine) window on a vector.
	DSP_TransformWindow_Hamm16	Perform a Hamming window on a vector.
	DSP_TransformWindow_Hamm32	Perform a Hamming window on a vector.
	DSP_TransformWindow_Hann16	Perform a Hanning window on a vector.
	DSP_TransformWindow_Hann32	Perform a Hanning window on a vector.
	DSP_TransformWindow_Kaiser16	Perform a Kaiser window on a vector.
	DSP_TransformWindow_Kaiser32	Perform a Kaiser window on a vector.
	DSP_TransformWinInit_Bart16	Create a Bartlett window.
	DSP_TransformWinInit_Bart32	Create a Bartlett window.
	DSP_TransformWinInit_Black16	Create a Blackman window.
	DSP_TransformWinInit_Black32	Create a Blackman window.
	DSP_TransformWinInit_Cosine16	Create a Cosine (Sine) window.

	DSP_TransformWinInit_Cosine32	Create a Cosine (Sine) window.
	DSP_TransformWinInit_Hamm16	Create a Hamming window.
	DSP_TransformWinInit_Hamm32	Create a Hamming window.
	DSP_TransformWinInit_Hann16	Create a Hanning window.
	DSP_TransformWinInit_Hann32	Create a Hanning window.
	DSP_TransformWinInit_Kaiser16	Create a Kaiser window.
	DSP_TransformWinInit_Kaiser32	Create a Kaiser window.
	DSP_VectorAbs16	Calculate the absolute value of a vector.
	DSP_VectorAbs32	Calculate the absolute value of a vector.
	DSP_VectorAdd16	Calculate the sum of two vectors.
	DSP_VectorAdd32	Calculate the sum of two vectors.
	DSP_VectorAddc16	Calculate the sum of two vectors.
	DSP_VectorAddc32	Calculate the sum of two vectors.
	DSP_VectorAutocorr16	Computes the Autocorrelation of a Vector.
	DSP_VectorBexp16	Computes the maximum binary exponent of a vector.
	DSP_VectorBexp32	Computes the maximum binary exponent of a vector.
	DSP_VectorChkEqu32	Compares two input vectors, returns an integer '1' if equal, and '0' if not equal.
	DSP_VectorCopy	Copies the elements of one vector to another.
	DSP_VectorCopyReverse32	Reverses the order of elements in one vector and copies them into another.
	DSP_VectorDivC	Divides the first N elements of inVector by a constant divisor, and stores the result in outVector.
	DSP_VectorDotp16	Computes the dot product of two vectors, and scales the output by a binary factor.
	DSP_VectorDotp32	Computes the dot product of two vectors, and scales the output by a binary factor
	DSP_VectorExp	Computes the EXP (e^x) of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorFill	Fills an input vector with scalar data.
	DSP_VectorLn	Computes the Natural Log, $\ln(x)$, of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorLog10	Computes the $\log_{10}(x)$, of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorLog2	Computes the $\log_2(x)$ of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorMax32	Returns the maximum value of a vector.
	DSP_VectorMaxIndex32	Returns the index of the maximum value of a vector.
	DSP_VectorMean32	Calculates the mean average of an input vector.
	DSP_VectorMin32	Returns the minimum value of a vector.
	DSP_VectorMinIndex32	Returns the index of the minimum value of a vector.
	DSP_VectorMul16	Multiplication of a series of numbers in one vector to another vector.
	DSP_VectorMul32	Multiplication of a series of numbers in one vector to another vector.
	DSP_VectorMulc16	Multiplication of a series of numbers in one vector to a scalar value.
	DSP_VectorMulc32	Multiplication of a series of numbers in one vector to a scalar value.
	DSP_VectorNegate	Inverses the sign (negates) the elements of a vector.
	DSP_VectorRecip	Computes the reciprocal ($1/x$) of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorRMS16	Computes the root mean square (RMS) value of a vector.

	DSP_VectorShift	Shifts the data index of an input data vector.
	DSP_VectorSqrt	Computes the square root of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorStdDev16	Computes the Standard Deviation of a Vector.
	DSP_VectorSub16	Calculate the difference of two vectors.
	DSP_VectorSub32	Calculate the difference of two vectors.
	DSP_VectorSumSquares16	Computes the sum of squares of a vector, and scales the output by a binary factor.
	DSP_VectorSumSquares32	Computes the sum of squares of a vector, and scales the output by a binary factor.
	DSP_VectorVari16	Computes the variance of N elements of a Vector.
	DSP_VectorVariance	Computes the variance of N elements of inVector.
	DSP_VectorZeroPad	Fills an input vector with zeros.
	mul16	multiply and shift integer
	mul16r	multiply and shift Q15
	mul32	multiply and shift Q31
	SAT16	saturate both positive and negative Q15
	SAT16N	saturate negative Q15
	SAT16P	saturate positive Q15

Macros

	Name	Description
	MAX16	maximum Q15
	MAX32	maximum Q31
	MIN16	minimum Q15
	MIN32	minimum Q31

Structures

	Name	Description
	_PARM_EQUAL_FILTER	IIR BQ filter structure Q15 data, Q31 storage
	_PARM_EQUAL_FILTER_16	IIR BQ filter structure Q15
	_PARM_EQUAL_FILTER_32	IIR BQ filter structure Q31
	biquad16	Q15 biquad
	int16c	Q15 complex number (a + bi)
	int32c	Q31 complex number (a + bi)
	matrix32	Q31 matrix
	PARM_EQUAL_FILTER	IIR BQ filter structure Q15 data, Q31 storage
	PARM_EQUAL_FILTER_16	IIR BQ filter structure Q15
	PARM_EQUAL_FILTER_32	IIR BQ filter structure Q31
	PARM_FILTER_GAIN	filter gain structure

File Name

dsp.h

Company

Microchip Technology Inc.

5.4.2 LibQ Fixed-Point Math Library

5.4.2.1 Introduction

LibQ Fixed-Point Math Library

for

Microchip Microcontrollers

The LibQ Fixed-Point Math Library is available for the PIC32MZ family of microcontrollers. This library was created from optimized assembly routines written specifically for the microAptive™ core.

Description

The LibQ Fixed-Point Math Library simplifies writing fixed point algorithms, supporting Q15, Q31 and other 16-bit and 32-bit data formats. Using the simple, C callable functions contained in the library, fast fixed point mathematical operations can be easily executed. Fixed-point mathematical calculations may replace some functions implemented in the floating point library (math.h), depending on performance and resolution requirements.

Functions included in the LibQ library include capabilities for trigonometric, power and logarithms, and data conversion. In many cases the functions are identical other than the precision of their operands and the corresponding value that they return.

These functions are implemented in efficient assembly, and generally tuned to optimize performance over code size. In some cases the library breaks out functions that enable one to be optimized for accuracy, while another version is optimized for speed. These functions such as `_LIBQ_Q2_29_acos_Q31` and `_LIBQ_Q2_29_acos_Q31_Fast` are otherwise identical and can be used interchangeably. Each of these functions are typically used in computationally intensive real-time applications where execution time is a critical parameter.

5.4.2.2 Release Notes

MPLAB Harmony Version: v0.70b **LibQ Fixed-Point Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

Fixed Point "libq" Math Library

- 31 functions
 - Trigonometric
 - Power, log and exponent functions
 - Divide & square root
 - Number conversions
 - Random number generator
- 16-bit and 32-bit fixed-point integer math
- Optimized in high-performance assembly
- Precision balanced with speed, some fast functions available

Known Issues:

- Optimized for PIC32MZ microAptiv core only
- `_fast` functions have reduced precision

5.4.2.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.4.2.4 Using the Library

This topic describes the basic architecture of the LibQ Fixed-Point Math Library and provides information and examples on how to use it.

Interface Header File: [libq.h](#)

The interface to the LibQ Fixed-Point Math Library is defined in the "[libq.h](#)" header file. Any C language source (.c) file that uses the LibQ Fixed-Point Library should include "[libq.h](#)".

Library File:

The LibQ Fixed-Point Math Library archive (.a) file installed with MPLAB Harmony.

5.4.2.4.1 Library Overview

The LibQ Fixed-Point Math Library contains functions for manipulating Q15, Q31 and other intermediate integer representations of real numbers. Library Interface details the operation of the data formats and explains each function in detail.

Signed fixed-point types are defined as follows:

`Qn_m` where:

- *n* is the number of data bits to the left of the radix point
- *m* is the number of data bits to the right of the radix point
- a signed bit is implied

For convenience, short names are also defined:

Exact Name	Number of Bits Required	Short Name
<u>Q0_15</u>	16	<u>Q15</u>
<u>Q15_16</u>	32	<u>Q16</u>
<u>Q0_31</u>	32	<u>Q31</u>

Q_n_m numerical values are used by the library processing data as integers. In this format the n represents the number of integer bits, and the m represents the number of fractional bits. All values assume a sign bit in the most significant bit. The range of the numerical value therefore is:

$$-2^{(n-1)} \text{ to } [2^{(n-1)} - 2^{(-m)}]; \text{ with a resolution of } 2^{(-m)}.$$

A Q16 format number (Q15_16) would range from -32768.0 (0x8000 0000) to 32767.99998474 with a precision of 0.000015259 (or 2^{-16}).

For example, a numerical representation of the number 3.14159 in Q2_13 notation would be:

$$3.14159 * 2^{13} = 25735.9 \Rightarrow 0x6488$$

And converting from the Q7_8 format with the value 0x1D89 would be:

$$0x1D89 / 2^8 = 7561 / 256 \Rightarrow 29.5316, \text{ accurate to } 0.00391$$

Functions in the library are prefixed with the type of the return value. For example, LIBQ_Q16Sqrt returns a Q16 value equal to the square root of its argument. Argument types do not always match the return type. Refer to the function prototype for a specification of its arguments.

In cases where the return value is not a fixed-point type, the argument type is appended to the function name. For example, LIBQ_ToFloatQ31 accepts a type Q31 argument.

In some cases, both the return type and the argument type are specified within the function name. For example:

Function Name	Return Type	Argument Type
<u>LIBQ_Q15_sin_Q2_13</u>	<u>Q15</u>	<u>Q2_13</u>
<u>LIBQ_Q31_sin_Q2_29</u>	<u>Q31</u>	<u>Q2_29</u>

5.4.2.4.2 Table of Library Functions

Math Function	Function Definition
Divide	<u>Q16</u> <u>LIBQ_Q16Div</u> (<u>Q16</u> dividend, <u>Q16</u> divisor);
Square Root	<u>Q16</u> <u>LIBQ_Q16Sqrt</u> (<u>Q16</u> x);
Exponential	<u>Q16</u> <u>LIBQ_Q16Exp</u> (<u>Q16</u> x);
Log	<u>Q4_11</u> <u>LIBQ_Q4_11_ln_Q16</u> (<u>Q16</u> x); <u>Q3_12</u> <u>LIBQ_Q3_12_log10_Q16</u> (<u>Q16</u> x); <u>Q5_10</u> <u>LIBQ_Q5_10_log2_Q16</u> (<u>Q16</u> x);
Power	<u>Q16</u> <u>LIBQ_Q16Power</u> (<u>Q16</u> x, <u>Q16</u> y);
Sine	<u>Q15</u> <u>LIBQ_Q15_sin_Q2_13</u> (<u>Q2_13</u> x); <u>Q31</u> <u>LIBQ_Q31_sin_Q2_29</u> (<u>Q2_29</u> x);

Cosine	<code>__Q15_LIBQ_Q15_cos_Q2_13 (__Q2_13 x);</code> <code>__Q31_LIBQ_Q31_cos_Q2_29 (__Q2_29 x);</code>
Tangent	<code>__Q7_8_LIBQ_Q7_8_tan_Q2_13 (__Q2_13 x);</code> <code>__Q16_LIBQ_Q16_tan_Q2_29 (__Q2_29 x);</code>
Arcsin	<code>__Q2_13_LIBQ_Q2_13_asin_Q15 (__Q15 x);</code> <code>__Q2_29_LIBQ_Q2_29_asin_Q31 (__Q31 x);</code> <code>__Q2_29_LIBQ_Q2_29_asin_Q31_Fast (__Q31 x);</code>
Arccos	<code>__Q2_13_LIBQ_Q2_13_acos_Q15 (__Q15 x);</code> <code>__Q2_29_LIBQ_Q2_29_acos_Q31 (__Q31 x);</code> <code>__Q2_29_LIBQ_Q2_29_acos_Q31_Fast (__Q31 x);</code>
Arctan	<code>__Q2_13_LIBQ_Q2_13_atan_Q7_8 (__Q7_8 x);</code> <code>__Q2_29_LIBQ_Q2_29_atan_Q16 (__Q16 x);</code>
Arctan2	<code>__Q2_13_LIBQ_Q2_13_atan2_Q7_8 (__Q7_8 y, __Q7_8 x);</code> <code>__Q2_29_LIBQ_Q2_29_atan2_Q16 (__Q16 y, __Q16 x);</code>
Random Number	<code>__Q15_LIBQ_Q15Rand (int64_t *pSeed);</code> <code>__Q31_LIBQ_Q31Rand (int64_t *pSeed);</code>
Float	<code>float __LIBQ_ToFloatQ31 (__Q31 x);</code> <code>float __LIBQ_ToFloatQ15 (__Q15 x);</code> <code>__Q31_LIBQ_Q31FromFloat (float x);</code> <code>__Q15_LIBQ_Q15FromFloat (float x);</code>
String	<code>void __LIBQ_ToStringQ15 (__Q15 x, char *s);</code> <code>__Q15_LIBQ_Q15FromString (char *s);</code>

5.4.2.5 Library Interface

Data Types and Constants

	Name	Description
	<code>__Q15_MAX</code>	Maximum value of <code>__Q15</code> (~1.0)
	<code>__Q15_MIN</code>	Minimum value of <code>__Q15</code> (-1.0)
	<code>__Q16_MAX</code>	Maximum value of <code>__Q16</code> (~32768.0)
	<code>__Q16_MIN</code>	Minimum value of <code>__Q16</code> (-32768.0)
	<code>__Q2_13_MAX</code>	Maximum value of <code>__Q2_13</code> (~4.0)
	<code>__Q2_13_MIN</code>	Minimum value of <code>__Q2_13</code> (-4.0)
	<code>__Q2_29_MAX</code>	Maximum value of <code>__Q2_29</code> (~4.0)
	<code>__Q2_29_MIN</code>	Minimum value of <code>__Q2_29</code> (-4.0)
	<code>__Q3_12_MAX</code>	Maximum value of <code>__Q3_12</code> (~8.0)
	<code>__Q3_12_MIN</code>	Minimum value of <code>__Q3_12</code> (-8.0)
	<code>__Q31_MAX</code>	Maximum value of <code>__Q31</code> (~1.0)
	<code>__Q31_MIN</code>	Minimum value of <code>__Q31</code> (-1.0)
	<code>__Q4_11_MAX</code>	Maximum value of <code>__Q4_11</code> (~16.0)
	<code>__Q4_11_MIN</code>	Minimum value of <code>__Q4_11</code> (-16.0)
	<code>__Q5_10_MAX</code>	Maximum value of <code>__Q5_10</code> (~32.0)
	<code>__Q5_10_MIN</code>	Minimum value of <code>__Q5_10</code> (-32.0)

	_Q7_8_MAX	Maximum value of _Q7_8 (~128.0)
	_Q7_8_MIN	Minimum value of _Q7_8 (-128.0)
	_Q0_15	1 sign bit, 15 bits right of radix
	_Q0_31	1 sign bit, 31 bits right of radix
	_Q15	Short name for _Q0_15
	_Q15_16	1 sign bit, 15 bits left of radix, 16 bits right of radix
	_Q16	Short name for _Q15_16
	_Q2_13	1 sign bit, 2 bits left of radix, 13 bits right of radix
	_Q2_29	1 sign bit, 2 bits left of radix, 29 bits right of radix
	_Q3_12	1 sign bit, 3 bits left of radix, 12 bits right of radix
	_Q31	Short name for _Q_0_31
	_Q4_11	1 sign bit, 4 bits left of radix, 11 bits right of radix
	_Q5_10	1 sign bit, 5 bits left of radix, 10 bits right of radix
	_Q7_8	1 sign bit, 7 bits left of radix, 8 bits right of radix

Arccos Functions

	Name	Description
	_LIBQ_Q2_13_acos_Q15	Calculates the value of $\text{acos}(x)$.
	_LIBQ_Q2_29_acos_Q31	Calculates the value of $\text{acos}(x)$.
	_LIBQ_Q2_29_acos_Q31_Fast	Calculates the value of $\text{acos}(x)$. This function executes faster than _LIBQ_Q2_29_acos_Q31 but is less precise.

Arcsin Functions

	Name	Description
	_LIBQ_Q2_13_asin_Q15	Calculates the asin value of $\text{asin}(x)$.
	_LIBQ_Q2_29_asin_Q31	Calculates the value of $\text{asin}(x)$.
	_LIBQ_Q2_29_asin_Q31_Fast	Calculates the value of $\text{asin}(x)$. This function executes faster than the _LIBQ_Q2_29_asin_Q31 function, but is less precise.

Arctan Functions

	Name	Description
	_LIBQ_Q2_13_atan_Q7_8	Calculates the value of $\text{atan}(x)$.
	_LIBQ_Q2_29_atan_Q16	Calculates the value of $\text{atan}(x)$.

Arctan2 Functions

	Name	Description
	_LIBQ_Q2_13_atan2_Q7_8	Calculates the value of $\text{atan2}(y, x)$.
	_LIBQ_Q2_29_atan2_Q16	Calculates the value of $\text{atan2}(y, x)$.

Cosine Functions

	Name	Description
	_LIBQ_Q15_cos_Q2_13	Calculates the value of $\text{cosine}(x)$.
	_LIBQ_Q31_cos_Q2_29	Calculates the value of $\text{cosine}(x)$.

Divide Functions

	Name	Description
	_LIBQ_Q16Div	_Q16 fixed point divide.

Exponential Functions

	Name	Description
	<u>_LIBQ_Q16Exp</u>	Calculates the exponential function e^x .

Float Functions

	Name	Description
	<u>_LIBQ_Q15FromFloat</u>	Converts a float to a <u>_Q15</u> value.
	<u>_LIBQ_Q31FromFloat</u>	Converts a float to a <u>_Q31</u> value.
	<u>_LIBQ_ToFloatQ15</u>	Converts a <u>_Q15</u> value to a float.
	<u>_LIBQ_ToFloatQ31</u>	Converts a <u>_Q31</u> value to a float.

Log Functions

	Name	Description
	<u>_LIBQ_Q3_12_log10_Q16</u>	Calculates the value of $\text{Log}_{10}(x)$.
	<u>_LIBQ_Q4_11_ln_Q16</u>	Calculates the natural logarithm $\ln(x)$.
	<u>_LIBQ_Q5_10_log2_Q16</u>	Calculates the value of $\log_2(x)$.

Power Functions

	Name	Description
	<u>_LIBQ_Q16Power</u>	Calculates the value of x raised to the y power (x^y).

Random Number Functions

	Name	Description
	<u>_LIBQ_Q15Rand</u>	Generate a <u>_Q15</u> random number.
	<u>_LIBQ_Q31Rand</u>	Generate a <u>_Q31</u> random number.

Sine Functions

	Name	Description
	<u>_LIBQ_Q15_sin_Q2_13</u>	Calculates the value of $\sin(x)$.
	<u>_LIBQ_Q31_sin_Q2_29</u>	Calculates the value of $\sin(x)$.

Square Root Functions

	Name	Description
	<u>_LIBQ_Q16Sqrt</u>	Square root of a positive <u>_Q16</u> fixed point value.

String Functions

	Name	Description
	<u>_LIBQ_Q15FromString</u>	Ascii to <u>_Q15</u> conversion.
	<u>_LIBQ_ToStringQ15</u>	<u>_Q15</u> to ascii conversion.

Target Functions

	Name	Description
	<u>_LIBQ_Q16_tan_Q2_29</u>	Calculates the value of $\tan(x)$.
	<u>_LIBQ_Q7_8_tan_Q2_13</u>	Calculates the value of $\tan(x)$.

Description

This section describes the Application Programming Interface (API) functions, macros, and types of the LibQ Fixed Point Math Library.

Refer to each section for a detailed description.

5.4.2.5.1 Divide Functions

5.4.2.5.1.1 _LIBQ_Q16Div Function

C

```
_Q16 _LIBQ_Q16Div(
    _Q16 dividend,
    _Q16 divisor
);
```

Description

Function _LIBQ_Q16Div:

_Q16 _LIBQ_Q16Div (_Q16 dividend, _Q16 divisor);

Quotient (_Q16) = Dividend (_Q16) / Divisor (_Q16).

Preconditions

Divisor must not equal 0.

Parameters

Parameters	Description
dividend	The divide operation dividend (<u>_Q16</u>)
divisor	The divide operation divisor (<u>_Q16</u>)

Returns

_Q16 quotient of the divide operation

Remarks

The _LIBQ_Q16Div operation saturates its result.

Execution Time (cycles): 143 typical (80 to 244) Program Memory 204 bytes

Error <= 0.000015258789 (accurate to least significant _Q16 bit within the non-saturated range)

Example

```
_Q16 quotient, dividend, divisor;

dividend = (_Q16)0x00010000; // 1
divisor  = (_Q16)0x00008000; // 0.5

quotient = _LIBQ_Q16Div (dividend, divisor);

// quotient now equals 2; ie. (_Q16)0x00020000;
```

5.4.2.5.2 Square Root Functions

5.4.2.5.2.1 _LIBQ_Q16Sqrt Function

C

```
_Q16 _LIBQ_Q16Sqrt(
    _Q16 x
);
```

Description

Function `_LIBQ_Q16Sqrt`:

`_Q16 _LIBQ_Q16Sqrt(_Q16 x);`

Calculate the square root of a positive `_Q16` fixed point value, and return the `_Q16` result.

Preconditions

The input value must be positive.

Parameters

Parameters	Description
x	The <code>_Q16</code> fixed point value input from which to find the square root.

Returns

`_LIBQ_Q16Sqrt` returns the `_Q16` fixed point value which is the square root of the input parameter.

Remarks

Execution Time (cycles): 240 typical (104 to 258) Program Memory 152 bytes

Error <= 0.000015258789 (accurate to least significant `_Q16` bit)

Example

```
_Q16 squareRoot;

squareRoot = _LIBQ_Q16Sqrt((_Q16)0x01000000); // The square root of 256.0 is 16.0 (0x00100000)
squareRoot = _LIBQ_Q16Sqrt((_Q16)0x00004000); // The square root of 0.25 is 0.5 (0x00008000)
squareRoot = _LIBQ_Q16Sqrt((_Q16)0x5851f42d); // The square root of 22609.953125 is
150.366074 (0x00965db7)
```

5.4.2.5.3 Log Functions

5.4.2.5.3.1 _LIBQ_Q3_12_log10_Q16 Function

C

```
_Q3_12 _LIBQ_Q3_12_log10_Q16(
    _Q16 x
);
```

Description

Function `_LIBQ_Q3_12_log10_Q16`:

`_Q3_12 _LIBQ_Q3_12_log10_Q16 (_Q16 x);`

Calculates the $\log_{10}(x)$, where $\log_{10}(x) = \ln(x) * \log_{10}(e)$. x is of type `_Q16` and must be positive. The resulting value is of type `_Q3_12`.

Preconditions

The input x must be positive.

Parameters

Parameters	Description
x	The input value from which to calculate $\log_{10}(x)$.

Returns

`_LIBQ_Q3_12_log10_Q16` returns the `_Q3_12` fixed point result from the calculation $\log_{10}(x)$.

Remarks

Execution Time (cycles): 301 typical (14 to 346) Program Memory 176 bytes

Error ≤ 0.000244140625 (accurate to least significant `_Q3_12` bit)

Example

```
_Q3_12 resultLog10;

resultLog10 = _LIBQ_Q3_12_log10_Q16 ((_Q16)0x12ed7d91); // _LIBQ_Q3_12_log10_Q16(4845.490494)
= 3.685303 (0x3af7)
```

5.4.2.5.3.2 _LIBQ_Q4_11_In_Q16 Function**C**

```
_Q4_11 _LIBQ_Q4_11_ln_Q16(
    _Q16 x
);
```

Description

Function `_LIBQ_Q4_11_ln_Q16`:

`_Q4_11 _LIBQ_Q4_11_ln_Q16 (_Q16 x)`;

Calculates the natural logarithm $\ln(x)$. x is of type `_Q16` and must be positive. The resulting value is of type `_Q4_11`.

Preconditions

The input x must be positive.

Parameters

Parameters	Description
x	The input value from which to calculate $\ln(x)$.

Returns

`_LIBQ_Q4_11_ln_Q16` returns the `_Q4_11` fixed point result from the calculation $\ln(x)$.

Remarks

Execution Time (cycles): 301 typical (14 to 346) Program Memory 176 bytes

Error ≤ 0.00048828 (accurate to least significant `_Q4_11` bit)

Example

```
_Q4_11 resultLN;

resultLN = _LIBQ_Q4_11_ln_Q16 ((_Q16)0x00004000); // _LIBQ_Q4_11_ln_Q16(0.250000) = -1.386230
(0xf4e9)
```

5.4.2.5.3.3 _LIBQ_Q5_10_log2_Q16 Function**C**

```
_Q5_10 _LIBQ_Q5_10_log2_Q16(
    _Q16 x
);
```

Description

Function `_LIBQ_Q5_10_log2_Q16`:

```
_Q5_10 _LIBQ_Q5_10_log2_Q16 (_Q16 x);
```

Calculates the $\log_2(x)$, where $\log_2(x) = \ln(x) * \log_2(e)$. x is of type Q16 and must be positive. The resulting value is of type Q5_10.

Preconditions

The input x must be positive.

Parameters

Parameters	Description
x	The input value from which to calculate $\log_2(x)$.

Returns

_LIBQ_Q5_10_log2_Q16 returns the Q5_10 fixed point result from the calculation $\log_2(x)$.

Remarks

Execution Time (cycles): 227 typical (14 to 268) Program Memory 164 bytes

Error ≤ 0.0009765625 (accurate to least significant Q5_10 bit)

Example

```
_Q5_10 resultLog2;

resultLog2 = _LIBQ_Q5_10_log2_Q16 ((_Q16)0x40000000); // _LIBQ_Q5_10_log2_Q16(16384.000000) =
14.000000 (0x3800)
```

5.4.2.5.4 Power Functions

5.4.2.5.4.1 _LIBQ_Q16Power Function

C

```
_Q16 _LIBQ_Q16Power(
    _Q16 x,
    _Q16 y
);
```

Description

Function _LIBQ_Q16Power:

```
_Q16 _LIBQ_Q16Power (_Q16 x, _Q16 y);
```

Calculates the x raised to the y power. Both x and y are of type Q16. x must be positive. The calculation will saturate if the resulting value is outside the range of the Q16 representation.

Preconditions

x must be positive.

Parameters

Parameters	Description
x	The <u>Q16</u> input value x from which to calculate x raised to the y.
y	The <u>Q16</u> input value y from which to calculate x raised to the y.

Returns

_LIBQ_Q16Power returns the Q16 fixed point result from the calculation x raised to the y.

Remarks

Execution Time (cycles): 882 typical (586 to 1042) Program Memory 1038 bytes

Error <= 0.000015258789 (accurate to least significant Q16 bit within the non-saturated range)

Example

```
_Q16 resultPower;

resultPower = _LIBQ_Q16Power ((_Q16)0x00020000, (_Q16)0xffff0000); //
_LIBQ_Q16Power(2.000000, -1.000000) = 0.500000 (0x00008000)
```

5.4.2.5.5 Exponential Functions

5.4.2.5.5.1 _LIBQ_Q16Exp Function

C

```
_Q16 _LIBQ_Q16Exp(
    _Q16 x
);
```

Description

Function _LIBQ_Q16Exp:

Q16 _LIBQ_Q16Exp(Q16 x);

Calculates the exponential function e^x . The calculation will saturate if the resulting value is outside the range of the Q16 representation. For $x > 10.3972015380859375$, the resulting value will be saturated to 0x7ffffff. For $x < -10.3972015380859375$ the resulting value will be saturated to 0.

Preconditions

None.

Parameters

Parameters	Description
x	The exponent value

Returns

_LIBQ_Q16Exp returns the Q16 fixed point result from the calculation e^x .

Remarks

The function _LIBQ_Q16Div is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 170 typical (18 to 292) Program Memory 446 bytes

Error <= 0.000015258789 (accurate to least significant Q16 bit within the non-saturated range)

Example

```
_Q16 expResult;

expResult = _LIBQ_Q16Exp((_Q16)0x00010000); // _LIBQ_Q16Exp(1.000000) = 2.718277 (0x0002b7e1)
```

5.4.2.5.6 Sine Functions

5.4.2.5.6.1 _LIBQ_Q15_sin_Q2_13 Function

C

```
_Q15 _LIBQ_Q15_sin_Q2_13(
    _Q2_13 x
);
```

Description

Function _LIBQ_Q15_sin_Q2_13:

Q15 _LIBQ_Q15_sin_Q2_13 (Q2_13 x);

Calculates the sine(x), where x is of type Q2_13 radians and the resulting value is of type Q15.

Preconditions

None.

Parameters

Parameters	Description
x	The <u>Q2_13</u> input value from which to calculate sine(x).

Returns

_LIBQ_Q15_sin_Q2_13 returns the Q15 fixed point result from the calculation sine(x).

Remarks

Execution Time (cycles): 100 typical (100 to 102) Program Memory 220 bytes

Error <= 0.00003052 (accurate to least significant Q15 bit)

Example

```
_Q15 resultSin;

resultSin = _LIBQ_Q15_sin_Q2_13 ((_Q2_13)0x4093); // _LIBQ_Q15_sin_Q2_13(2.017944) = 0.901672
(0x736a)
```

5.4.2.5.6.2 _LIBQ_Q31_sin_Q2_29 Function

C

```
_Q31 _LIBQ_Q31_sin_Q2_29(
    _Q2_29 x
);
```

Description

Function _LIBQ_Q31_sin_Q2_29:

Q31 _LIBQ_Q31_sin_Q2_29 (Q2_29 x);

Calculates the sine(x), where x is of type Q2_29 radians and the resulting value is of type Q31.

Preconditions

None.

Parameters

Parameters	Description
x	The <u>Q2_29</u> input value from which to calculate sine(x).

Returns

`_LIBQ_Q31_sin_Q2_29` returns the `_Q31` fixed point result from the calculation `sine(x)`.

Remarks

Execution Time (cycles): 246 typical (244 to 266) Program Memory 598 bytes

Error <= 0.00000000047 (accurate to least significant `_Q31` bit)

Example

```
_Q31 resultSin;

resultSin = _LIBQ_Q31_sin_Q2_29 ((_Q2_29)0x5a637cfe); // _LIBQ_Q31_sin_Q2_29( 2.824644562) =
0.311668121 (0x27e4bdb1)
```

5.4.2.5.7 Cosine Functions

5.4.2.5.7.1 _LIBQ_Q15_cos_Q2_13 Function

C

```
_Q15 _LIBQ_Q15_cos_Q2_13(
    _Q2_13 x
);
```

Description

Function `_LIBQ_Q15_cos_Q2_13`:

`_Q15 _LIBQ_Q15_cos_Q2_13 (_Q2_13 x);`

Calculates the `cosine(x)`, where `x` is of type `_Q2_13` radians and the resulting value is of type `_Q15`.

Preconditions

None

Parameters

Parameters	Description
<code>x</code>	The <code>_Q2_13</code> input value from which to calculate <code>cosine(x)</code> .

Returns

`_LIBQ_Q15_cos_Q2_13` returns the `_Q15` fixed point result from the calculation `cosine(x)`.

Remarks

Execution Time (cycles): 102 cycles Program Memory 224 bytes

Error <= 0.00003052 (accurate to least significant `_Q15` bit)

Example

```
_Q15 resultCos;

resultCos = _LIBQ_Q15_cos_Q2_13 ((_Q2_13)0x2171); // _LIBQ_Q15_cos_Q2_13(1.045044) = 0.501862
(0x403d)
```

5.4.2.5.7.2 _LIBQ_Q31_cos_Q2_29 Function

C

```
_Q31 _LIBQ_Q31_cos_Q2_29(
    _Q2_29 x
```

```
);
```

Description

Function `_LIBQ_Q31_cos_Q2_29`:

```
_Q31 _LIBQ_Q31_cos_Q2_29 (_Q2_29 x);
```

Calculates the cosine(x), where x is of type `_Q2_29` radians and the resulting value is of type `_Q31`.

Preconditions

None.

Parameters

Parameters	Description
x	The <code>_Q2_29</code> input value from which to calculate cosine(x).

Returns

`_LIBQ_Q31_cos_Q2_29` returns the `_Q31` fixed point result from the calculation sine(x).

Remarks

Execution Time (cycles): 265 typical (22 to 288) Program Memory 746 bytes

Error <= 0.00000000047 (accurate to least significant `_Q31` bit)

Example

```
_Q31 resultCos;

resultCos = _LIBQ_Q31_cos_Q2_29 ((_Q2_29)0x07e2e1c2); // _LIBQ_Q31_cos_Q2_29( 0.246445540) =
0.969785686 (0x7c21eff7)
```

5.4.2.5.8 Target Functions

5.4.2.5.8.1 _LIBQ_Q16_tan_Q2_29 Function

C

```
_Q16 _LIBQ_Q16_tan_Q2_29(
_Q2_29 x
);
```

Description

Function `_LIBQ_Q16_tan_Q2_29`:

```
_Q16 _LIBQ_Q16_tan_Q2_29 (_Q2_29 x);
```

Calculates the tan(x), where x is of type `_Q2_29` radians and the resulting value is of type `_Q16`.

Preconditions

None

Parameters

Parameters	Description
x	The <code>_Q2_29</code> input value from which to calculate tan(x).

Returns

`_LIBQ_Q16_tan_Q2_29` returns the `_Q16` fixed point result from the calculation tan(x). The resulting value is saturated.

Remarks

The functions `_LIBQ_Q31_sin_Q2_29`, `_LIBQ_Q31_cos_Q2_29`, and `_LIBQ_Q16Div` are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 703 typical (22 to 796) Program Memory 88 bytes

Error <= 0.000015259 (accurate to least significant `_Q16` bit for the input range -1.568 .. 1.568) Error rises from 0.0 to 0.065 for the input range -1.568 .. -1.570765808 and 1.568 .. 1.570765808)

Example

```
_Q16 resultTan;

resultTan = _LIBQ_Q16_tan_Q2_29 ((_Q2_29)0x16720c36); // _LIBQ_Q16_tan_Q2_29( 0.701421838) =
0.844726562 (0x0000d840)
```

5.4.2.5.8.2 _LIBQ_Q7_8_tan_Q2_13 Function**C**

```
_Q7_8 _LIBQ_Q7_8_tan_Q2_13(
    _Q2_13 x
);
```

Description

Function `_LIBQ_Q7_8_tan_Q2_13`:

`_Q7_8 _LIBQ_Q7_8_tan_Q2_13 (_Q2_13 x);`

Calculates the tan(x), where x is of type `_Q2_13` radians and the resulting value is of type `_Q7_8`.

Preconditions

None

Parameters

Parameters	Description
x	The <code>_Q2_13</code> input value from which to calculate tan(x).

Returns

`_LIBQ_Q7_8_tan_Q2_13` returns the `_Q7_8` fixed point result from the calculation tan(x).

Remarks

Execution Time (cycles): 288 typical (18 to 346) Program Memory 980 bytes

Error <= 0.00390625 (accurate to least significant `_Q7_8` bit)

Example

```
_Q7_8 resultTan;

resultTan = _LIBQ_Q7_8_tan_Q2_13 ((_Q2_13)0x2e20); // _LIBQ_Q7_8_tan_Q2_13(1.441406) =
7.683594 (0x07af)
```

5.4.2.5.9 Arcsin Functions**5.4.2.5.9.1 _LIBQ_Q2_13_asin_Q15 Function****C**

```
_Q2_13 _LIBQ_Q2_13_asin_Q15(
```

```
    _Q15 x
);
```

Description

Function `_LIBQ_Q2_13_asin_Q15`:

`_Q2_13` `_LIBQ_Q2_13_asin_Q15` (`_Q15` x);

Calculates `asin(x)`, where x is of type `_Q15` and the resulting value is of type `_Q2_13`. The output value will be radians in the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
x	The <code>_Q15</code> input value from which to calculate <code>asin(x)</code> .

Returns

`_LIBQ_Q2_13_asin_Q15` returns the `_Q2_13` fixed point result from the calculation `asin(x)`.

Remarks

The functions `_LIBQ_Q16Sqrt` and `_LIBQ_Q16Div` are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 578 typical (22 to 656) Program Memory 336 bytes

Error ≤ 0.00012207 (accurate to least significant `_Q2_13` bit)

A higher resolution version of this function exists with equivalent performance, see `_LIBQ_Q2_29_asin_Q31_Fast`

Example

```
_Q2_13 resultAsin;

resultAsin = _LIBQ_Q2_13_asin_Q15 ((_Q15)0x3231); // _LIBQ_Q2_13_asin_Q15(0.392120) =
0.402954 (0x0ce5)
```

5.4.2.5.9.2 `_LIBQ_Q2_29_asin_Q31` Function

C

```
_Q2_29 _LIBQ_Q2_29_asin_Q31(
    _Q31 x
);
```

Description

Function `_LIBQ_Q2_29_asin_Q31`:

`_Q2_29` `_LIBQ_Q2_29_asin_Q31` (`_Q31` x);

Calculates the `asin(x)`, where x is of type `_Q31` and the resulting value is of type `_Q2_29`. The output value will be in radians the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
x	The <code>_Q31</code> input value from which to calculate <code>asin(x)</code> .

Returns

`_LIBQ_Q2_29_asin_Q31` returns the `_Q2_29` fixed point result from the calculation `asin(x)`.

Remarks

The functions `_LIBQ_Q2_29_asin_Q31_Fast` and `_LIBQ_Q31_sin_Q2_29` are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 2525 typical (286 to 4330) Program Memory 138 bytes

Error ≤ 0.0000000019 (accurate to least significant `_Q2_29` bit for the range -0.9993..0.9993) Error ≤ 0.0000000346 (accurate to 5th least significant `_Q2_29` bit for the range -1.0 .. -0.9993 and 0.9993 .. 1.0)

A faster version of this function exists with modestly reduced accuracy, see `_LIBQ_Q2_29_asin_Q31_Fast`

Example

```
_Q2_29 resultAsin;

resultAsin = _LIBQ_Q2_29_asin_Q31 ((_Q31)0x7fe50658); // _LIBQ_Q2_29_asin_Q31( 0.9991767816)
= 1.5302172359 (0x30f78a23)
```

5.4.2.5.9.3 _LIBQ_Q2_29_asin_Q31_Fast Function**C**

```
_Q2_29 _LIBQ_Q2_29_asin_Q31_Fast(
    _Q31 x
);
```

Description

Function `_LIBQ_Q2_29_asin_Q31_Fast`:

`_Q2_29` `_LIBQ_Q2_29_asin_Q31_Fast` (`_Q31` x);

Calculates the `asin(x)`, where x is of type `_Q31` and the resulting value is of type `_Q2_29`. The output value will be in radians the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
x	The <code>_Q31</code> input value from which to calculate <code>asin(x)</code> .

Returns

`_LIBQ_Q2_29_asin_Q31_Fast` returns the `_Q2_29` fixed point result from the calculation `asin(x)`.

Remarks

Execution Time (cycles): 507 typical (22 to 1300) Program Memory 638 bytes

Error ≤ 0.000000911 (accurate to 9 least significant `_Q2_29` bits)

A higher resolution version of this function exists with reduced performance, see `_LIBQ_Q2_29_asin_Q31`

Example

```
_Q2_29 resultAsin;

resultAsin = _LIBQ_Q2_29_asin_Q31_Fast ((_Q31)0x7fe50658); // _LIBQ_Q2_29_asin_Q31_Fast(
0.9991767816) = 1.5302172359 (0x30f78a23)
```

5.4.2.5.10 Arccos Functions

5.4.2.5.10.1 _LIBQ_Q2_13_acos_Q15 Function

C

```
_Q2_13 _LIBQ_Q2_13_acos_Q15(
    _Q15 x
);
```

Description

Function _LIBQ_Q2_13_acos_Q15:

[_Q2_13](#) _LIBQ_Q2_13_acos_Q15 ([_Q15](#) x);

Calculates the acos(x), where x is of type [_Q15](#) and the resulting value is of type [_Q2_13](#). The output value will be radians in the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
x	The _Q15 input value from which to calculate acos(x).

Returns

_LIBQ_Q2_13_acos_Q15 returns the [_Q2_13](#) fixed point result from the calculation acos(x).

Remarks

The function [_LIBQ_Q2_13_asin_Q15](#) is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 588 typical (32 to 666) Program Memory 24 bytes

Error ≤ 0.00012207 (accurate to least significant [_Q2_13](#) bit)

A higher precision function with equivalent performance exists, see [_LIBQ_Q2_29_acos_Q31_Fast](#)

Example

```
_Q2_13 resultAcos;

resultAcos = _LIBQ_Q2_13_acos_Q15((_Q15)0x2993); // \_LIBQ\_Q2\_13\_acos\_Q15\(0.324799\) = 1.239990
(0x27ae)
```

5.4.2.5.10.2 _LIBQ_Q2_29_acos_Q31 Function

C

```
_Q2_29 _LIBQ_Q2_29_acos_Q31(
    _Q31 x
);
```

Description

Function _LIBQ_Q2_29_acos_Q31:

[_Q2_29](#) _LIBQ_Q2_29_acos_Q31 ([_Q31](#) x);

Calculates the acos(x), where x is of type [_Q31](#) and the resulting value is of type [_Q2_29](#). The output value will be radians in the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
x	The _Q31 input value from which to calculate acos(x).

Returns

[_LIBQ_Q2_29_acos_Q31](#) returns the [_Q2_29](#) fixed point result from the calculation acos(x).

Remarks

The functions [_LIBQ_Q2_29_asin_Q31_Fast](#) and [_LIBQ_Q31_cos_Q2_29](#) are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 3370 typical (70 to 4824) Program Memory 142 bytes

Error <= 0.0000000019 (accurate to least significant [_Q2_29](#) bit for the range -0.9993..0.9993) Error <= 0.0000000355 (accurate to 5th least significant [_Q2_29](#) bit for the range -1.0 .. -0.9993 and 0.9993 .. 1.0)

A similar function with higher performance and reduced precision exists, see [_LIBQ_Q2_29_acos_Q31_Fast](#)

Example

```
_Q2_29 resultAcos;

resultAcos = _LIBQ_Q2_29_acos_Q31 ((_Q31)0xee63708c); // \_LIBQ\_Q2\_29\_acos\_Q31\(-0.1375903431\)
= 1.7088244837 \(0x36aeb0af\)
```

5.4.2.5.10.3 [_LIBQ_Q2_29_acos_Q31_Fast](#) Function**C**

```
_Q2_29 \_LIBQ\_Q2\_29\_acos\_Q31\_Fast(
    _Q31 x
);
```

Description

Function [_LIBQ_Q2_29_acos_Q31_Fast](#):

[_Q2_29](#) [_LIBQ_Q2_29_acos_Q31_Fast](#) ([_Q31](#) x);

Calculates the acos(x), where x is of type [_Q31](#) and the resulting value is of type [_Q2_29](#). The output value will be radians in the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
x	The _Q31 input value from which to calculate acos(x).

Returns

[_LIBQ_Q2_29_acos_Q31_Fast](#) returns the [_Q2_29](#) fixed point result from the calculation acos(x).

Remarks

The function [_LIBQ_Q2_29_asin_Q31_Fast](#) is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 517 typical (32 to 1310) Program Memory 28 bytes

Error <= 0.000000911 (accurate to 9 least significant Q2_29 bits)

A higher precision function with reduced performance exists, see LIBQ_Q2_29_acos_Q31

Example

```
_Q2_29 resultAcos;

resultAcos = _LIBQ_Q2_29_acos_Q31_Fast ((_Q31)0xee63708c); //
_LIBQ_Q2_29_acos_Q31_Fast(-0.1375903431) = 1.7088244837 (0x36aeb0af)
```

5.4.2.5.11 Arctan Functions

5.4.2.5.11.1 _LIBQ_Q2_13_atan_Q7_8 Function

C

```
_Q2_13 _LIBQ_Q2_13_atan_Q7_8(
    _Q7_8 x
);
```

Description

Function LIBQ_Q2_13_atan_Q7_8:

Q2_13 LIBQ_Q2_13_atan_Q7_8 (Q7_8 x);

Calculates the atan(x), where x is of type Q7_8 and the resulting value is of type Q2_13. The output value will be radians in the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
x	The <u>Q7_8</u> input value from which to calculate atan(x).

Returns

LIBQ_Q2_13_atan_Q7_8 returns the Q2_13 fixed point result from the calculation atan(x).

Remarks

The function LIBQ_Q2_13_atan2_Q7_8 is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 240 typical (202 to 256) Program Memory 16 bytes

Error <= 0.00012207 (accurate to least significant Q2_13 bit)

Example

```
_Q2_13 resultAtan;

resultAtan = _LIBQ_Q2_13_atan_Q7_8 ((_Q7_8)0x0097); // _LIBQ_Q2_13_atan_Q7_8(0.589844) =
0.532959 (0x110e)
```

5.4.2.5.11.2 _LIBQ_Q2_29_atan_Q16 Function

C

```
_Q2_29 _LIBQ_Q2_29_atan_Q16(
    _Q16 x
);
```

Description

Function `_LIBQ_Q2_29_atan_Q16`:

`_Q2_29 _LIBQ_Q2_29_atan_Q16 (_Q16 x);`

Calculates the $\text{atan}(x)$, where x is of type `_Q16` and the resulting value is of type `_Q2_29`. The output value will be radians in the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
x	The <code>_Q16</code> input value from which to calculate $\text{atan}(x)$.

Returns

`_LIBQ_Q2_29_atan_Q16` returns the `_Q2_29` fixed point result from the calculation $\text{atan}(x)$.

Remarks

The function `_LIBQ_Q2_29_atan2_Q16` is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 354 typical (178 to 360) Program Memory 16 bytes

Error ≤ 0.000000003 (accurate within one least significant `_Q2_29` bit)

Example

```
_Q2_29 resultAtan;
```

```
resultAtan = _LIBQ_Q2_29_atan_Q16 ((_Q16)0x00098b31); // _LIBQ_Q2_29_atan_Q16(9.543716) = 1.466396 (0x2eecb7ee)
```

5.4.2.5.12 Arctan2 Functions

5.4.2.5.12.1 _LIBQ_Q2_13_atan2_Q7_8 Function

C

```
_Q2_13 _LIBQ_Q2_13_atan2_Q7_8(
    _Q7_8 y,
    _Q7_8 x
);
```

Description

Function `_LIBQ_Q2_13_atan2_Q7_8`:

`_Q2_13 _LIBQ_Q2_13_atan2_Q7_8 (_Q7_8 y, _Q7_8 x);`

Calculates the $\text{atan2}(y, x)$, where y and x are of type `_Q7_8` and the resulting value is of type `_Q2_13`. The output value will be radians in the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
y	The <code>_Q7_8</code> input value from which to calculate $\text{atan2}(y, x)$.

x	The <u>Q7_8</u> input value from which to calculate atan2(y, x).

Returns

LIBQ_Q2_13_atan2_Q7_8 returns the Q2_13 fixed point result from the calculation atan2(y, x).

Remarks

The function LIBQ_Q16Div is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 220 typical (22 to 250) Program Memory 288 bytes

Error <= 0.00012207 (accurate to least significant Q2_13 bit)

Example

```
Q2_13 resultAtan2;

resultAtan2 = LIBQ_Q2_13_atan2_Q7_8 ((_Q7_8)0x589d, (_Q7_8)0xf878); //
LIBQ_Q2_13_atan2_Q7_8(88.613281, -7.531250) = 1.655518 (0x34fa)
```

5.4.2.5.12.2 LIBQ_Q2_29_atan2_Q16 Function**C**

```
Q2_29 LIBQ_Q2_29_atan2_Q16(
    Q16 y,
    Q16 x
);
```

Description

Function LIBQ_Q2_29_atan2_Q16:

Q2_29 LIBQ_Q2_29_atan2_Q16 (Q16 y, Q16 x);

Calculates the atan(y, x), where y and x are of type Q16 and the resulting value is of type Q2_29. The output value will be radians in the range $\pi \geq \text{result} \geq -\pi$.

Preconditions

None.

Parameters

Parameters	Description
y	The <u>Q16</u> input value from which to calculate atan2(y, x).
x	The <u>Q16</u> input value from which to calculate atan2(y, x).

Returns

LIBQ_Q2_29_atan2_Q16 returns the Q2_29 fixed point result from the calculation atan2(y, x).

Remarks

The C function `__divdi3` is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 348 typical (20 to 376) Program Memory 464 bytes

Error <= 0.000000003 (accurate within one least significant Q2_29 bit)

Example

```
Q2_29 resultAtan2;

resultAtan2 = LIBQ_Q2_29_atan2_Q16 ((_Q16)0xf6276270, x(_Q16)0x34b4b4c0); //
LIBQ_Q2_29_atan2_Q16(-2520.615479, 13492.706055) = -0.184684 (0xfa1710c7)
```


5.4.2.5.13 Random Number Functions

5.4.2.5.13.1 _LIBQ_Q15Rand Function

C

```
_Q15 _LIBQ_Q15Rand(
    int64_t * pSeed
);
```

Description

Function _LIBQ_Q15Rand:

```
_Q15 _LIBQ_Q15Rand (int64_t *pSeed);
```

Generates a _Q15 pseudo-random value based on the seed supplied as a parameter. The first time this function is called, the seed value must be supplied by the user; this initial seed value can either be constant or random, depending on whether the user wants to generate a repeatable or a non-repeatable pseudo-random sequence.

The function updates the *pSeed value each time it is called. The updated *pSeed value must be passed back to the function with each subsequent call.

Warning: The pseudo-random sequence generated by this function may be insufficient for cryptographic use.

Preconditions

None.

Parameters

Parameters	Description
pSeed	A pointer to the seed value used by the function to generate a pseudo-random sequence.

Returns

_LIBQ_Q15Rand returns a random _Q15 value. _LIBQ_Q15Rand also updates the int64_t *pSeed value.

Remarks

Execution Time (cycles): 32 Program Memory 92 bytes

Example

```
// Initialize seed to a constant or random value
static int64_t randomSeed = 0xA71078BE72D4C1F1;

_Q15 randomValue;

randomValue = _LIBQ_Q15Rand(&randomSeed);
...
randomValue = _LIBQ_Q15Rand(&randomSeed);
```

5.4.2.5.13.2 _LIBQ_Q31Rand Function

C

```
_Q31 _LIBQ_Q31Rand(
    int64_t * pSeed
);
```

Description

Function _LIBQ_Q31Rand:

```
_Q31 _LIBQ_Q31Rand (int64_t *pSeed);
```

Generates a _Q31 pseudo-random value based on the seed supplied as a parameter. The first time this function is called, the seed value must be supplied by the user; this initial seed value can either be constant or random, depending on whether the user wants to generate a repeatable or a non-repeatable pseudo-random sequence.

The function updates the *pSeed value each time it is called. The updated *pSeed value must be passed back to the function with each subsequent call.

Warning: The pseudo-random sequence generated by this function may be insufficient for cryptographic use.

Preconditions

None.

Parameters

Parameters	Description
pSeed	A pointer to the seed value used by the function to generate a pseudo-random sequence.

Returns

_LIBQ_Q31Rand returns a pseudo-random _Q31 value. _LIBQ_Q31Rand also updates the int64_t *pSeed value.

Remarks

Execution Time (cycles): 32 Program Memory 88 bytes

Example

```
// Initialize seed to a constant or random value
static int64_t randomSeed = 0x7F18BA710E72D4C1;

_Q31 randomValue;

randomValue = _LIBQ_Q31Rand(&randomSeed);
...
randomValue = _LIBQ_Q31Rand(&randomSeed);
```

5.4.2.5.14 Float Functions

5.4.2.5.14.1 _LIBQ_Q15FromFloat Function

C

```
_Q15 _LIBQ_Q15FromFloat(
    float x
);
```

Description

Function _LIBQ_Q15FromFloat:

```
_Q15 _LIBQ_Q15FromFloat(float x);
```

Converts a floating point value to a _Q15 fixed point representation. The _Q15 fixed point value is returned by the function. The conversion will saturate if the value is outside the range of the _Q15 representation.

Preconditions

None.

Parameters

Parameters	Description
x	The float point value to convert to <u>Q15</u> fixed point

Returns

`_LIBQ_Q15FromFloat` returns the Q15 fixed point value corresponding to the floating point (float) input value.

Remarks

The C library functions `__gesf2`, `__lesf2`, `__addsf3`, `__mulsf3`, and `__fixsfsi` are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 213 typical (158 to 224) Program Memory 96 bytes

Example

```
_Q15 q15;

q15 = _LIBQ_Q15FromFloat((float)0.5); // q15 now equals (_Q15)0x4000
q15 = _LIBQ_Q15FromFloat((float)-1.0); // q15 now equals (_Q15)0x8000
q15 = _LIBQ_Q15FromFloat((float)-0.233828); // q15 now equals (_Q15)0xe212
```

5.4.2.5.14.2 _LIBQ_Q31FromFloat Function**C**

```
_Q31 _LIBQ_Q31FromFloat(
    float x
);
```

Description

Function `_LIBQ_Q31FromFloat`:

Q31 `_LIBQ_Q31FromFloat(float x);`

Converts a floating point value to a Q31 fixed point representation. The Q31 fixed point value is returned by the function. The conversion will saturate if the value is outside the range of the Q31 representation.

Preconditions

None.

Parameters

Parameters	Description
x	The floating point value to convert to <u>Q31</u> fixed point.

Returns

`_LIBQ_Q31FromFloat` returns the Q31 fixed point value corresponding to the floating point (float) input value.

Remarks

The C library functions `__gesf2`, `__lesf2`, `__addsf3`, `__mulsf3`, and `__fixsfsi` are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 210 typical (158 to 214) Program Memory 100 bytes

Example

```
_Q31 q31;

q31 = _LIBQ_Q31FromFloat((float)0.000008); // q31 now equals (_Q31)0x00004000
```

```
q31 = _LIBQ_Q31FromFloat((float)-1.0); // q31 now equals (_Q31)0x80000000
q31 = _LIBQ_Q31FromFloat((float)0.690001); // q31 now equals (_Q31)0x5851f400
```

5.4.2.5.14.3 _LIBQ_ToFloatQ15 Function

C

```
float _LIBQ_ToFloatQ15(
    _Q15 x
);
```

Description

Function _LIBQ_ToFloatQ15:

```
float _LIBQ_ToFloatQ15(_Q15 x);
```

Converts a Q15 fixed point value to a floating point representation. The floating point value is returned by the function.

Preconditions

None.

Parameters

Parameters	Description
x	The <u>Q15</u> fixed point value to convert to float

Returns

_LIBQ_ToFloatQ15 returns the floating point (float) value corresponding to the Q15 input value.

Remarks

The C library functions __floatsisf and __divsf3 are called by this routine and thus must be linked in to the executable image.

Execution Time (cycles): 158 typical (54 to 176) Program Memory 28 bytes

Example

```
float f;

f = _LIBQ_ToFloatQ15((_Q15)0x4000); // f now equals 0.5
f = _LIBQ_ToFloatQ15((_Q15)0x8000); // f now equals -1.0
f = _LIBQ_ToFloatQ15((_Q15)0xb7ff); // f now equals -0.562531
```

5.4.2.5.14.4 _LIBQ_ToFloatQ31 Function

C

```
float _LIBQ_ToFloatQ31(
    _Q31 x
);
```

Description

Function _LIBQ_ToFloatQ31:

```
float _LIBQ_ToFloatQ31(_Q31 x);
```

Converts a Q31 fixed point value to a floating point representation. The floating point value is returned by the function.

Preconditions

None.

Parameters

Parameters	Description
x	The <u>Q31</u> fixed point value to convert to float

Returns

LIBQ_ToFloatQ31 returns the floating point (float) value corresponding to the Q31 input value.

Remarks

The C library functions `__floatsisf` and `__divsf3` are called by this routine and thus must be linked in to the executable image.

Execution Time (cycles): 163 typical (54 to 176) Program Memory 28 bytes

Example

```
float f;

f = _LIBQ_ToFloatQ31((_Q31)0x00004000); // f now equals 0.000008
f = _LIBQ_ToFloatQ31((_Q31)0x80000000); // f now equals -1.0
f = _LIBQ_ToFloatQ31((_Q31)0x5851f42d); // f now equals 0.690001
```

5.4.2.5.15 String Functions

5.4.2.5.15.1 LIBQ_Q15FromString Function

C

```
_Q15 _LIBQ_Q15FromString(
    char * s
);
```

Description

Function LIBQ_Q15FromString:

Q15 LIBQ_Q15FromString(char *s);

Convert an ascii string into a Q15 fixed point value. The ascii string must be in an -N.NNNNNN format. Leading spaces are ignored. The conversion stops at either the first non-conforming character in the string or the Null string terminator. There must be no spaces within the string value itself.

Preconditions

None.

Parameters

Parameters	Description
s	A pointer to the ascii input string representing the <u>Q15</u> fixed point value.

Returns

LIBQ_Q15FromString returns the Q15 fixed point value represented by the input string.

Remarks

Execution Time (cycles): 296 typical (28 to 346) Program Memory 172 bytes

Example

```
_Q15 x;
```

```
x = _LIBQ_Q15FromString("0.125");    // x will equal 0.125 using
                                     // an internal value of 0x1000

x = _LIBQ_Q15FromString("-1.0");      // x will equal -1.0 using
                                     // an internal value of 0x8000

x = _LIBQ_Q15FromString("0.999969"); // x will equal 0.999969 using
                                     // an internal value of 0x7FFF
```

5.4.2.5.15.2 **_LIBQ_ToStringQ15 Function**

```
C
void _LIBQ_ToStringQ15(
    _Q15 x,
    char * s
);
```

Description

Function `_LIBQ_ToStringQ15`:

```
void _LIBQ_ToStringQ15(_Q15 x, char *s);
```

Convert a _Q15 fixed point value to an ascii string representation in a -N.NNNNNN format.

Preconditions

The character string "s" must be at least 10 characters long, including the Null string terminator.

Parameters

Parameters	Description
x	The fixed point value to be converted into an ascii string (<u>_Q15</u>)
s	A pointer to the output string of at least 10 characters

Returns

An ascii string that represents the _Q15 fixed point value in -N.NNNNNN format. The output string will be terminated by a Null (0x00) character.

Remarks

Execution Time (cycles): 118 typical (28 to 132) Program Memory 200 bytes

Example

```
char s[10];

_LIBQ_ToStringQ15((_Q15)0x1000, s);    // s will equal "0.125000"
_LIBQ_ToStringQ15((_Q15)0x8000, s);    // s will equal "-1.000000"
_LIBQ_ToStringQ15((_Q15)0x7FFF, s);    // s will equal "0.999969"
```

5.4.2.5.16 **Data Types and Constants**

5.4.2.5.16.1 **_Q15_MAX Macro**

```
C
#define _Q15_MAX ((_Q15)0x7FFF)      // Maximum value of _Q15 (~1.0)
```

Description

Maximum value of _Q15 (~1.0)

5.4.2.5.16.2 _Q15_MIN Macro

C

```
#define _Q15_MIN ((_Q15)0x8000) // Minimum value of _Q15 (-1.0)
```

Description

Minimum value of **_Q15** (-1.0)

5.4.2.5.16.3 _Q16_MAX Macro

C

```
#define _Q16_MAX ((_Q16)0x7FFFFFFF) // Maximum value of _Q16 (~32768.0)
```

Description

Maximum value of **_Q16** (~32768.0)

5.4.2.5.16.4 _Q16_MIN Macro

C

```
#define _Q16_MIN ((_Q16)0x80000000) // Minimum value of _Q16 (-32768.0)
```

Description

Minimum value of **_Q16** (-32768.0)

5.4.2.5.16.5 _Q2_13_MAX Macro

C

```
#define _Q2_13_MAX ((_Q2_13)0x7FFF) // Maximum value of _Q2_13 (~4.0)
```

Description

Maximum value of **_Q2_13** (~4.0)

5.4.2.5.16.6 _Q2_13_MIN Macro

C

```
#define _Q2_13_MIN ((_Q2_13)0x8000) // Minimum value of _Q2_13 (-4.0)
```

Description

Minimum value of **_Q2_13** (-4.0)

5.4.2.5.16.7 _Q2_29_MAX Macro

C

```
#define _Q2_29_MAX ((_Q2_29)0x7FFFFFFF) // Maximum value of _Q2_29 (~4.0)
```

Description

Maximum value of **_Q2_29** (~4.0)

5.4.2.5.16.8 _Q2_29_MIN Macro

C

```
#define _Q2_29_MIN ((_Q2_29)0x80000000) // Minimum value of _Q2_29 (-4.0)
```

Description

Minimum value of Q2_29 (-4.0)

5.4.2.5.16.9 _Q3_12_MAX Macro

C

```
#define _Q3_12_MAX (( _Q3_12)0x7FFF) // Maximum value of _Q3_12 (~8.0)
```

Description

Maximum value of Q3_12 (~8.0)

5.4.2.5.16.10 _Q3_12_MIN Macro

C

```
#define _Q3_12_MIN (( _Q3_12)0x8000) // Minimum value of _Q3_12 (-8.0)
```

Description

Minimum value of Q3_12 (-8.0)

5.4.2.5.16.11 _Q31_MAX Macro

C

```
#define _Q31_MAX (( _Q31)0x7FFFFFFF) // Maximum value of _Q31 (~1.0)
```

Description

Maximum value of Q31 (~1.0)

5.4.2.5.16.12 _Q31_MIN Macro

C

```
#define _Q31_MIN (( _Q31)0x80000000) // Minimum value of _Q31 (-1.0)
```

Description

Minimum value of Q31 (-1.0)

5.4.2.5.16.13 _Q4_11_MAX Macro

C

```
#define _Q4_11_MAX (( _Q4_11)0x7FFF) // Maximum value of _Q4_11 (~16.0)
```

Description

Maximum value of Q4_11 (~16.0)

5.4.2.5.16.14 _Q4_11_MIN Macro

C

```
#define _Q4_11_MIN (( _Q4_11)0x8000) // Minimum value of _Q4_11 (-16.0)
```

Description

Minimum value of Q4_11 (-16.0)

5.4.2.5.16.15 _Q5_10_MAX Macro

C

```
#define _Q5_10_MAX (( _Q5_10)0x7FFF) // Maximum value of _Q5_10 (~32.0)
```

Description

Maximum value of **_Q5_10** (~32.0)

5.4.2.5.16.16 _Q5_10_MIN Macro

C

```
#define _Q5_10_MIN (( _Q5_10)0x8000) // Minimum value of _Q5_10 (-32.0)
```

Description

Minimum value of **_Q5_10** (-32.0)

5.4.2.5.16.17 _Q7_8_MAX Macro

C

```
#define _Q7_8_MAX (( _Q7_8)0x7FFF) // Maximum value of _Q7_8 (~128.0)
```

Description

Maximum value of **_Q7_8** (~128.0)

5.4.2.5.16.18 _Q7_8_MIN Macro

C

```
#define _Q7_8_MIN (( _Q7_8)0x8000) // Minimum value of _Q7_8 (-128.0)
```

Description

Minimum value of **_Q7_8** (-128.0)

5.4.2.5.16.19 _Q0_15 Type

C

```
typedef int16_t _Q0_15;
```

Description

1 sign bit, 15 bits right of radix

5.4.2.5.16.20 _Q0_31 Type

C

```
typedef int32_t _Q0_31;
```

Description

1 sign bit, 31 bits right of radix

5.4.2.5.16.21 _Q15 Type

C

```
typedef _Q0_15 _Q15;
```

Description

Short name for [_Q0_15](#)

5.4.2.5.16.22 _Q15_16 Type

C

```
typedef int32_t _Q15_16;
```

Description

1 sign bit, 15 bits left of radix, 16 bits right of radix

5.4.2.5.16.23 _Q16 Type

C

```
typedef _Q15_16 _Q16;
```

Description

Short name for [_Q15_16](#)

5.4.2.5.16.24 _Q2_13 Type

C

```
typedef int16_t _Q2_13;
```

Description

1 sign bit, 2 bits left of radix, 13 bits right of radix

5.4.2.5.16.25 _Q2_29 Type

C

```
typedef int32_t _Q2_29;
```

Description

1 sign bit, 2 bits left of radix, 29 bits right of radix

5.4.2.5.16.26 _Q3_12 Type

C

```
typedef int16_t _Q3_12;
```

Description

1 sign bit, 3 bits left of radix, 12 bits right of radix

5.4.2.5.16.27 _Q31 Type

C

```
typedef _Q0_31 _Q31;
```

Description

Short name for [_Q_0_31](#)

5.4.2.5.16.28 _Q4_11 Type

C

```
typedef int16_t _Q4_11;
```

Description

1 sign bit, 4 bits left of radix, 11 bits right of radix

5.4.2.5.16.29 _Q5_10 Type

C

```
typedef int16_t _Q5_10;
```

Description

1 sign bit, 5 bits left of radix, 10 bits right of radix

5.4.2.5.16.30 _Q7_8 Type

C

```
typedef int16_t _Q7_8;
```

Description

1 sign bit, 7 bits left of radix, 8 bits right of radix

5.4.2.6 Files

Files

Name	Description
libq.h	Optimized fixed point math functions for the PIC32MZ microAptiv™ core platform.

Description

5.4.2.6.1 libq.h

Fixed Point Library

libq.h provides a library of fixed point math functions for the PIC32MZ platform. All functions are optimized for speed. This header file specifies characteristics of each function, including execution time, memory size, and resolution.

Signed fixed point types are defined as follows:

Qn_m where:

- n is the number of data bits to the left of the radix point
- m is the number of data bits to the right of the radix point
- a signed bit is implied

For convenience, short names are also defined:

Exact Name (# Bits) Required Short Name [_Q0_15](#) (16) [_Q15](#); [_Q15_16](#) (32) [_Q16](#); [_Q0_31](#) (32) [_Q31](#)

Functions in the library are prefixed with the type of the return value. For example, `_LIBQ_Q16Sqrt` returns a Q16 value equal to the square root of its argument.

Argument types do not always match the return type. Refer to the function prototype for a specification of its arguments.

In cases where the return value is not a fixed point type, the argument type is appended to the function name. For example, `_LIBQ_ToFloatQ31` accepts a type `_Q31` argument.

In some cases, both the return type and the argument type are specified within the function name. For example,

Function Name (Return Type) [Argument Type]: `_LIBQ_Q15_sin_Q2_13 (_Q15) [_Q2_13]`; `_LIBQ_Q31_sin_Q2_29 (_Q31) [_Q2_29]`

Table of LIBQ functions:

Divide: `_Q16 _LIBQ_Q16Div (_Q16 dividend, _Q16 divisor)`;

Square root: `_Q16 _LIBQ_Q16Sqrt (_Q16 x)`;

Exponential: `_Q16 _LIBQ_Q16Exp (_Q16 x)`;

Log: `_Q4_11 _LIBQ_Q4_11_ln_Q16 (_Q16 x)`; `_Q3_12 _LIBQ_Q3_12_log10_Q16 (_Q16 x)`; `_Q5_10 _LIBQ_Q5_10_log2_Q16 (_Q16 x)`;

Power: `_Q16 _LIBQ_Q16Power (_Q16 x, _Q16 y)`;

Sine: `_Q15 _LIBQ_Q15_sin_Q2_13 (_Q2_13 x)`; `_Q31 _LIBQ_Q31_sin_Q2_29 (_Q2_29 x)`;

Cosine: `_Q15 _LIBQ_Q15_cos_Q2_13 (_Q2_13 x)`; `_Q31 _LIBQ_Q31_cos_Q2_29 (_Q2_29 x)`;

Tangent: `_Q7_8 _LIBQ_Q7_8_tan_Q2_13 (_Q2_13 x)`; `_Q16z _LIBQ_Q16_tan_Q2_29 (_Q2_29 x)`;

Arcsin: `_Q2_13 _LIBQ_Q2_13_asin_Q15 (_Q15 x)`; `_Q2_29 _LIBQ_Q2_29_asin_Q31 (_Q31 x)`; `_Q2_29 _LIBQ_Q2_29_asin_Q31_Fast (_Q31 x)`;

Arccos: `_Q2_13 _LIBQ_Q2_13_acos_Q15 (_Q15 x)`; `_Q2_29 _LIBQ_Q2_29_acos_Q31 (_Q31 x)`; `_Q2_29 _LIBQ_Q2_29_acos_Q31_Fast (_Q31 x)`;

Arctan: `_Q2_13 _LIBQ_Q2_13_atan_Q7_8 (_Q7_8 x)`; `_Q2_29 _LIBQ_Q2_29_atan_Q16 (_Q16 x)`;

Arctan2: `_Q2_13 _LIBQ_Q2_13_atan2_Q7_8 (_Q7_8 y, _Q7_8 x)`; `_Q2_29 _LIBQ_Q2_29_atan2_Q16 (_Q16 y, _Q16 x)`;

Random number: `_Q15 _LIBQ_Q15Rand (int64_t *pSeed)`; `_Q31 _LIBQ_Q31Rand (int64_t *pSeed)`;

Float: `float _LIBQ_ToFloatQ31 (_Q31 x)`; `float _LIBQ_ToFloatQ15 (_Q15 x)`; `_Q31 _LIBQ_Q31FromFloat (float x)`; `_Q15 _LIBQ_Q15FromFloat (float x)`;

String: `void _LIBQ_ToStringQ15 (_Q15 x, char *s)`; `_Q15 _LIBQ_Q15FromString (char *s)`;

Functions

	Name	Description
≡	<code>_LIBQ_Q15_cos_Q2_13</code>	Calculates the value of cosine(x).
≡	<code>_LIBQ_Q15_sin_Q2_13</code>	Calculates the value of sine(x).
≡	<code>_LIBQ_Q15FromFloat</code>	Converts a float to a <code>_Q15</code> value.
≡	<code>_LIBQ_Q15FromString</code>	Ascii to <code>_Q15</code> conversion.
≡	<code>_LIBQ_Q15Rand</code>	Generate a <code>_Q15</code> random number.
≡	<code>_LIBQ_Q16_tan_Q2_29</code>	Calculates the value of tan(x).
≡	<code>_LIBQ_Q16Div</code>	<code>_Q16</code> fixed point divide.
≡	<code>_LIBQ_Q16Exp</code>	Calculates the exponential function e^x .

	_LIBQ_Q16Power	Calculates the value of x raised to the y power (x^y).
	_LIBQ_Q16Sqrt	Square root of a positive _Q16 fixed point value.
	_LIBQ_Q2_13_acos_Q15	Calculates the value of $\text{acos}(x)$.
	_LIBQ_Q2_13_asin_Q15	Calculates the asin value of $\text{asin}(x)$.
	_LIBQ_Q2_13_atan_Q7_8	Calculates the value of $\text{atan}(x)$.
	_LIBQ_Q2_13_atan2_Q7_8	Calculates the value of $\text{atan2}(y, x)$.
	_LIBQ_Q2_29_acos_Q31	Calculates the value of $\text{acos}(x)$.
	_LIBQ_Q2_29_acos_Q31_Fast	Calculates the value of $\text{acos}(x)$. This function executes faster than _LIBQ_Q2_29_acos_Q31 but is less precise.
	_LIBQ_Q2_29_asin_Q31	Calculates the value of $\text{asin}(x)$.
	_LIBQ_Q2_29_asin_Q31_Fast	Calculates the value of $\text{asin}(x)$. This function executes faster than the _LIBQ_Q2_29_asin_Q31 function, but is less precise.
	_LIBQ_Q2_29_atan_Q16	Calculates the value of $\text{atan}(x)$.
	_LIBQ_Q2_29_atan2_Q16	Calculates the value of $\text{atan2}(y, x)$.
	_LIBQ_Q3_12_log10_Q16	Calculates the value of $\text{Log}_{10}(x)$.
	_LIBQ_Q31_cos_Q2_29	Calculates the value of $\text{cosine}(x)$.
	_LIBQ_Q31_sin_Q2_29	Calculates the value of $\text{sine}(x)$.
	_LIBQ_Q31FromFloat	Converts a float to a _Q31 value.
	_LIBQ_Q31Rand	Generate a _Q31 random number.
	_LIBQ_Q4_11_ln_Q16	Calculates the natural logarithm $\ln(x)$.
	_LIBQ_Q5_10_log2_Q16	Calculates the value of $\log_2(x)$.
	_LIBQ_Q7_8_tan_Q2_13	Calculates the value of $\tan(x)$.
	_LIBQ_ToFloatQ15	Converts a _Q15 value to a float.
	_LIBQ_ToFloatQ31	Converts a _Q31 value to a float.
	_LIBQ_ToStringQ15	_Q15 to ascii conversion.

Macros

	Name	Description
	_Q15_MAX	Maximum value of _Q15 (~1.0)
	_Q15_MIN	Minimum value of _Q15 (-1.0)
	_Q16_MAX	Maximum value of _Q16 (~32768.0)
	_Q16_MIN	Minimum value of _Q16 (-32768.0)
	_Q2_13_MAX	Maximum value of _Q2_13 (~4.0)
	_Q2_13_MIN	Minimum value of _Q2_13 (-4.0)
	_Q2_29_MAX	Maximum value of _Q2_29 (~4.0)
	_Q2_29_MIN	Minimum value of _Q2_29 (-4.0)
	_Q3_12_MAX	Maximum value of _Q3_12 (~8.0)
	_Q3_12_MIN	Minimum value of _Q3_12 (-8.0)
	_Q31_MAX	Maximum value of _Q31 (~1.0)
	_Q31_MIN	Minimum value of _Q31 (-1.0)
	_Q4_11_MAX	Maximum value of _Q4_11 (~16.0)
	_Q4_11_MIN	Minimum value of _Q4_11 (-16.0)
	_Q5_10_MAX	Maximum value of _Q5_10 (~32.0)
	_Q5_10_MIN	Minimum value of _Q5_10 (-32.0)
	_Q7_8_MAX	Maximum value of _Q7_8 (~128.0)
	_Q7_8_MIN	Minimum value of _Q7_8 (-128.0)

Types

	Name	Description
	<u>_Q0_15</u>	1 sign bit, 15 bits right of radix
	<u>_Q0_31</u>	1 sign bit, 31 bits right of radix
	<u>_Q15</u>	Short name for <u>_Q0_15</u>
	<u>_Q15_16</u>	1 sign bit, 15 bits left of radix, 16 bits right of radix
	<u>_Q16</u>	Short name for <u>_Q15_16</u>
	<u>_Q2_13</u>	1 sign bit, 2 bits left of radix, 13 bits right of radix
	<u>_Q2_29</u>	1 sign bit, 2 bits left of radix, 29 bits right of radix
	<u>_Q3_12</u>	1 sign bit, 3 bits left of radix, 12 bits right of radix
	<u>_Q31</u>	Short name for <u>_Q_0_31</u>
	<u>_Q4_11</u>	1 sign bit, 4 bits left of radix, 11 bits right of radix
	<u>_Q5_10</u>	1 sign bit, 5 bits left of radix, 10 bits right of radix
	<u>_Q7_8</u>	1 sign bit, 7 bits left of radix, 8 bits right of radix

File Name

libq.h

Company

Microchip Technology Inc.

5.5 Operating System Abstraction Layer (OSAL) Library Help

5.5.1 Introduction

OSAL Library

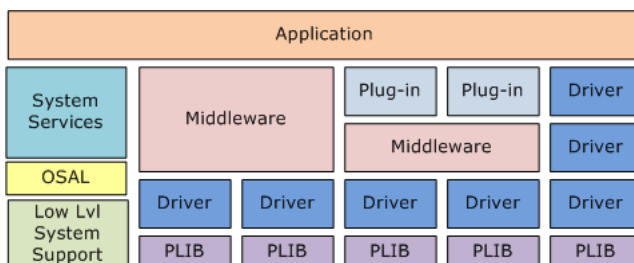
for

Microchip Microcontrollers

The Operating System Abstraction Layer (OSAL) provides a consistent interface to an application and parts of MPLAB Harmony that allow them to take advantage of Operating System constructs when running in an OS environment OR when operating without one. It is designed to allow operation of MPLAB Harmony on any Microchip microcontroller and takes care of the underlying differences between the available OS Kernels or when no kernel is present.

Description

The OSAL provides the interface to commonly available RTOSes such that drivers and middleware (and optionally, applications) may be written using a single interface to a minimal set of OS-specific features needed to provide thread safety. The OSAL interface can be implemented appropriately to support almost any desired RTOS. For applications where no RTOS is available, or desired, a bare version of the OSAL supports either polled or interrupt-driven environments running directly on the hardware. This allows applications and the MPLAB Harmony stack to be executed in all three common embedded environments: polled (shared multi-tasking), interrupt-driven, or RTOS-based.



Scope

The OSAL is not designed to replace a commercial kernel and therefore the user is encouraged to use any of the specific features of their chosen RTOS in order to achieve best performance. As such the OSAL can also be considered to be an Operating System Compatibility Layer offering MPLAB Harmony and users the required common functions to ensure that MPLAB Harmony will correctly compile and operate in both RTOS and non-RTOS environments.

The common interface presented by the OSAL is designed to offer a set of services typically found on micro-kernel and mini-scheduler systems. Because of this it has no aspirations to provide an equivalent set of capabilities as those found on large multi-tasking systems such as uCLinux. The common services are designed to allow MPLAB Harmony to implement thread-safe Drivers and Middleware. The design intention is that drivers will use the minimal set of OSAL features necessary to ensure that they can safely operate in a multi-threaded environment yet can also compile correctly when no underlying RTOS is present. The

range of features used by a driver is typically limited to these OSAL features:

- Semaphore Functions
- Mutexe Functions
- Critical Section Functions

5.5.2 Release Notes

MPLAB Harmony Version: v0.70b **OSAL Library Version :** 0.1 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Three OSAL implementations:

- **OSAL_USE_NONE** - Default implementation of the OSAL that includes no RTOS functionality and features only sufficient code to ensure that elements of MPLAB Harmony that make OSAL calls will compile correctly. 'Safe' parameters and #defines have been used so that MPLAB Harmony drivers will compile correctly.
- **OSAL_USE_BASIC** - Minimal OSAL implementation not designed to operate in multi-threaded environments but offers some usable features so that interrupt driven applications can still compile and operate correctly. Minimal implementations include Events, Semaphores and Mutexes.
- **OSAL_USE_RTOS** - An RTOS is being used and the relevant RTOS header file will be included in the sys_config.h file.

Currently a reference implementation of the FreeRTOS(tm) and Micrium uC/OSIII operating systems have been implemented. Other operating systems implementations are under development.

Known Issues:

Nothing to report in this release.

5.5.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.5.4 Using the Library

This topic describes the basic architecture of the OSAL Library and provides information and examples on how to use it.

Description

Interface Header File: [osal.h](#)

The interface to the OSAL Library is defined in the "[osal.h](#)" header file. Any C language source (.c) file that uses the OSAL System Service library should include "[osal.h](#)".

Library File: [osal.c](#) OR osal_xxxx.c

The OSAL Library consists of a base implementation and individual ports of the OSAL to target operating systems. These target operating systems consist of a basic implementation (used when the macro OSAL_USE_BASIC is defined) and ports of third party RTOSs (used when the macro OSAL_USE_RTOS is defined).

If no RTOS support is required then the file "[osal.c](#)" can be *optionally* included into the project; however, it is *not required*. This file provides implementations for the OSAL_USE_BASIC and configurations of the library.

When an RTOS is being used (the OSAL_USE_RTOS configuration) then an external implementation file which provides the required interface wrappers should be added to the project. For instance for the FreeRTOS operating system the file "osal_freertos.c" should be added, whilst for the Micrium uC/OS-III operating system the file "osal_micrium.c" should be added.

The base implementation and some generic ports are provided with the Library however it is the responsibility of third party vendors to supply the correctly implemented interface file.

Please refer to the [MPLAB Harmony Overview](#) for how the OSAL Library interacts with the framework.

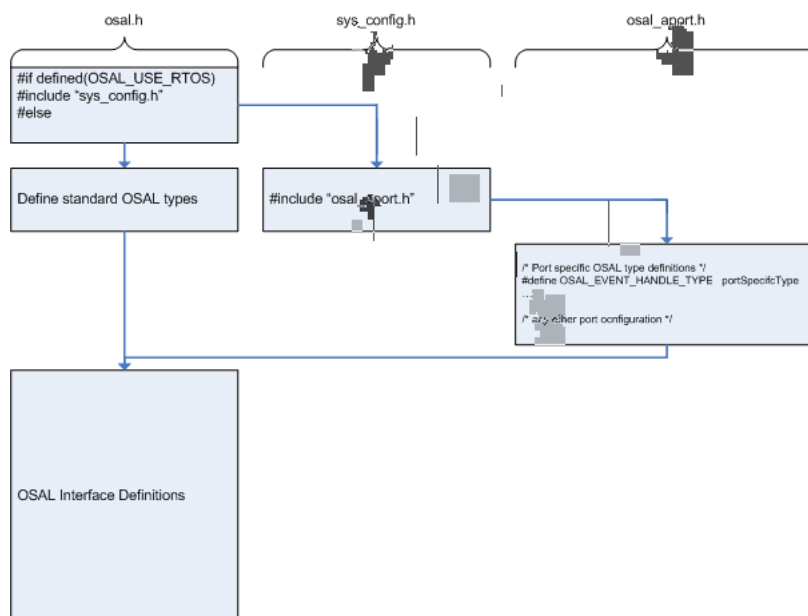
5.5.4.1 Abstraction Model

The OSAL Library provides a predefined set of functions and types that match those services that an RTOS will typically provide. It is designed to be a lightweight model and deliberately excludes the much broader depth and breadth of services that a fully fledged RTOS provides. As such the interface defines only those core functions necessary for the MPLAB Harmony Drivers and middleware to operate in a multi-threaded environment AND some additional functionality to support RTOS independent development of middleware and user applications in the future.

Description

The common interface can easily be ported to many host RTOSs by third parties and the set of functions provides a basic level of RTOS compatibility. Where a specific RTOS does not implement a given architectural feature (e.g., events), the OSAL port for that RTOS should endeavor to imitate that feature using the constructs that are available. Although it is recognized that this may have a detrimental effect on the performance of that system it does allow MPLAB Harmony developers the broadest scope for using RTOS features in their designs.

Include Hierarchy of osal.h



5.5.4.2 Library Overview

This section provides an overview of the OSAL Library.

Description

Refer to the section System Services Overview for how the system services operates in a system.

The OSAL Library provides a defined interface such that driver and middleware developers will be able to create MPLAB Harmony code that can safely operate in a multi-threaded environment when a supported RTOS is present yet will still compile and function correctly when MPLAB Harmony is being used in a non-RTOS environment with an interrupt or non-interrupt driven application model.

The defined interface will help application developers during the early stages of their design when various platforms and architectures are still under consideration. The developer is encouraged to use the specific features of a chosen RTOS once it has been selected since this is likely to provide a more effective and rich programming environment.

The OSAL Library is deliberately designed to be a thin layer over an underlying RTOS which presents a predefined interface to the common features used by the majority of RTOSes, which includes:

Library Interface Section	Description
Semaphore Functions	Binary and counting semaphores for inter thread communications.
Queue Functions	Arbitrary queue mechanism with pass by copy.
Mutex Functions	Thread and resource locking mechanism.
Critical Section Functions	Application and scheduler locking mechanism.
Thread Functions	Basic thread creation and control.
Interrupt Handling Functions	ISR primitives to allow RTOS calls from within an ISR context.
Error Handling Functions	Simple error handling.

Memory Allocation Functions	Memory allocation primitives or wrappers.
OSAL Control Functions	OS initialization and start-up.

One of the primary design guidelines is that a host operating system may not be present and so any operations that the OSAL presents are designed to compile out to safe default values if no RTOS is present. This can mean the following:

- Implementing a dummy function that mimics typical RTOS behavior
- Implementing a #define or inline function that returns a 'safe' generic return value, such as 'TRUE' or 'Call succeeded'
- Returning a OSAL_RESULT_NOT_IMPLEMENTED value to indicate an unsupported operation
- Throwing an ASSERT failure to indicate a terminal error that prevents operation under specific circumstances

5.5.4.3 How the Library Works

This section provides information on how the OSAL Library works.

5.5.4.3.1 Core Functionality

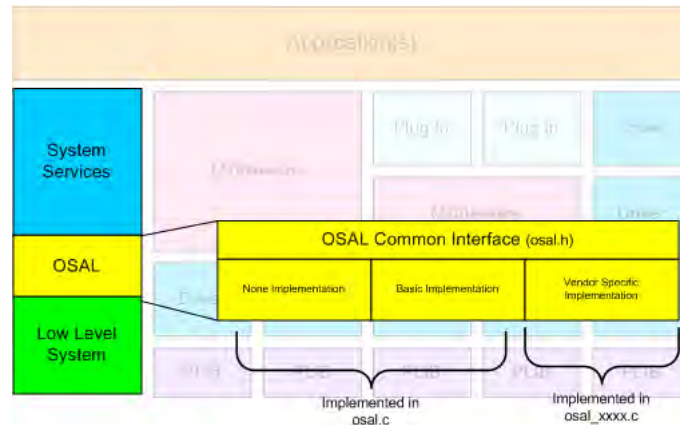
The OSAL Library implements a lightweight abstraction layer on top of a host operating system. The OSAL functionality is designed to allow a programmer access to typical RTOS based functions than can reasonably be expected to exist on a small kernel or micro-scheduler typically found on the target Microchip Microcontrollers.

The OSAL is designed to be compiled under three specific mutually exclusive operating modes:

Compilation Mode	Description
<code>OSAL_USE_NONE</code>	No RTOS or OSAL support is provided. The OSAL interface functions will compile to 'nothing' or are defined to have no value.
<code>OSAL_USE_BASIC</code>	Basic level of support. There is no underlying RTOS however certain facilities are replicated such as Semaphores and Mutexes.
<code>OSAL_USE_RTOS</code>	A full implementation of the OSAL interface is available and there is an underlying RTOS as part of the program.

In the absence of any one of these values being defined or passed into the compiler as a command line parameter then the OSAL will default to the `OSAL_USE_NONE` mode of operation.

The following conceptual diagram shows the common interface and how the individual implementations are derived depending upon the configured mode of compilation.



The OSAL interface makes extensive use of object handles when passing RTOS parameters around the various functions. This has been done since depending on the operating mode and underlying RTOS (if any) there may be substantial differences between the C types of the variables. For instance, in the basic implementation mode a semaphore is implemented as a single 8-bit quantity to ensure atomic access on all architectures. On a third party RTOS the same semaphore may be implemented as a 16-bit index into an array of pointers whilst on another RTOS it could be represented as a `size_t` pointer (a pointer to the basic word type for that architecture). Because of these differences, the most convenient mechanism to maintain a regular interface is by the use of handles to implementation defined variables.

The majority of the OSAL objects also provide a `OSAL_XXX_Delete` function. This is provided for completeness however care should be taken in its use. Many RTOS implementations do not provide the ability to delete an object and excessive use of the delete can lead to memory fragmentation and unpredictable system behavior. It is recommended that it is not used at this time, or if used, the return value should be checked for a possible failure to delete the object.

In the following sections the basic behavior of the supported functions are documented. For the purposes of documentation the assumed operation mode will be that a full RTOS is present and so the behavior is that which will be observed in that scenario.

5.5.4.3.2 Other Functionality

5.5.4.3.3 Thread Operation

The OSAL implements a simple threading model that assumes the use of a priority based pre-emptive scheduler.

Description

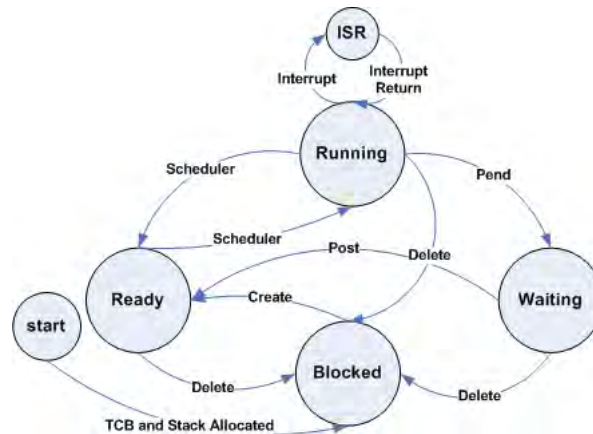
Each thread is implemented as a single C function which can be reused across multiple threads if desired. For this reason care should be taken if static variables are allocated within the thread function.

The priority of a thread is defined using higher values to indicate higher priority. Thread priorities range from 1 up to the application defined value `OSAL_CONFIG_MAX_PRIORITIES` minus 1. No importance is placed upon any specific value of priority other than that a thread with a higher priority will execute before a lower priority one if they are both in the ready state. Certain RTOS implementations may require threads to each have a unique priority; however, since MPLAB Harmony has no knowledge of the various ways in which libraries may be connected together it is not possible to define any fixed priorities. The application programmer is therefore responsible for ensuring the relative priority of threads.

It is assumed that each thread is created with its own stack and that the RTOS is responsible for the correct allocation of memory resources. The requested memory size is specified in bytes and its value is normally arrived at by inspection of the program, the underlying hardware architecture (16-/32-bits) and the total available memory.

No significance is placed upon the textual name given to a task and it is normally only used for diagnostic purposes.

The following diagram shows the sequence of states that a typical RTOS will move a given task through during its lifetime.



The *pdata* parameter passed during thread creation allows the application programmer to generate more sophisticated behavior based upon passing a configuration parameter to a thread. The thread can be created multiple times and perform per instance operations related to the passed parameter.

The *phandle* parameter allows the created threads handle to be passed back to the calling code allowing later manipulation of the thread by other parts of the program.

Example Code:

```

/* Create the application thread for this test harness. This example thread
 * replicates the functionality of the ADC data logger thread
 * The priority of the application task should be selected as a middle value
 * since the check tasks are manipulated above and below this priority level */
OSAL_THREAD_Create(APP_Tasks, "APP", 500, OTH_APP_THREAD_PRIORITY, NULL, NULL);

```

The OSAL supports the creation of daemon threads which are designed to allow multi-threaded drivers with their own task routine to operate with an underlying RTOS. The [OSAL_THREAD_CreateDaemon](#) function takes a single parameter, which is the name of the Daemon thread. The user is responsible for implementing the Daemon thread as appropriate for their system. This thread should then call the individual driver state machines ensuring that they continue to operate. For example, consider the following scenario:

1. The application uses a driver that has no tasks routine although it does use driver tasks called as a result of an interrupt.
2. The application also uses a driver with a state machine that needs calling every 500 us.
3. The application also uses two more drivers that can conveniently be called one after the other (perhaps it is the same driver but with two instantiations) and only needs processing every 100 ms.

For item 1, no Deamon thread is required. For item 2, a Daemon thread is required that calls the driver task routine and then sleeps for 500 us. For the remaining driver, a single Daemon thread can be created that calls the driver twice (with different object handles) and then sleeps for 100 ms.

5.5.4.3.4 Events

The event mechanism is based upon a single boolean value that can be set or reset by a thread. Multiple threads may pend (or block) waiting on the value to be set or flagged.

The OSAL implementation of events is extremely basic and no combinatorial operations are available. Furthermore, whilst multiple threads may block on the event, only the highest priority thread will be released by the event being flagged.

It is strongly suggested that developers consider the use of queues or semaphores for inter process communication or

sequencing since they provide a more deterministic response time than multiple event flags.

5.5.4.3.5 Semaphores

The semaphore implements a method for thread synchronization. This synchronization can be either between one thread and another or between an interrupt service routine (ISR) and a thread. A semaphore once signalled will unblock the highest priority thread currently pending on it.

Description

A semaphore can be used to lock a shared resource, although it is more normal to use a mutex for such an activity. Once obtained a semaphore should be posted back to enable it to be re-taken at a later time or in another thread.

```
/* mainline code prior to OS start */
/* declare a variable of type semaphore handle */
OSAL_SEM_DECLARE(semSync);
/* create the semaphore */
OSAL_SEM_Create(&semSync, OSAL_SEM_TYPE_BINARY, 0, 0);

/* thread one */
...
/* take the semaphore without waiting */
OSAL_SEM_Pend(semSync, 0);
... perform some actions
/* return the semaphore */
OSAL_SEM_Post(semSync);
...

/* thread two must not execute until thread one has finished its operations*/
...
/* block on the semaphore */
OSAL_SEM_Pend(semSync, OSAL_WAIT_FOREVER);
... perform some more actions
/* return the semaphore */
OSAL_SEM_Post(semSync);
```

A semaphore can be signalled multiple times and so provides a method for an ISR to release a thread waiting on it. Even though the blocked thread never returns the semaphore, because the asynchronous ISR repeatedly posts it the next time the thread wants to pend on the semaphore it will be available. By moving the majority of interrupt service processing from the ISR to a high priority thread the system response time is improved and the eventual processing can take advantage of OSAL features such as mutexes and queues which would normally be harder to implement inside the ISR. This technique is known as deferred interrupt processing.

```
/* an example interrupt handler called from an ISR that performs task synchronization using a semaphore */
void _ISRTasksRX(void) /* N.B. pseudo-code ISR */
{
    ...

    _DRV_USART_InterruptSourceStatusClear(_DRV_USART_GET_INT_SRC_RX(_DRV_USART_OBJ(dObj,
rxInterruptSource)));

    /* Release the receive semaphore unblocking any tasks */
    OSAL_SEM_PostISR(_DRV_USART_OBJ(dObj, rxSemID));
} /* DRV_USART_TasksRX */
```

5.5.4.3.6 Mutex Operation

A mutex or mutual exclusion is used to protect a shared resource from access by multiple threads at the same time. A shared resource may be a common data structure in RAM or it may be a hardware peripheral. In either case a mutex can be used to

ensure the integrity of the entire resource by only allowing one thread to access it at a time.

Description

The program must be written in such a way that before the shared resources is accessed the mutex has to be obtained. Once obtained the accesses should occur, and once complete the mutex should then be released. While no restrictions are enforced the sequence of operations between the lock and unlock should ideally take as few lines of code as possible to ensure good system performance.

The mutex may be implemented as a form of binary semaphore but an underlying RTOS will often add other features. It is normal to add the restriction that a mutex may only be unlocked from the thread that originally obtained the lock in the first place. The RTOS may also provide features to mitigate priority inversion problems (where a high priority thread blocks on a lower priority one holding a mutex) by providing priority inheritance allowing lower priority threads to be temporarily raised to complete and release a locked mutex.

```
/* perform operations on a shared data structure */
struct myDataStructure {
    uint16_t x;
    uint8_t y;
} myDataStructure;

...
OSAL_MUTEX_DECLARE(mutexDS);
OSAL_MUTEX_Create(&mutexDS);

...
/* wait 2 seconds to obtain the mutex */
if (OSAL_MUTEX_Lock(mutexDS, 2000) == OSAL_RESULT_TRUE)
{
    /* operate on the data structure */
    myDataStructure.x = 32;
    OSAL_MUTEX_Unlock(mutexDS);
}
```

5.5.4.3.7 Queue Operation

This section describes the implementation of queues using the OSAL Library.

Description

The OSAL implements queues as a fixed structure of elements, with each element the same size. The size and number of each element is defined when the queue is created but cannot be reconfigured later. Elements are added to the queue when using the Add function call and removed on a first-in-first-out basis when using the Remove function. If the queue is full at the time of attempting to add data, the call will block for the specified time before failing. Data placed on the queue is done using a copy operation so that the source may be over written if required. For large data items it may be better to pass pointers to the data via the queue however care must be taken on ownership once the pointer has been copied between threads.

It is possible to place an item at the front of a queue using the AddHead function. Since the implementation of this is heavily dependent upon the underlying RTOS the call may return with OSAL_RETURN_NOT_IMPLEMENTED if this operation is not possible.

```
/* Create a queue */
OSAL_QUEUE_Declare(myQueue);

...
/* create a queue that accepts 1 byte chars and is 4 items long */
OSAL_QUEUE_Create(&myQueue, 1, 4);

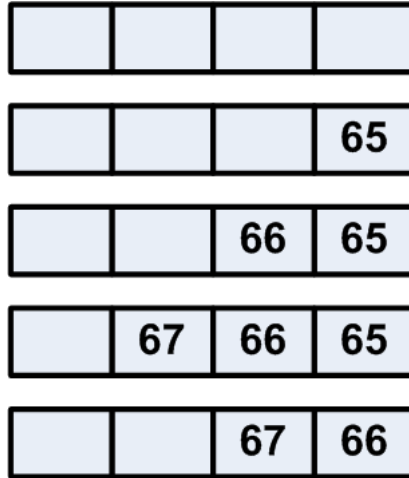
...
/* Add data */
char c = 'A';
OSAL_QUEUE_Add(myQueue, &c, 0);
c = 'B';
OSAL_QUEUE_Add(myQueue, &c, 0);
```

```

c = 'C';
OSAL_QUEUE_Add(myQueue, &c, 0);

/* remove data */
char d;
OSAL_QUEUE_Remove(myQueue, &d, OSAL_WAIT_FOREVER);

```



5.5.4.3.8 Critical Section Operation

This section describes how critical sections are used.

Description

Critical sections are used to form sequences of code that must operate in an atomic manner. The interface allows for the possibility of two types of critical section.

- When the critical section is entered all interrupts on the microcontroller are disabled. This prevents the protected sequence of code from being interrupted and ensures the complete atomicity of the operation. This is denoted by the OSAL_CRIT_TYPE_HIGH value.
- When the critical section is entered the RTOS scheduler is disabled. In this second case other threads are prevented from running however interrupts can still occur which allows any asynchronous events to still be received and for the temporal accuracy of the RTOS scheduler to be maintained. This is denoted by the OSAL_CRIT_TYPE_LOW value.

Since the behavior in the two cases is different the type of critical section must be identified in both the call to enter and leave.

```

/* enter and leave a critical section disabling interrupts */
OSAL_CRIT_Enter(OSAL_CRIT_TYPE_HIGH);
/* perform an atomic sequence of code */
...
/* leave the critical section */
OSAL_CRIT_Leave(OSAL_CRIT_TYPE_HIGH);

```

The underlying RTOS may not support the second scenario, in which case the OSAL implementation will default to disabling all interrupts.

5.5.4.3.9 Interrupt Operation

This section describes interrupt handling by the OSAL Library.

Description

Interrupt handling in the OSAL is implemented using a separate set of interrupt-only functions. Within MPLAB Harmony, interrupts are deemed to be handled at an application level, meaning the initial interrupt handler or vector is dealt with while

writing the application rather than defaulting to MPLAB Harmony provided handlers. However, within MPLAB Harmony there may be functions that are called within the context of an ISR and also during mainline program flow depending on whether MPLAB Harmony has been built with interrupt or polled drivers. Because of this, to allow the most flexible and adaptable OSAL implementation, the following call sequence should be performed.

1. At the application interrupt handler level, immediately after the interrupt has been entered and before any further calls are made, call `OSAL_ISR_Enter()`.
2. Once inside the ISR OR inside any descendent functions called as a result of entering the ISR ensure that only `OSAL_XXX_xxxxISR` variants of functions are called.
3. At the exit point(s) of the ISR call `OSAL_ISR_Exit()`.

By following this sequence of operations, the OSAL can be made to adapt to the requirements of different underlying RTOS architectures.

In the main ISR handler:

```
void __ISR ( _TIMER_1_VECTOR ) _InterruptHandler_TMR_1_stub ( void )
{
    OSAL_ISR_Enter();

    /* Call the timer driver's "Tasks" routine */
    DRV_TMR_Tasks ( appDrvObject.tmrObject );

    OSAL_ISR_Exit();
}
```

In the called function:

```
void _DRV_TMR_MAKE_NAME( Tasks )
(
    _DRV_TMR_DYN_ARG( SYS_MODULE_OBJ object )
)
{
    ...

    /* Clear Timer Interrupt/Status Flag */
    _DRV_TMR_InterruptSourceClear( _DRV_TMR_GET_INT_SRC( _DRV_TMR_OBJ(dObj, interruptSource) )
);

    /* release a semaphore, note use of ISR variant */
    OSAL_SEM_PostISR(&semTMR);
} /* DRV_TMR_Tasks */
```

It is not appropriate (or logical) to block inside an ISR so all variants of the OSAL functions with ISR at the end do not take a block time parameter. The use of ISR specific versions of OSAL functions is dependent on the RTOS; however, if the xxxISR variants are used by default inside any ISRs, the actual implementation of the interface on a given RTOS can be allowed to handle the complexities on that specific platform.

5.5.4.3.10 Error Operation

This section describes error handling within an OSAL function.

Description

If an error occurs within an OSAL function or inside the underlying RTOS, the `OSAL_ErrorCallback` function will be called. This callback function is declared as a `_weak_` function and the programmer can override it to provide application specific error handling. Currently, only a few error codes are listed and it is normally difficult to provide any kind of constructive recovery from an internal error. The default course of action is to either block and indicate the condition using some visual means or to perform a processor reset.

```
/* user implementation of the error callback */
void OSAL_ErrorCallback(OSAL_ERROR_CALLBACK_TYPE type)
{
    while (1)
    {
        // a non recoverable error occurred so loop here
        BlinkLED();
    }
}
```

5.5.4.3.11 Memory Operation

The OSAL Library provides an interface to a memory allocation mechanism. The memory required for dynamic instantiation of variables is normally provided by allocating it from the heap. However the standard C library implementation of malloc and free are not considered thread safe and so OSAL specific functions must be used if MPLAB Harmony or the application requires dynamic memory during operation.

When operating without an underlying RTOS the OSAL memory allocators default to using standard malloc and free functions. However, when operating with an RTOS the calls will defer to the specific scheme used by the RTOS. This may involve multiple memory pools or it may simply involve adding a critical section around calls to malloc and free. It is left to the implementation to define the most appropriate scheme.

```
/* allocate a large buffer */
uint8_t* buffer;

buffer = OSAL_Malloc(8000);
if (buffer != NULL)
{
    ... manipulate the buffer
    /* free the buffer */
    OSAL_Free(buffer);
    buffer = NULL;
}
```

5.5.4.3.12 OSAL Operation

This section describes OSAL control features.

Description

When the OSAL is using an underlying RTOS it may be necessary to allow the RTOS to perform one-time initialization before any calls to it are made. For instance, the RTOS might implement multiple memory pools for managing threads, queues and semaphores and it must be given the chance to create these pools before any of the objects are created. For this reason the application program should call [OSAL_Initialize\(\)](#) early on and certainly before any MPLAB Harmony Drivers or middleware is initialized (since these may also create OSAL objects at creation time).

Once the OSAL is initialized and any other remaining parts of the system are configured correctly, the OSAL can be entered with [OSAL_Start\(\)](#). There is no expectation that control will be returned from this function and any return is likely to indicate a fault condition.

5.5.5 Configuring the Library

The configuration of the OSAL Library is based on the file sys_config.h

This header file contains the configuration selection for the OSAL Library. Based on the selections made, the OSAL will support or not support selected features.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

Description

To configure the OSAL the compilation mode needs to be selected. Three options are currently supported and these define the implementation details for the OSAL. In all cases the same interface is present; however, the functionality at the varying levels will be different or not supported. Only one of these #define statements should be specified, and this can be either done in a common location (such as sysconfig.h) or on the build tool command line.

Compilation Mode	Description
<code>OSAL_USE_NONE</code>	No RTOS or OSAL support is provided. The OSAL interface functions will compile to 'nothing' or are defined to have no value.
<code>OSAL_USE_BASIC</code>	Basic level of support. There is no underlying RTOS; however, certain facilities are replicated such as Semaphores and Mutexes.
<code>OSAL_USE_RTOS</code>	A full implementation of the OSAL interface is available and there is an underlying RTOS as part of the program.

5.5.5.1 Initialization Overrides

This section provides information for overriding OSAL initialization.

Description

The OSAL requires minimal configuration. In the case of the `OSAL_USE_NONE` and `OSAL_USE_BASIC` all of the configuration required is provided by the standard "`osal.h`" include file. When a particular RTOS is being used then additional OSAL configuration parameters should be placed in "`sys_config.h`".

The primary requirement is to include the relevant interface implementation for the selected RTOS being used. This file will contain the mappings from OSAL interface types to types relevant for the chosen RTOS.

```
/* included the name of the port specific interface file */
#include "osal_<port name>.h"
```

Configuration parameter to define the maximum number of unique priority levels that a thread may take.

Configuration Parameter	Purpose
<code>OSAL_MAX_PRIORITIES</code>	Maximum number of thread priority levels allowed in the application. The assignable priorities for a thread are assumed to run from 1 to <code>OSAL_MAX_PRIORITIES</code> - 1 . Some underlying RTOSs may permit multiple threads at the same priority level (implementing a round-robin scheduling) so the total number of threads may actually be higher than the defined value. This is an application design decision.

If the application is to support multi-threaded drivers with Daemon threads then the stack size and priority of the these should be defined using the following macros:

Configuration Parameter	Purpose
<code>OSAL_DAEMON_STACK_SIZE</code>	Number of bytes to allocate to a daemon thread. These threads are likely to be small and require small stacks although the exact size depends on the architecture. Any OSAL port should implement safe defaults. Typical values are 800 bytes for a PIC32 or 300 for a PIC24.

OSAL_DAEMON_PRIORITY	Default priority of a daemon thread. Since these threads are typically calling MPLAB Harmony drivers and middleware their priority should be higher than the majority of the application in order to keep them from suffering driver starvation. The most appropriate value is left to the programmer to decide.

5.5.5.2 Others

[Drag and Drop other configuration options](#)

5.5.5.3 Examples - Sample Functionality

Example sys_config.h fragment:

```
// *****
// *****
// Section: Application Configuration
// *****
// *****
/* These definitions select the configuration options for the application
*/
/* configure the priorities and stack sizes for the daemon tasks */
#define OSAL_DAEMON_STACK_SIZE      800
#define OSAL_DAEMON_PRIORITY        4

/* configure the maximum number of priorities in this application */
#define OSAL_CONFIG_MAX_PRIORITIES  31

/* declare the daemon tasks used by the OSAL to implement driver tasks */
extern void osalDaemon1(void* pv);
extern void osalDaemon2(void* pv);

/* Include the port specific OS implementation files. The file is required to
 * create an implementation layer for the selected OS. This include will be
 * changed depending on the required OS being used */
#include "osal_freertos.h"
```

5.5.6 Building the Library

This section list the files that are available in the \src of the OSAL Library. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

Files

Name	Description
osal.c	Bare metal implementation of OSAL functions

Description

As additional implementations of the OSAL Library are added to MPLAB Harmony, their interface files may be added to the \src directory. Since many RTOS vendors offer their products as a commercial and licensed offering it is likely that considerably more source files will be required than just the "osal_xxxx.c" file. These RTOS source files will be placed inside directories specific to the manufacturer and instructions on how to add them to a MPLAB Harmony project will be provided by the supplier.

Summary

if you are building using the default OSAL Library and have selected the [OSAL_USE_NONE](#) or `OSAL_USE_BASIC` compilation options then only "[osal.c](#)" needs to be added to the project. However, if a specific RTOS is being targeted and `OSAL_USE_RTOS` is selected, the port specific implementation file will need to be added to the project along with the files for that vendor.

5.5.6.1 osal.c

Basic OSAL compatibility layer.

This file contains functional implementations of the bare metal OSAL. Where it is logical and possible to implement non threading versions of the common function calls they have been implemented here directly or in the [osal.h](#) file as macros. If it is not possible, or logical, to implement a feature WITHOUT an RTOS present, the the matching definition in [osal.h](#) will be set to a NULL (or empty) implementation.

File Name

osal.c

5.5.7 Library Interface

Data Types and Constants

	Name	Description
	OSAL_MUTEX_HANDLE_TYPE	Handle to a mutex, this is the default implementation and may be overridden in an actual implementation file
	OSAL_QUEUE_HANDLE_TYPE	Handle to a queue, this is the default implementation and may be overridden in an actual implementation file
	OSAL_SEM_HANDLE_TYPE	Handle to a semaphore, this is the default implementation and may be overridden in an actual implementation file
	OSAL_THREAD_HANDLE_TYPE	Handle to a thread, this is the default implementation and may be overridden in an actual implementation file
	OSAL_USE_NONE	This is macro <code>OSAL_USE_NONE</code> .
	OSAL_WAIT_FOREVER	Default value indicating block forever
	OSAL_RESULT	Enumerated type representing the general return value from OSAL functions.
	OSAL_SEM_TYPE	Enumerated type representing the possible types of semaphore.

Critical Section Functions

	Name	Description
	OSAL_CRIT_Enter	Enters a critical section with the specified severity level.
	OSAL_CRIT_Leave	Leaves a critical section with the specified severity level.
	OSAL_CRIT_TYPE	Enumerated type representing the possible types of critical section.

Error Handling Functions

	Name	Description
	OSAL_ERROR_CALLBACK_TYPE	Enumerated type representing the possible types of callback message.
	OSAL_ErrorCallback	Lightweight error handling callback function for the RTOS.

Interrupt Handling Functions

	Name	Description
≡	OSAL_ISR_Enter	Informs the OSAL and RTOS that an ISR is being entered.
≡	OSAL_ISR_Exit	Informs the OSAL and RTOS that an ISR is being exited.

Memory Allocation Functions

	Name	Description
≡	OSAL_Free	Deallocates a block of memory and return to the default pool.
≡	OSAL_Malloc	Allocates memory using the OSAL default allocator.

Mutex Functions

	Name	Description
≡	OSAL_MUTEX_Create	Creates a mutex.
≡	OSAL_MUTEX_Delete	Deletes a mutex.
≡	OSAL_MUTEX_Lock	Locks a mutex.
≡	OSAL_MUTEX_Unlock	Unlocks a mutex.
≡	OSAL_MUTEX_DECLARE	Declares an OSAL mutex.

OSAL Control Functions

	Name	Description
≡	OSAL_Initialize	Performs OSAL initialization.
≡	OSAL_Name	Obtains the name of the underlying RTOS.
≡	OSAL_Start	Starts the underlying RTOS

Queue Functions

	Name	Description
≡	OSAL_QUEUE_DECLARE	Declares an OSAL queue.
≡	OSAL_QUEUE_Add	Add the specified item to the queue with a time-out if the queue is full.
≡	OSAL_QUEUE_AddHead	Adds the specified item to the front of the queue with a timeout if the queue is full.
≡	OSAL_QUEUE_AddHeadISR	Adds the specified item to the front of the queue from within an ISR.
≡	OSAL_QUEUE_AddISR	Adds the specified item to the queue from with an ISR.
≡	OSAL_QUEUE_Create	Creates a queue with the specified width and depth.
≡	OSAL_QUEUE_Delete	Deletes an OSAL_QUEUE.
≡	OSAL_QUEUE_Peek	Copys the item at the head of the queue but leave it on the queue. If the queue is empty, it will block up to waitMS.
≡	OSAL_QUEUE_PeekISR	Copys the item at the head of the queue but leaves it on the queue.
≡	OSAL_QUEUE_Remove	Removes the item at the head of the queue. If the queue is empty it will block up to waitMS.
≡	OSAL_QUEUE_RemoveISR	Removes the item at the head of the queue.

Semaphore Functions

	Name	Description
≡	OSAL_SEM_Create	Creates an OSAL Semaphore.
≡	OSAL_SEM_Delete	Deletes an OSAL Semaphore.
≡	OSAL_SEM_GetCount	Returns the current value of a counting semaphore.
≡	OSAL_SEM_Pend	Waits on a semaphore. Returns true if the semaphore was obtained within the time limit.

	OSAL_SEM_Post	Posts a semaphore or increments a counting semaphore.
	OSAL_SEM_PostISR	Posts a semaphore or increments a counting semaphore from within an Interrupt Service Routine (ISR).
	OSAL_SEM_DECLARE	Declares an OSAL semaphore.

Thread Functions

	Name	Description
	OSAL_THREAD_FUNCTION	Functions that will operate as a separate thread under the OSAL should match this type.
	OSAL_THREAD_Create	Creates an OSAL thread with the specified parameters.
	OSAL_THREAD_CreateDaemon	Creates a Daemon thread for use with multi-threaded Harmony drivers.
	OSAL_THREAD_PriorityGet	Gets the priority of a thread.
	OSAL_THREAD_PrioritySet	Sets the priority of a thread.
	OSAL_THREAD_Resume	Resumes a specified thread.
	OSAL_THREAD_ResumeAll	Resumes all suspended threads.
	OSAL_THREAD_Sleep	Puts the currently executing thread to sleep.
	OSAL_THREAD_Suspend	Suspends a specified thread.
	OSAL_THREAD_SuspendAll	Suspends all threads apart from the calling thread.

Description

This section describes the APIs of the OSAL Library. Refer to each section for a description.

5.5.7.1 Thread Functions

5.5.7.1.1 OSAL_THREAD_FUNCTION Type

C

```
typedef void (* OSAL_THREAD_FUNCTION)(void* pData);
```

Description

OSAL thread function template

OSAL thread functions must match this type or use it explicitly. Thread functions are expected to return nothing. The pData parameter is information passed at creation time and is implementation specific.

Remarks

None.

5.5.7.1.2 OSAL_THREAD_Create Function

C

```
OSAL_RESULT OSAL_THREAD_Create(
    OSAL_THREAD_FUNCTION task,
    const char* name,
    uint16_t stack_size,
    uint8_t priority,
    void* pData,
    OSAL_THREAD_HANDLE_TYPE* pHandle
);
```

Description

This function creates an OSAL thread. The thread function is implemented as a simple task that will normally not return (i.e., a while(1) loop). The name is not used by the OSAL, but may be used by the RTOS for diagnostic and reporting purposes. The thread is created with a fixed stack size that is allocated from one of the available memory pools or more normally from the heap. The priority of the thread is an 8-bit quantity with higher values denoting threads of a higher priority. Optionally, a parameter may be passed to the thread function that can be used to implement per instance behavior. For example, one LED flashing task could be created several times with the LED number passed as a parameter at creation time. A handle to the created thread is returned in pHandle. Threads may be created prior to the RTOS starting, however, no execution will occur until it is started. A created thread is immediately placed into the READY state and is able to run provided that it is the highest priority thread when all threads are evaluated by the kernel, either during normal operation, or at RTOS start-up time.

Preconditions

Thread function must exist in program memory.

Parameters

Parameters	Description
task	Name of the function that implements the thread code
name	Textual name of the thread. Not used by the OSAL but may be used by the RTOS for diagnostic and tracing purposes.
stack_size	Size of the stack associated with a thread in bytes. The RTOS will allocate this many bytes from any underlying memory pool or the heap.
priority	Priority of the thread from 1-255. Higher values indicate threads with higher priorities. It is left down to the underlying RTOS whether threads of the same priority are permitted or if threads should be given a unique priority.
pData	Pointer to data passed to the thread. Optionally, NULL may be passed.
pHandle	Pointer to a variable that can contain the created threads handle

Returns

OSAL_RESULT_TRUE - Thread successfully created

OSAL_RESULT_FALSE - Thread not created or insufficient memory available

Remarks

None.

Example

```
// create a thread passing a data structure as a parameter during creation
OSAL_THREAD_Create(threadCheck, "CHECK", 500, OTH_APP_THREAD_PRIORITY + 2, (void*) &appData,
NULL);
```

5.5.7.1.3 OSAL_THREAD_CreateDaemon Function**C**

```
OSAL_RESULT OSAL_THREAD_CreateDaemon(
    OSAL_THREAD_FUNCTION task
);
```

Description

This function will create a daemon or system thread designed to assist Harmony driver modules with their operation. Some Harmony drivers will incorporate state machines or other loop structures that must be periodically called for the driver to continue operating. For instance, a USART driver may have a state machine that manages the prioritization and transmission of data from multiple clients. This could work in addition to any processing that occurs when operating under interrupts so it will have a separate DRV_xxxx_Tasks routine (or similar) that must be periodically called. Since the prioritization and frequency of these

calls is only known to the programmer when designing the end application, the OSAL provides a lightweight thread mechanism that is primarily intended to be used internally by MPLAB Harmony. The Daemon threads have a fixed size stack (defined in `system_config.h`) and operate at a fixed priority.

Preconditions

Daemon function must exist in program memory.

Parameters

Parameters	Description
task	The name of the function that implements the Daemon

Returns

OSAL_RESULT_TRUE - Thread successfully created

OSAL_RESULT_FALSE - Thread not created or insufficient memory available

Remarks

None.

Example

```
// inside driver initialization routine
...
OSAL_THREAD_CreateDaemon(DRV_USART_Tasks);
```

5.5.7.1.4 OSAL_THREAD_PriorityGet Function

C

```
uint8_t OSAL_THREAD_PriorityGet(
    OSAL_THREAD_HANDLE_TYPE handle
);
```

Description

This function gets the priority of a thread. The priority of the thread with the specified handle is returned. A handle value of NULL returns the priority of the thread making the call.

Preconditions

Thread must have been created.

Parameters

Parameters	Description
handle	The thread handle. NULL indicates use the current thread.

Returns

0-255 - The current priority of the thread. Higher values indicate higher priority.

Remarks

None.

Example

```
// raise the priority of this thread to ensure it completes a complex
// operation
uint8_t currPriority;

currPriority = OSAL_THREAD_PriorityGet(NULL);
OSAL_THREAD_PrioritySet(NULL, currPriority + 1);
```

```
... perform a time critical operation

// restore the priority
OSAL_THREAD_PrioritySet(NULL, currPriority);
```

5.5.7.1.5 OSAL_THREAD_PrioritySet Function

C

```
OSAL_RESULT OSAL_THREAD_PrioritySet(
    OSAL_THREAD_HANDLE_TYPE handle,
    uint8_t priority
);
```

Description

This function sets the priority of a thread. The priority of the thread with the specified handle is changed to the new value. If as a result of the change, it now has the highest priority and if not blocked, will start executing.

Preconditions

Thread must have been created.

Parameters

Parameters	Description
handle	The thread handle. NULL indicates change the priority of the thread in which the call was made.
priority	Priority to be assigned to the thread

Returns

OSAL_RESULT_TRUE - Priority changed

OSAL_RESULT_FALSE - Priority not changed

Remarks

None.

Example

```
// raise the priority of this thread to ensure it completes a complex
// operation
uint8_t currPriority;

currPriority = OSAL_THREAD_GetPriority(NULL);
OSAL_THREAD_PrioritySet(NULL, currPriority + 1);

... perform a time critical operation

// restore the priority
OSAL_THREAD_PrioritySet(NULL, currPriority);
```

5.5.7.1.6 OSAL_THREAD_Resume Function

C

```
OSAL_RESULT OSAL_THREAD_Resume(
    OSAL_THREAD_HANDLE_TYPE handle
);
```

Description

This function resumes a specific thread.

Preconditions

Thread must have been created.

Parameters

Parameters	Description
handle	The thread handle

Returns

OSAL_RESULT_TRUE - Thread resumed

OSAL_RESULT_FALSE - Thread could not be resumed

Remarks

None.

Example

```
// suspend a thread until a later time
OSAL_THREAD_Suspend(thread1Handle);

...

// resume the suspended thread
OSAL_THREAD_RESUME(thread1Handle);
```

5.5.7.1.7 OSAL_THREAD_ResumeAll Function

C

```
OSAL_RESULT OSAL_THREAD_ResumeAll();
```

Description

This function resumes all suspended threads. Execution will continue with the highest priority thread that is ready to execute.

Preconditions

Thread must have been created.

Returns

OSAL_RESULT_TRUE - Successful call

Remarks

None.

Example

```
// resume all threads
OSAL_THREAD_ResumeAll();
```

5.5.7.1.8 OSAL_THREAD_Sleep Function

C

```
OSAL_RESULT OSAL_THREAD_Sleep(
    uint16_t waitMS
);
```

Description

This function puts the currently executing thread to sleep. Execution of the current thread is placed into a blocked state for the specified amount of time. The exact amount of time that the thread will sleep will depend upon the granularity of the underlying

pre-emptive tick.

Preconditions

Thread must have been created.

Parameters

Parameters	Description
waitMS	Number of milliseconds the thread will be blocked

Returns

OSAL_RESULT_TRUE - Call succeeded

Remarks

None.

Example

```
while (1)
{
    // sleep for 500ms
    OSAL_THREAD_Sleep(500);
    // toggle an LED
    DRV_IO_Toggle(drvHandle, LED_ID);
}
```

5.5.7.1.9 OSAL_THREAD_Suspend Function

C

```
OSAL_RESULT OSAL_THREAD_Suspend(
    OSAL_THREAD_HANDLE_TYPE handle
);
```

Description

This function suspends a specific thread. If NULL is passed, the RTOS will attempt to suspend the thread in which the call was made. The thread will then only resume when unsuspended by another part of the program with an appropriate resume call.

Preconditions

Thread must have been created.

Parameters

Parameters	Description
handle	The thread handle. NULL may be passed indicating suspend the calling thread.

Returns

OSAL_RESULT_TRUE - Thread suspended

OSAL_RESULT_FALSE - Thread could not be suspended

Remarks

None.

Example

```
// suspend this thread
OSAL_THREAD_Suspend(NULL);
```

5.5.7.1.10 OSAL_THREAD_SuspendAll Function

C

```
OSAL_RESULT OSAL_THREAD_SuspendAll();
```

Description

This function suspends all threads. Interrupts are still enabled. The calling thread will continue executing at the next instruction.

Preconditions

Thread must have been created.

Returns

OSAL_RESULT_TRUE - Threads suspended

OSAL_RESULT_FALSE - Threads could not be suspended

Remarks

None.

Example

```
// suspend all threads
OSAL_THREAD_SuspendAll();

// perform critical code sequence
// NB a critical section may be a better choice here
...
```

5.5.7.2 Semaphore Functions

5.5.7.2.1 OSAL_SEM_Create Function

C

```
OSAL_RESULT OSAL_SEM_Create(
    OSAL_SEM_HANDLE_TYPE* semID,
    OSAL_SEM_TYPE type,
    uint8_t maxCount,
    uint8_t initialCount
);
```

Description

This function creates an OSAL binary or counting semaphore. If OSAL_SEM_TYPE_BINARY is specified, the maxcount and initialCount values are ignored; otherwise, these must contain valid values.

Preconditions

Semaphore must have been declared.

Parameters

Parameters	Description
semID	Pointer to the Semaphore ID
type	OSAL_SEM_TYPE_BINARY, create a binary semaphore OSAL_SEM_TYPE_COUNTING, create a counting semaphore with the specified count values
maxCount	Maximum value for a counting semaphore (ignored for a BINARY semaphore)

initialCount	Starting count value for the semaphore (ignored for a BINARY semaphore)

Returns

OSAL_RESULT_TRUE - Semaphore created

OSAL_RESULT_FALSE - Semaphore creation failed

semID - Updated with valid semaphore handle if call was successful

Remarks

None.

Example

```
OSAL_SEM_Create(&mySemID, OSAL_SEM_TYPE_COUNTING, 10, 5);
```

5.5.7.2.2 OSAL_SEM_Delete Function

C

```
OSAL_RESULT OSAL_SEM_Delete(  
    OSAL_SEM_HANDLE_TYPE* semID  
);
```

Description

This function deletes an OSAL semaphore, freeing up any allocated storage associated with it.

Preconditions

Semaphore must have been created.

Parameters

Parameters	Description
semID	Pointer to the semID

Returns

OSAL_RESULT_TRUE - Semaphore deleted

OSAL_RESULT_FALSE - Semaphore deletion failed

Remarks

None.

Example

```
OSAL_SEM_Delete(&mySemID);
```

5.5.7.2.3 OSAL_SEM_GetCount Function

C

```
uint8_t OSAL_SEM_GetCount(  
    OSAL_SEM_HANDLE_TYPE semID  
);
```

Description

This function returns the current value of a counting semaphore. The value returned is assumed to be a single value ranging from 0-255.

Preconditions

Semaphore must have been created.

Parameters

Parameters	Description
semID	The semID

Returns

0 - Semaphore is unavailable

1-255 - Current value of the counting semaphore

Remarks

None.

Example

```
uint8_t semCount;

semCount = OSAL_SEM_GetCount(semUART);

if (semCount > 0)
{
    // obtain the semaphore
    if (OSAL_SEM_Pend(semUART) == OSAL_RESULT_TRUE)
    {
        // perform processing on the comm channel
        ...
    }
}
else
{
    // no comm channels available
    ...
}
```

5.5.7.2.4 OSAL_SEM_Pend Function

C

```
OSAL_RESULT OSAL_SEM_Pend(
    OSAL_SEM_HANDLE_TYPE semID,
    uint16_t waitMS
);
```

Description

This function is a blocking function call that waits (i.e., pends) on a semaphore. The function will return true if the semaphore has been obtained, or false if it was not available or the time limit was exceeded.

Preconditions

Semaphore must have been created.

Parameters

Parameters	Description
semID	The semID
waitMS	Time limit to wait in milliseconds. 0, do not wait OSAL_WAIT_FOREVER , return only when semaphore is obtained Other values, timeout delay

Returns

- OSAL_RESULT_TRUE - Semaphore obtained
- OSAL_RESULT_FALSE - Semaphore not obtained or timeout occurred

Remarks

None.

Example

```
if (OSAL_SEM_Pend(semUARTRX, 50) == OSAL_RESULT_TRUE)
{
    // character available
    c = DRV_USART_ReadByte(drvID);
    ...
}
else
{
    // character not available, resend prompt
    ...
}
```

5.5.7.2.5 OSAL_SEM_Post Function

C

```
OSAL_RESULT OSAL_SEM_Post(
    OSAL_SEM_HANDLE_TYPE semID
);
```

Description

This function posts a binary semaphore or increments a counting semaphore. The highest priority task currently blocked on the semaphore will be released and made ready to run.

Preconditions

Semaphore must have been created.

Parameters

Parameters	Description
semID	The semID

Returns

- OSAL_RESULT_TRUE - Semaphore posted
- OSAL_RESULT_FALSE - Semaphore not posted

Remarks

None.

Example

```
OSAL_SEM_Post(semSignal);
```

5.5.7.2.6 OSAL_SEM_PostISR Function

C

```
OSAL_RESULT OSAL_SEM_PostISR(
    OSAL_SEM_HANDLE_TYPE semID
);
```


Description

This function posts a binary semaphore or increments a counting semaphore. The highest priority task currently blocked on the semaphore will be released and made ready to run. This form of the post function should be used inside an ISR.

Preconditions

Semaphore must have been created.

Parameters

Parameters	Description
semID	The semID

Returns

OSAL_RESULT_TRUE - Semaphore posted

OSAL_RESULT_FALSE - Semaphore not posted

Remarks

This version of the [OSAL_SEM_Post](#) function should be used if the program is, or may be, operating inside an ISR. The OSAL will take the necessary steps to ensure correct operation possibly disabling interrupts or entering a critical section. The exact requirements will depend upon the particular RTOS being used.

Example

```
void __ISR(UART_2_VECTOR) _UART2RXHandler()  
{  
    char c;  
  
    // indicate to the OSAL that this is an interrupt function  
    OSAL_ISR_Enter();  
  
    // read the character  
    c = U2RXREG;  
    // clear the interrupt flag  
    IFS1bits.U2IF = 0;  
    // post a semaphore indicating a character has been received  
    OSAL_SEM_PostISR(semSignal);  
  
    // indicate to the OSAL the ISR is ending  
    OSAL_ISR_Exit();  
}
```

5.5.7.2.7 OSAL_SEM_DECLARE Macro

C

```
#define OSAL_SEM_DECLARE(semID) OSAL_SEM_HANDLE_TYPE semID
```

Description

OSAL_SEM_Declare(semID)

This function declares a data item of type [OSAL_SEM_HANDLE_TYPE](#).

Remarks

None.

5.5.7.3 Mutex Functions

5.5.7.3.1 OSAL_MUTEX_Create Function

C

```
OSAL_RESULT OSAL_MUTEX_Create(  
    OSAL_MUTEX_HANDLE_TYPE* mutexID  
);
```

Description

This function creates a mutex, allocating storage if required and placing the mutex handle into the passed parameter.

Preconditions

Mutex must have been declared.

Parameters

Parameters	Description
mutexID	Pointer to the mutex handle

Returns

OSAL_RESULT_TRUE - Mutex successfully created

OSAL_RESULT_FALSE - Mutex failed to be created

Remarks

None.

Example

```
OSAL_MUTEX_HANDLE_TYPE mutexData;  
OSAL_MUTEX_Create(&mutexData);  
...  
if (OSAL_MUTEX_Lock(mutexData, 1000) == OSAL_RESULT_TRUE)  
{  
    // manipulate the shared data  
    ...  
}
```

5.5.7.3.2 OSAL_MUTEX_Delete Function

C

```
OSAL_RESULT OSAL_MUTEX_Delete(  
    OSAL_MUTEX_HANDLE_TYPE* mutexID  
);
```

Description

This function deletes a mutex and frees associated storage if required.

Preconditions

None.

Parameters

Parameters	Description
mutexID	Pointer to the mutex handle

Returns

OSAL_RESULT_TRUE - Mutex successfully deleted

OSAL_RESULT_FALSE - Mutex failed to be deleted

Remarks

None.

Example

```
OSAL_MUTEX_Delete(&mutexData);
```

5.5.7.3.3 OSAL_MUTEX_Lock Function

C

```
OSAL_RESULT OSAL_MUTEX_Lock(  
    OSAL_MUTEX_HANDLE_TYPE mutexID,  
    uint16_t waitMS  
);
```

Description

This function locks a mutex, waiting for the specified timeout. If it cannot be obtained or the timeout period elapses 'false' is returned.

Preconditions

Mutex must have been created.

Parameters

Parameters	Description
mutexID	The mutex handle
waitMS	Timeout value in milliseconds 0, do not wait return immediately OSAL_WAIT_FOREVER , wait until mutex is obtained before returning Other values, Timeout delay

Returns

OSAL_RESULT_TRUE - Mutex successfully obtained

OSAL_RESULT_FALSE - Mutex failed to be obtained or timeout occurred

Remarks

None.

Example

```
OSAL_MUTEX_HANDLE_TYPE mutexData;  
  
OSAL_MUTEX_Create(&mutexData);  
  
...  
if (OSAL_MUTEX_Lock(mutexData, 1000) == OSAL_RESULT_TRUE)  
{  
    // manipulate the shared data  
    ...  
  
    // unlock the mutex  
    OSAL_MUTEX_Unlock(mutexData);  
}
```

5.5.7.3.4 OSAL_MUTEX_Unlock Function

C

```
OSAL_RESULT OSAL_MUTEX_Unlock(  
    OSAL_MUTEX_HANDLE_TYPE mutexID  
);
```

Description

This function unlocks a previously obtained mutex.

Preconditions

Mutex must have been created.

Parameters

Parameters	Description
mutexID	The mutex handle

Returns

OSAL_RESULT_TRUE - Mutex released

OSAL_RESULT_FALSE - Mutex failed to be released or error occurred

Remarks

None.

Example

```
OSAL_MUTEX_HANDLE_TYPE mutexData;  
OSAL_MUTEX_Create(&mutexData);  
...  
if (OSAL_MUTEX_Lock(mutexData, 1000) == OSAL_RESULT_TRUE)  
{  
    // manipulate the shared data  
    ...  
    // unlock the mutex  
    OSAL_MUTEX_Unlock(mutexData);  
}
```

5.5.7.3.5 OSAL_MUTEX_DECLARE Macro

C

```
#define OSAL_MUTEX_DECLARE(mutexID) OSAL_MUTEX_HANDLE_TYPE mutexID
```

Description

OSAL_MUTEX_Declare(mutexID)

This function declares a data item of type [OSAL_MUTEX_HANDLE_TYPE](#)

Remarks

None

5.5.7.4 Queue Functions

5.5.7.4.1 OSAL_QUEUE_DECLARE Macro

C

```
#define OSAL_QUEUE_DECLARE(queueID) OSAL_QUEUE_HANDLE_TYPE queueID
```

Description

OSAL_QUEUE_Declare(queueID)

This function declares a data item of type [OSAL_QUEUE_HANDLE_TYPE](#).

Remarks

None

5.5.7.4.2 OSAL_QUEUE_Add Function

C

```
OSAL_RESULT OSAL_QUEUE_Add(  
    OSAL_QUEUE_HANDLE_TYPE queueID,  
    uint8_t* pdata,  
    uint16_t waitMS  
);
```

Description

This function adds the specified item to the queue by copying it. If the queue has space, the function will return immediately. If there is no space on the queue, the function will wait up to the specified time before returning. The item is added to the tail of the queue.

Preconditions

Queue must have been created.

Parameters

Parameters	Description
queueID	The OSAL_QUEUE handle
pdata	Pointer to the data to be copied to the queue. The data item must be the same size as that specified when creating the queue.
waitMS	Time to wait while trying to write to a full queue before timing out

Returns

OSAL_RESULT_TRUE - Item successfully added to the queue

OSAL_RESULT_FALSE - Item could not be written to the queue or a timeout occurred

Remarks

None.

Example

```
// create a queue containing 16 byte items, queue is 5 items long.  
OSAL_QUEUE_Create(&msgQueue, 16, 5);  
  
...  
// define a message (Note 0 at end is allowed for)  
const char msg[16] = "Hello World....";  
  
// write data to the queue. Wait until space is available  
// Note in this case that the 0 at the end of msg is also copied
```

```
OSAL_QUEUE_Add(msgQueue, (uint8_t*) msg, OSAL_WAIT_FOREVER);
```

5.5.7.4.3 OSAL_QUEUE_AddHead Function

C

```
OSAL_RESULT OSAL_QUEUE_AddHead(  
    OSAL_QUEUE_HANDLE_TYPE queueID,  
    uint8_t* pdata,  
    uint16_t waitMS  
);
```

Description

This function adds the specified item to the front of the queue by copying it. If the queue has space the function will return immediately. If there is no space on the queue then wait up to the specified time before returning. The item is added to the front of the queue.

Preconditions

Queue must have been created.

Parameters

Parameters	Description
queueID	The OSAL_QUEUE handle
pdata	Pointer to the data to be copied to the queue. The data item must be the same size as that specified when creating the queue.
waitMS	Time to wait while trying to write to a full queue before timing out

Returns

OSAL_RESULT_TRUE - Item successfully added to the queue
OSAL_RESULT_FALSE - Item could not be written to the queue or a timeout occurred
OSAL_RESULT_NOT_IMPLEMENTED - Function not implemented in the current RTOS

Remarks

None.

Example

```
// create a queue containing byte items, queue is 5 items long.  
OSAL_QUEUE_Create(&msgQueue, sizeof(uint8_t), 5);  
  
...  
// Send an important message  
uint8_t sigMsg = 42;  
  
// write data to the front of the queue.  
OSAL_QUEUE_AddHead(msgQueue, &sigMsg, OSAL_WAIT_FOREVER);
```

5.5.7.4.4 OSAL_QUEUE_AddHeadISR Function

C

```
OSAL_RESULT OSAL_QUEUE_AddHeadISR(  
    OSAL_QUEUE_HANDLE_TYPE queueID,  
    uint8_t* pdata  
);
```

Description

This function adds the specified item to the queue by copying it. The item is added to the front of the queue. This version of Add

should be used within an ISR.

Preconditions

Queue must have been created.

Parameters

Parameters	Description
queueID	The OSAL_QUEUE handle
pdata	Pointer to the data to be copied to the queue. The data item must be the same size as that specified when creating the queue.

Returns

OSAL_RESULT_TRUE - Item successfully added to the queue

OSAL_RESULT_FALSE - Item could not be written to the queue

OSAL_RESULT_NOT_IMPLEMENTED - Function not implemented in the current RTOS

Remarks

None.

Example

```
void __ISR(EXTERNAL_VECTOR) _ExternalIRQHandler()
{
    const uint8_t msg = EMERGENCY_STOP;

    OSAL_ISR_Enter();

    // clear the interrupt flag
    IFS1bit.EIF = 0;

    // send an emergency message to any listening task
    OSAL_QUEUE_AddHeadISR(myRXQueue, msg);

    OSAL_ISR_Exit();
}
```

5.5.7.4.5 OSAL_QUEUE_AddISR Function

C

```
OSAL_RESULT OSAL_QUEUE_AddISR(
    OSAL_QUEUE_HANDLE_TYPE queueID,
    uint8_t* pdata
);
```

Description

This function adds the specified item to the queue by copying it. The item is added to the tail of the queue. This version of Add should be used within an ISR.

Preconditions

Queue must have been created.

Parameters

Parameters	Description
queueID	The OSAL_QUEUE handle
pdata	Pointer to the data to be copied to the queue. The data item must be the same size as that specified when creating the queue.

Returns

OSAL_RESULT_TRUE - Item successfully to the queue

OSAL_RESULT_FALSE - Item could not be added to the queue

Remarks

None.

Example

```
void __ISR(UART_2_VECTOR) _UART2RXHandler()  
{  
    char c;  
  
    OSAL_ISR_Enter();  
  
    // read the character  
    c = U2RXREG;  
    // clear the interrupt flag  
    IFS1bits.U2IF = 0;  
  
    // post the received character onto a queue for later processing  
    OSAL_QUEUE_AddISR(myRXQueue, c);  
  
    OSAL_ISR_Exit();  
}
```

5.5.7.4.6 OSAL_QUEUE_Create Function

C

```
OSAL_RESULT OSAL_QUEUE_Create(  
    OSAL_QUEUE_HANDLE_TYPE* queueID,  
    uint16_t width,  
    uint16_t depth  
);
```

Description

This function allocates storage and creates an OSAL queue entity. Each entry in the queue will be 'width' bytes in size and the queue can contain at most 'depth' items.

Preconditions

Queue must have been declared.

Parameters

Parameters	Description
queueID	Pointer to the OSAL queue
width	Width in bytes of each item in the queue
depth	Maximum number of entries that can be stored in the queue

Returns

OSAL_RESULT_TRUE - Queue created

OSAL_RESULT_FALSE - Queue failed to be created

Remarks

The function creates an OSAL_QUEUE with the specified parameters. OSAL_QUEUES are assumed to work on copy with passed data being copied into the queue or other temporary storage so that the passed data storage is released.

Example

```
// declare a queue variable
OSAL_QUEUE_Declare(serialQueue);

// create a queue that contains 16bit integers and is 10 items long
OSAL_QUEUE_Create(&serialQueue, sizeof(uint16_t), 10);
```

5.5.7.4.7 OSAL_QUEUE_Delete Function

C

```
OSAL_RESULT OSAL_QUEUE_Delete(
    OSAL_QUEUE_HANDLE_TYPE* queueID
);
```

Description

This function deletes an OSAL_QUEUE pointed to be the passed argument. Associated storage is released upon completion.

Preconditions

Queue must have been created.

Parameters

Parameters	Description
queueID	Pointer to the OSAL_QUEUE object handle

Returns

OSAL_RESULT_TRUE - Queue deleted

OSAL_RESULT_FALSE - Queue failed to be deleted

Remarks

None.

Example

```
// declare a queue variable
OSAL_QUEUE_Declare(serialQueue);

// create a queue that contains 16bit integers and is 10 items long
OSAL_QUEUE_Create(&serialQueue, sizeof(uint16_t), 10);

// application is terminating, delete queue object
OSAL_QUEUE_Delete(&serialQueue);
```

5.5.7.4.8 OSAL_QUEUE_Peek Function

C

```
OSAL_RESULT OSAL_QUEUE_Peek(
    OSAL_QUEUE_HANDLE_TYPE queueID,
    uint8_t* pdata,
    uint16_t waitMS
);
```

Description

This function examines the queue and copy the item at the head leaving the item on the queue for the next access. If there is nothing on the queue, block until waitMS milliseconds have elapsed before returning, or until something is placed on the queue.

Preconditions

Queue must have been created.

Parameters

Parameters	Description
queueID	The OSAL_QUEUE handle
pdata	Pointer to the location to receive data copied from the queue. The data item must be the same size as that specified when creating the queue.
waitMS	Maximum time to wait before returning if the queue is empty 0, return immediately OSAL_WAIT_FOREVER , wait until data is in the queue Others, maximum blocking time in milliseconds

Returns

OSAL_RESULT_TRUE - Item successfully copied from the queue

OSAL_RESULT_FALSE - Nothing copied from the queue or timeout occurred

OSAL_RESULT_NOT_IMPLEMENTED - Operation not available on the current RTOS

Remarks

None.

Example

```
// copy filenames from a queue and process the data in the file
// however if another filename is on the queue then abort and
// restart the operation
OSAL_QUEUE_Remove(rxQueue, fileName, OSAL_WAITFOREVER);

while (abort == false)
{
    // read data from the file and process it
    ...

    // check for a new filename to process
    if (OSAL_QUEUE_Peek(rxQueue, fileName, 0) == OSAL_RESULT_TRUE)
    {
        abort = true;
    }
}
```

5.5.7.4.9 OSAL_QUEUE_PeekISR Function**C**

```
OSAL_RESULT OSAL_QUEUE_PeekISR(
    OSAL_QUEUE_HANDLE_TYPE queueID,
    uint8_t* pdata
);
```

Description

This function examines the queue and copies the item at the head, leaving the item on the queue for the next access. This version should be called from inside an ISR.

Preconditions

Queue must have been created.

Parameters

Parameters	Description
queueID	The OSAL_QUEUE handle
pdata	Pointer to the location to receive data copied from the queue. The data item must be the same size as that specified when creating the queue.

Returns

OSAL_RESULT_TRUE - Item successfully copied from the queue

OSAL_RESULT_FALSE - Nothing copied from the queue

OSAL_RESULT_NOT_IMPLEMENTED - Operation not available on the current RTOS

Remarks

None.

Example

```
uint16_t data;  
OSAL_QUEUE_PeekISR(myQueue, &data);
```

5.5.7.4.10 OSAL_QUEUE_Remove Function

C

```
OSAL_RESULT OSAL_QUEUE_Remove(  
    OSAL_QUEUE_HANDLE_TYPE queueID,  
    uint8_t* pdata,  
    uint16_t waitMS  
);
```

Description

This function removes the item at the head of the queue. The item is copied to the location pointed to by pdata. The user should ensure that there is sufficient space to receive the copied data. If no data is available, the function will block for waitMS before returning.

Preconditions

Queue must have been created.

Parameters

Parameters	Description
queueID	The OSAL_QUEUE handle
pdata	Pointer to the location to receive data copied from the queue. The data item must be the same size as that specified when creating the queue.
waitMS	Maximum time to wait before returning if the queue is empty 0, return immediately OSAL_WAIT_FOREVER , wait until data is in the queue Others, maximum blocking time in milliseconds

Returns

OSAL_RESULT_TRUE - Item successfully removed from the queue

OSAL_RESULT_FALSE - Nothing removed from the queue or timeout occurred

Remarks

None.

Example

```
char rxChar;
// wait up to one second to process the next character
if (OSAL_QUEUE_Remove(rxQueue, &rxChar, 1000) == OSAL_RESULT_TRUE)
{
    // process the data
    ...
}
else
{
    // nothing received in the last one second so perform other processing
    ...
}
```

5.5.7.4.11 OSAL_QUEUE_RemoveISR Function**C**

```
OSAL_RESULT OSAL_QUEUE_RemoveISR(
    OSAL_QUEUE_HANDLE_TYPE queueID,
    uint8_t* pdata
);
```

Description

This function removes the item at the head of the queue. Item is copied to the location pointed to by pdata. The user should ensure that there is sufficient space to receive the copied data. This version of the queue remove function should be used inside an ISR.

Preconditions

Queue must have been created.

Parameters

Parameters	Description
queueID	The OSAL_QUEUE handle
pdata	Pointer to the location to receive data copied from the queue. The data item must be the same size as that specified when creating the queue.

Returns

OSAL_RESULT_TRUE - Item successfully removed from the queue

OSAL_RESULT_FALSE - Nothing removed from the queue

Remarks

None.

Example

```
char rxChar;
void __ISR(TIMER_1_VECTOR) _ET1IRQHandler()
{
    const uint8_t msg;

    OSAL_ISR_Enter();

    // clear the interrupt flag
    IFS0bits.T1IF = 0;

    // check for data to be sent from a queue
    if (OSAL_QUEUE_RemoveISR(myTXQueue, msg) == OSAL_RESULT_TRUE)
    {

```

```
        DRV_USART_WriteByte(uart_drv, msg);
    }
    OSAL_ISR_Exit();
}
```

5.5.7.5 Critical Section Functions

5.5.7.5.1 OSAL_CRIT_Enter Function

C

```
void OSAL_CRIT_Enter(
    OSAL_CRIT_TYPE severity
);
```

Description

Critical Section group

This function enters a critical section of code. It is assumed that the sequence of operations bounded by the enter and leave critical section operations is treated as one atomic sequence that will not be disturbed.

Preconditions

None.

Parameters

Parameters	Description
severity	OSAL_CRIT_TYPE_LOW, The RTOS should disable all other running tasks effectively locking the scheduling mechanism. OSAL_CRIT_TYPE_HIGH, The RTOS should disable all possible interrupts sources including the scheduler ensuring that the sequence of code operates without interruption.

Returns

None.

Remarks

The sequence of operations bounded by the OSAL_CRIT_Enter and [OSAL_CRIT_Leave](#) form a critical section. The severity level defines whether the RTOS should perform task locking or completely disable all interrupts.

Example

```
// prevent other tasks pre-empting this sequence of code
OSAL_CRIT_Enter(OSAL_CRIT_TYPE_LOW);
// modify the peripheral
DRV_USART_Reinitialize( objUSART, &initData);
OSAL_CRIT_Leave(OSAL_CRIT_TYPE_LOW);
```

5.5.7.5.2 OSAL_CRIT_Leave Function

C

```
void OSAL_CRIT_Leave(
    OSAL_CRIT_TYPE severity
);
```

Description

This function leaves a critical section of code. It is assumed that the sequence of operations bounded by the enter and leave critical section operations is treated as one atomic sequence that will not be disturbed. The severity should match the severity level used in the corresponding [OSAL_CRIT_Enter](#) call to ensure that the RTOS carries out the correct action.

Preconditions

None.

Parameters

Parameters	Description
severity	OSAL_CRIT_TYPE_LOW, The RTOS should disable all other running tasks effectively locking the scheduling mechanism. OSAL_CRIT_TYPE_HIGH, The RTOS should disable all possible interrupts sources including the scheduler ensuring that the sequence of code operates without interruption.

Returns

None.

Remarks

The sequence of operations bounded by the [OSAL_CRIT_Enter](#) and [OSAL_CRIT_Leave](#) form a critical section. The severity level defines whether the RTOS should perform task locking or completely disable all interrupts.

Example

```
// prevent other tasks pre-empting this sequence of code
OSAL_CRIT_Enter(OSAL_CRIT_TYPE_LOW);
// modify the peripheral
DRV_USART_Reinitialize( objUSART,  &initData);
OSAL_CRIT_Leave(OSAL_CRIT_TYPE_LOW);
```

5.5.7.5.3 OSAL_CRIT_TYPE Enumeration

C

```
enum OSAL_CRIT_TYPE {
    OSAL_CRIT_TYPE_LOW,
    OSAL_CRIT_TYPE_HIGH
};
```

Description

OSAL Critical Type

This enum represents possible critical section types.

OSAL_CRIT_TYPE_LOW - Low priority critical section, can be formed by locking the scheduler (if supported by RTOS)

OSAL_CRIT_TYPE_HIGH - High priority critical section, will be formed by disabling all interrupts.

Remarks

Critical section types

5.5.7.6 Interrupt Handling Functions

5.5.7.6.1 OSAL_ISR_Enter Function

C

```
void OSAL_ISR_Enter();
```

Description

This function allows the OSAL and RTOS to implement any set up required should an OSAL function be called during the interrupt handler. Depending on the underlying RTOS, it may be necessary to set a global flag indicating that an interrupt is currently being executed for the correct context code to be executed during calls to the kernel. OSAL_ISR_Enter allows the RTOS to implement the required functions.

Preconditions

None, (in an interrupt handler).

Returns

None.

Remarks

None.

Example

```
void __ISR(UART_2_VECTOR) _UART2RXHandler()
{
    char c;

    OSAL_ISR_Enter();

    // perform interrupt processing and clear the flag
    ...

    OSAL_ISR_Exit();
}
```

5.5.7.6.2 OSAL_ISR_Exit Function

C

```
void OSAL_ISR_Exit();
```

Description

This function allows the OSAL and RTOS to implement any restoration required after an ISR has completed. Depending on the underlying RTOS, it may be necessary to set a global flag indicating that an interrupt is currently being executed for the correct context code to be executed during calls to the kernel. OSAL_ISR_Exit allows the RTOS to implement the required functions.

Preconditions

None, (in an interrupt handler).

Returns

None.

Remarks

None.

Example

```
void __ISR(UART_2_VECTOR) _UART2RXHandler()
{
```

```
char c;

OSAL_ISR_Enter();

// perform interrupt processing and clear the flag
...

OSAL_ISR_Exit();
}
```

5.5.7.7 Error Handling Functions

5.5.7.7.1 OSAL_ERROR_CALLBACK_TYPE Enumeration

C

```
enum OSAL_ERROR_CALLBACK_TYPE {
    OSAL_ERROR_CALLBACK_DEFAULT = 0,
    OSAL_ERROR_CALLBACK_OUT_OF_MEMORY,
    OSAL_ERROR_CALLBACK_STACK_OVERFLOW,
    OSAL_ERROR_CALLBACK_ERROR
};
```

Description

OSAL Error Callback Type

This enum represents possible OSAL error callback types.

OSAL_ERROR_CALLBACK_DEFAULT - Default error
OSAL_ERROR_CALLBACK_OUT_OF_MEMORY - Out of memory allocating resource such as creating a task or queue
OSAL_ERROR_CALLBACK_STACK_OVERFLOW - Stack overflow detected in a task
OSAL_ERROR_CALLBACK_ERROR - Generic error message

Remarks

Error types are heavily RTOS specific so only basic types are indicated

5.5.7.7.2 OSAL_ErrorCallback Function

C

```
void OSAL_ErrorCallback(
    OSAL_ERROR_CALLBACK_TYPE type
);
```

Description

The callback function will be executed whenever an error occurs within an underlying RTOS. The OSAL implements this function as a weak type allowing it to be overridden by the user. The type and severity of errors that can be encountered by an RTOS will vary according to the specific implementation so this only provides lightweight error handling. There is no expectation that the function should return and if this does happen there is no guarantee that the RTOS will continue to function properly. A user application could override this function and provide simple signalling of the error such as logging a message or flashing a light.

Preconditions

None.

Parameters

Parameters	Description
type	Type of error OSAL_ERROR_CALLBACK_DEFAULT, Default error OSAL_ERROR_CALLBACK_OUT_OF_MEMORY, Memory allocation error OSAL_ERROR_CALLBACK_STACK_OVERFLOW, Stack overflow in a thread OSAL_ERROR_CALLBACK_ERROR, Generic error

Returns

None

Remarks

None.

Example

```
// application override of the callback
void OSAL_ErrorCallback(OSAL_ERROR_CALLBACK_TYPE type)
{
    // turn on the fault light
    LATAbits.LATA0 = 1;
    // do not return
    while (1)
    {
    }
}
```

5.5.7.8 Memory Allocation Functions

5.5.7.8.1 OSAL_Free Function

C

```
void OSAL_Free(
    void* pData
);
```

Description

This function deallocates memory and returns it to the default pool. In an RTOS-based application, the memory may have been allocated from multiple pools or simply from the heap. In non-RTOS applications, this function calls the C standard function free.

Preconditions

None.

Parameters

Parameters	Description
pData	Pointer to the memory block to be set free

Returns

None.

Remarks

None.

Example

```
// create a working array
uint8_t* pData;

pData = OSAL_Malloc(32);
if (pData != NULL)
{
    ...

    // deallocate the memory
    OSAL_Free(pData);
    // and prevent it accidentally being used again
    pData = NULL;
}
```

5.5.7.8.2 OSAL_Malloc Function

C

```
void* OSAL_Malloc(
    size_t size
);
```

Description

This function allocates a block of memory from the default allocator from the underlying RTOS. If no RTOS is present, it defaults to malloc. Many operating systems incorporate their own memory allocation scheme, using pools, blocks or by wrapping the standard C library functions in a critical section. Since an Harmony application may not know what target OS is being used (if any), this function ensures that the correct thread safe memory allocator will be used.

Preconditions

None.

Parameters

Parameters	Description
size	Size of the requested memory block in bytes

Returns

Pointer to the block of allocated memory. NULL is returned if memory could not be allocated.

Remarks

None.

Example

```
// create a working array
uint8_t* pData;

pData = OSAL_Malloc(32);
if (pData != NULL)
{
    ...
}
```

5.5.7.9 OSAL Control Functions

5.5.7.9.1 OSAL_Initialize Function

C

```
OSAL_RESULT OSAL_Initialize();
```

Description

This function performs OSAL initialization. This function should be called near the start of main in an application that will use an underlying RTOS. This permits the RTOS to perform any one time initialization before the application attempts to create drivers or other items that may use the RTOS. Typical actions performed by OSAL_Initialize would be to allocate and prepare any memory pools for later use.

Preconditions

None.

Returns

OSAL_RESULT_TRUE - Initialization completed successfully.

Remarks

None.

Example

```
int main()
{
    OSAL_Initialize();

    App_Init();
    OSAL_Start();
}
```

5.5.7.9.2 OSAL_Name Function

C

```
const char* OSAL_Name();
```

Description

This function returns a const char* to the textual name of the RTOS. The name is a NULL terminated string.

Preconditions

None.

Returns

const char* - Name of the underlying RTOS or NULL

Remarks

None.

Example

```
// get the RTOS name
const char* sName;

sName = OSAL_Name();
sprintf(buff, "RTOS: %s", sName);
```

5.5.7.9.3 OSAL_Start Function

C

```
OSAL_RESULT OSAL_Start();
```

Description

This function starts the underlying RTOS application. The application will not normally return from this function call since the RTOS and interrupts will be responsible for all program operation. A return from this function is usually fatal and indicates that the program has failed in some way.

Preconditions

None.

Returns

OSAL_RESULT_FALSE - The OS terminated for some reason

Remarks

None.

Example

```
int main()
{
    OSAL_Initialize();

    App_Init();
    OSAL_Start();
}
```

5.5.7.10 Data Types and Constants

5.5.7.10.1 OSAL_MUTEX_HANDLE_TYPE Macro

C

```
#define OSAL_MUTEX_HANDLE_TYPE uint8_t
```

Description

Handle to a mutex, this is the default implementation and may be overridden in an actual implementation file

5.5.7.10.2 OSAL_QUEUE_HANDLE_TYPE Macro

C

```
#define OSAL_QUEUE_HANDLE_TYPE uint8_t
```

Description

Handle to a queue, this is the default implementation and may be overridden in an actual implementation file

5.5.7.10.3 OSAL_SEM_HANDLE_TYPE Macro

C

```
#define OSAL_SEM_HANDLE_TYPE uint8_t
```

Description

Handle to a semaphore, this is the default implementation and may be overridden in an actual implementation file

5.5.7.10.4 OSAL_THREAD_HANDLE_TYPE Macro**C**

```
#define OSAL_THREAD_HANDLE_TYPE uintptr_t
```

Description

Handle to a thread, this is the default implementation and may be overridden in an actual implementation file

5.5.7.10.5 OSAL_USE_NONE Macro**C**

```
#define OSAL_USE_NONE
```

Description

This is macro OSAL_USE_NONE.

5.5.7.10.6 OSAL_WAIT_FOREVER Macro**C**

```
#define OSAL_WAIT_FOREVER (uint16_t) 0xFFFF
```

Description

Default value indicating block forever

5.5.7.10.7 OSAL_RESULT Enumeration**C**

```
enum OSAL_RESULT {  
    OSAL_RESULT_NOT_IMPLEMENTED = -1,  
    OSAL_RESULT_FALSE = 0,  
    OSAL_RESULT_TRUE = 1  
};
```

Description

OSAL Result type

This enum represents possible return types from OSAL functions.

Remarks

These enum values are the possible return values from OSAL functions where a standard success/fail type response is required. The majority of OSAL functions will return this type with a few exceptions.

5.5.7.10.8 OSAL_SEM_TYPE Enumeration**C**

```
enum OSAL_SEM_TYPE {  
    OSAL_SEM_TYPE_BINARY,
```

```
    OSAL_SEM_TYPE_COUNTING
};
```

Description

OSAL Semaphore Type

This enum represents possible semaphore types.

OSAL_SEM_TYPE_BINARY - Simple binary type that can be taken once

OSAL_SEM_TYPE_COUNTING - Complex type that can be taken set number of times defined at creation time

Remarks

Binary and counting semaphore type.

5.5.8 Files

Files

Name	Description
osal.h	Common interface definitions for the Operating System Abstraction Layer (OSAL).

Description

5.5.8.1 osal.h

Descriptive File Name: Operating System Abstraction Layer

This file defines the common interface to the Operating System Abstraction Layer. It defines the common types used by the OSAL and defines the function prototypes. Depending upon the OSAL mode, a support level specific implementation file is included by this file to give the required level of compatibility. The available support levels include, [OSAL_USE_NONE](#), [OSAL_USE_BASIC](#), and [OSAL_USE_RTOS](#).

Enumerations

	Name	Description
	OSAL_CRIT_TYPE	Enumerated type representing the possible types of critical section.
	OSAL_ERROR_CALLBACK_TYPE	Enumerated type representing the possible types of callback message.
	OSAL_RESULT	Enumerated type representing the general return value from OSAL functions.
	OSAL_SEM_TYPE	Enumerated type representing the possible types of semaphore.

Functions

	Name	Description
	OSAL_CRIT_Enter	Enters a critical section with the specified severity level.
	OSAL_CRIT_Leave	Leaves a critical section with the specified severity level.
	OSAL_ErrorCallback	Lightweight error handling callback function for the RTOS.
	OSAL_Free	Deallocates a block of memory and return to the default pool.
	OSAL_Initialize	Performs OSAL initialization.
	OSAL_ISR_Enter	Informs the OSAL and RTOS that an ISR is being entered.
	OSAL_ISR_Exit	Informs the OSAL and RTOS that an ISR is being exited.

	OSAL_Malloc	Allocates memory using the OSAL default allocator.
	OSAL_MUTEX_Create	Creates a mutex.
	OSAL_MUTEX_Delete	Deletes a mutex.
	OSAL_MUTEX_Lock	Locks a mutex.
	OSAL_MUTEX_Unlock	Unlocks a mutex.
	OSAL_Name	Obtains the name of the underlying RTOS.
	OSAL_QUEUE_Add	Adds the specified item to the queue with a time-out if the queue is full.
	OSAL_QUEUE_AddHead	Adds the specified item to the front of the queue with a timeout if the queue is full.
	OSAL_QUEUE_AddHeadISR	Adds the specified item to the front of the queue from within an ISR.
	OSAL_QUEUE_AddISR	Adds the specified item to the queue from with an ISR.
	OSAL_QUEUE_Create	Creates a queue with the specified width and depth.
	OSAL_QUEUE_Delete	Deletes an OSAL_QUEUE.
	OSAL_QUEUE_Peek	Copys the item at the head of the queue but leave it on the queue. If the queue is empty, it will block up to waitMS.
	OSAL_QUEUE_PeekISR	Copys the item at the head of the queue but leaves it on the queue.
	OSAL_QUEUE_Remove	Removes the item at the head of the queue. If the queue is empty it will block up to waitMS.
	OSAL_QUEUE_RemoveISR	Removes the item at the head of the queue.
	OSAL_SEM_Create	Creates an OSAL Semaphore.
	OSAL_SEM_Delete	Deletes an OSAL Semaphore.
	OSAL_SEM_GetCount	Returns the current value of a counting semaphore.
	OSAL_SEM_Pend	Waits on a semaphore. Returns true if the semaphore was obtained within the time limit.
	OSAL_SEM_Post	Posts a semaphore or increments a counting semaphore.
	OSAL_SEM_PostISR	Posts a semaphore or increments a counting semaphore from within an Interrupt Service Routine (ISR).
	OSAL_Start	Starts the underlying RTOS
	OSAL_THREAD_Create	Creates an OSAL thread with the specified parameters.
	OSAL_THREAD_CreateDaemon	Creates a Daemon thread for use with multi-threaded Harmony drivers.
	OSAL_THREAD_PriorityGet	Gets the priority of a thread.
	OSAL_THREAD_PrioritySet	Sets the priority of a thread.
	OSAL_THREAD_Resume	Resumes a specified thread.
	OSAL_THREAD_ResumeAll	Resumes all suspended threads.
	OSAL_THREAD_Sleep	Puts the currently executing thread to sleep.
	OSAL_THREAD_Suspend	Suspends a specified thread.
	OSAL_THREAD_SuspendAll	Suspends all threads apart from the calling thread.

Macros

	Name	Description
	OSAL_MUTEX_DECLARE	Declares an OSAL mutex.
	OSAL_MUTEX_HANDLE_TYPE	Handle to a mutex, this is the default implementation and may be overridden in an actual implementation file
	OSAL_QUEUE_DECLARE	Declares an OSAL queue.
	OSAL_QUEUE_HANDLE_TYPE	Handle to a queue, this is the default implementation and may be overridden in an actual implementation file
	OSAL_SEM_DECLARE	Declares an OSAL semaphore.

	<u>OSAL_SEM_HANDLE_TYPE</u>	Handle to a semaphore, this is the default implementation and may be overridden in an actual implementation file
	<u>OSAL_THREAD_HANDLE_TYPE</u>	Handle to a thread, this is the default implementation and may be overridden in an actual implementation file
	<u>OSAL_USE_NONE</u>	This is macro OSAL_USE_NONE.
	<u>OSAL_WAIT_FOREVER</u>	Default value indicating block forever

Types

	Name	Description
	<u>OSAL_THREAD_FUNCTION</u>	Functions that will operate as a separate thread under the OSAL should match this type.

File Name

osal.h

Company

Microchip Technology Inc.

5.6 Peripheral Library Help

This section provides descriptions of the MPLAB Harmony peripheral libraries.

Description

This section contains the list of peripheral libraries.

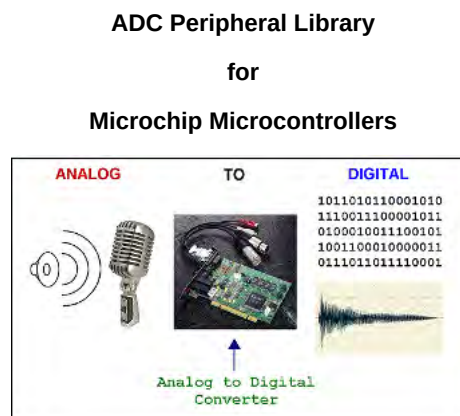
5.6.1 ADC Peripheral Library

This topic describes the Analog-to-Digital Converter peripheral library interface.

Description

This section contains the list of topics.

5.6.1.1 Introduction



This library provides a low-level abstraction of the Analog-to-Digital Converter (ADC) module, which is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

ADC Overview

An ADC is an analog peripheral that converts a continuous quantity to a discrete digital number. An ADC is a signal conversion process that periodically samples and converts a continuously varying signal - analog level into digital values. An ADC might be used to make an isolated measurement. ADCs are also used to quantize time-varying signals by turning them into a sequence of digital samples. This results in the signal being quantized in both time and value. The resolution of a converter indicates the number of discrete values it can produce over the range of analog values. This library currently supports 8-bit, 10-bit, and 12-bit resolution.

5.6.1.2 Release Notes

MPLAB Harmony Version: v0.70b **ADC Peripheral Library Version:** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.1.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED AS IS MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.1.4 Using the Library

Using the Library

This topic describes the basic architecture of the ADC Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_adc.h](#)

The interface to the ADC library is defined in the "[plib_adc.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the ADC library must include "peripheral.h".

Library File:

The ADC peripheral library is part of the processor-specific peripheral library installed with the compiler. This library is automatically available (in the default search path) for any project built using a Microchip compiler.

5.6.1.4.1 Hardware Abstraction Model

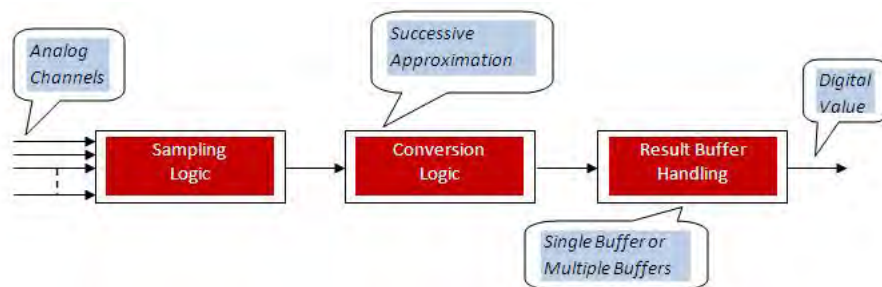
This library provides a low-level abstraction of the ADC module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model

The ADC module accepts an analog signal at any one instance and converts it to a corresponding 8-bit, 10-bit, or 12-bit digital value. It can accommodate a number of analog inputs and separate reference inputs; the actual number available on a particular device depends on the package size.

Hardware Abstraction Block Diagram



A combination of input multiplexers can select the signal to be converted from multiple analog input pins. The entire multiplexer path includes provision for differential analog input, although the number of negative input pins is limited, and the signal difference must remain positive (i.e., unipolar).

Sampling Logic:

An internal Sample and Hold (S&H) circuit acquires a sample of an input signal, and then holds that value constant during the conversion process. The purpose of the S&H circuitry is to take a snapshot of the sensor signal and hold the value. The sampled voltage is held and converted to a digital value, which strictly speaking, represents the ratio of that input voltage to a reference voltage. Configuration choices can allow connection of an external reference or use of the device power and ground (AVDD and AVSS).

Conversion Logic:

The heart of the ADC is the conversion logic that converts the analog signal value into its equivalent discrete representation. Conversions can be started individually by program control, continuously free-running, or triggered by selected hardware events. A single channel may be repeatedly converted, alternate conversions may be performed on two channels, or any or all of the channels may be sequentially scanned and converted according to a user-defined bit map.

Result Handling:

The resulting conversion output is a 8-bit, 10-bit, or 12-bit digital number that can be signed or unsigned, left- or right-justified in a 16-bit word.

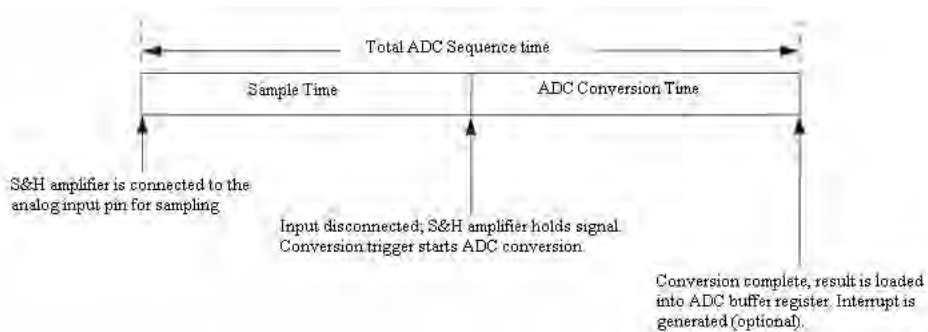
ADC Timing Details:

Sample time is the time that the ADC module's S&H circuit is connected to the analog input pin. The sample time may be started and ended automatically by the ADC's hardware or under direct program control. There is a minimum sample time to ensure that the S&H circuit will provide sufficient accuracy for the analog-to-digital conversion.

Conversion time is the time required for the ADC to convert the voltage held by the S&H circuit. The conversion trigger ends the sampling time and begins an analog-to-digital conversion or a repeating sequence. The conversion trigger sources can be taken from a variety of hardware sources or can be controlled directly in software.

Once the conversion is complete, the S&H circuit can be reconnected to the input pin and a CPU interrupt may be generated. The sum of the sample time and the analog-to-digital conversion time provides the total ADC sequence time. The following figure shows the basic conversion sequence and the relationship between intervals.

ADC Sample/Convert Sequence



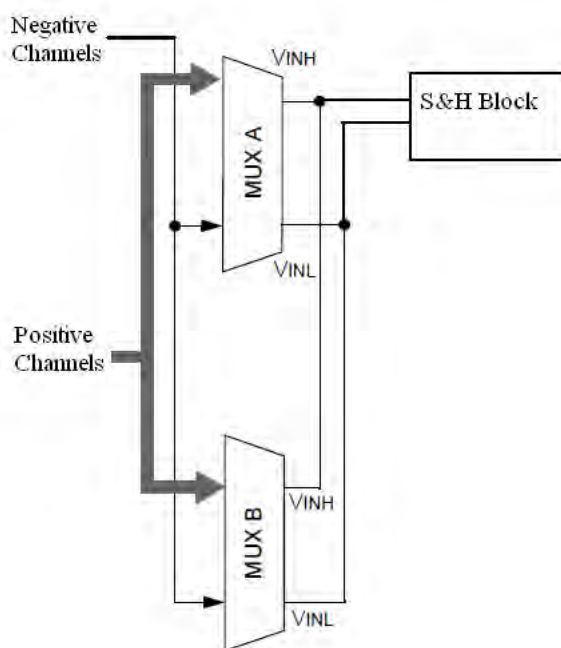
The conversion trigger sources can be taken from a variety of hardware sources, or can be controlled directly by software. One of the conversion trigger options is an auto-conversion, which uses a counter and the ADC clock to set the time between auto-conversions. The Auto-Sample mode and auto-conversion trigger can be used together to provide continuous automatic conversions without software intervention.

A sample/convert sequence that uses multiple S&H channels can be simultaneously sampled or sequentially sampled. Simultaneously sampling multiple signals ensures that the snapshot of the analog inputs occurs at precisely the same time for all inputs. Sequential sampling takes a snapshot of each analog input just before conversion starts on that input. The sampling of multiple inputs is not correlated.

Channel Multiplexers:

On some devices, S&H circuits have analog multiplexers on both their non-inverting and inverting inputs to select which analog input(s) are sampled. The ADC of some devices incorporate two independent sets of input multiplexers (MUX A and MUX B), which allow users to choose the analog channels that are to be sampled. Functionally, MUX A and MUX B are very similar to each other. Both multiplexers allow any of the analog input channels to be selected for individual sampling and allow selection of a negative reference source for differential signals. In addition, MUX A can be configured for sequential analog channel scanning. By default the ADC only samples and converts the inputs selected by MUX A. There is also a possibility to alternate between two sets of inputs selected by MUX A and MUX B during successive samples.

MUX Abstraction Model



When using MUX A to select analog inputs, the ADC module has the ability to scan multiple analog channels sequentially.

Input Selection:

The ADC module provides a flexible mechanism to select analog inputs for conversion:

- Fixed input selection
- Alternate input selection
- Channel scanning

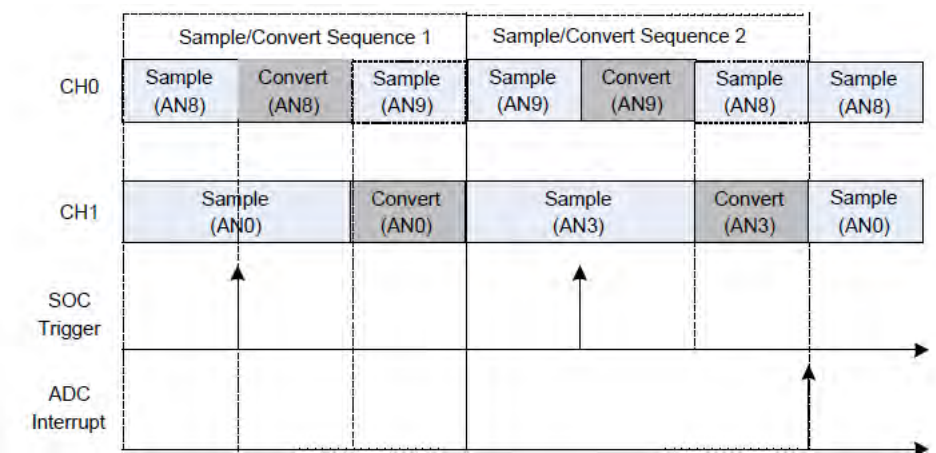
Fixed input selection:

This is achieved through one or more of the S&H channels available in the device. The S&H channels are connected to the analog input pins through the analog multiplexer.

Alternate Sampling:

In an Alternate Input Selection mode, the ADC completes one sweep using the MUX A selection, and then another sweep using the MUX B selection, and then another sweep using the MUX A selection, and so on.

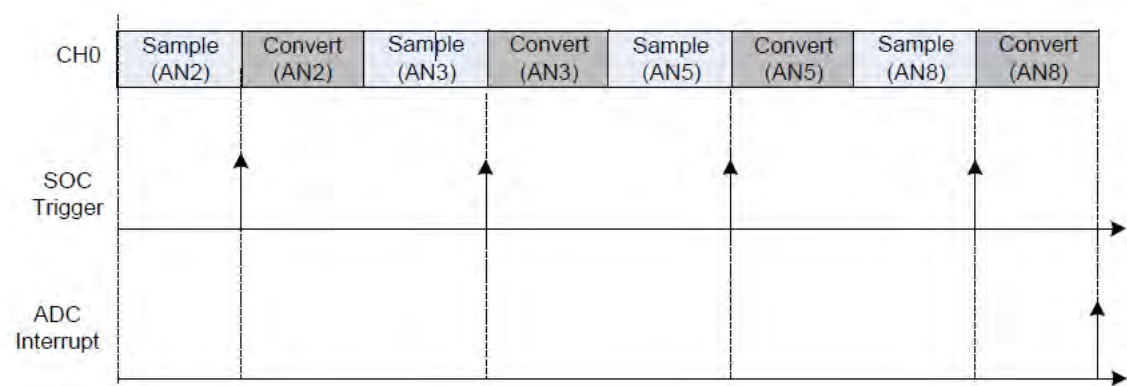
Alternate Input Selection in 2-Channel Sequential Sampling Configuration



Channel Scanning:

On some devices, the ADC module supports the Channel Scan mode using S&H Channel 0 (CH0). The number of inputs scanned is software selectable. Any subset of the analog inputs from AN0 to AN31 (depending on the number of analog inputs present on a specific device) can be selected for conversion. The selected inputs are converted in ascending order. For example, if the input selection includes AN4, AN1, and AN3, the conversion sequence is AN1, AN3, and AN4.

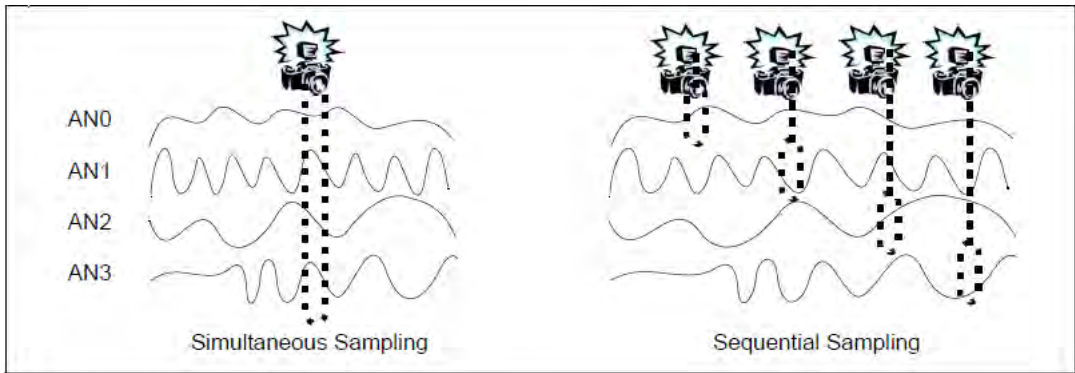
Scan Four Analog Inputs Using CH0



Multi-Channel Sample Conversion:

Multi-channel analog-to-digital converters typically convert each input channel sequentially using an input multiplexer. Simultaneously sampling multiple signals ensures that the snapshot of the analog inputs occurs at precisely the same time for all inputs. Certain applications require simultaneous sampling, especially when phase information exists between different channels. Sequential sampling takes a snapshot of each analog input just before conversion starts on that input. Certain ADC modules supports simultaneous sampling using two S&H or four S&H channels to sample the inputs at the same instance, and then perform the conversion for each channel sequentially.

Simultaneous and Sequential Sampling



Specifying Conversion Results Buffering for Devices with Direct Memory Access (DMA):

The ADC module contains a single-word, read-only, dual-port register, which stores the ADC conversion result. If more than one conversion result needs to be buffered before triggering an interrupt, DMA data transfers can be used. If DMA is supported and enabled, multiple conversion results can be automatically transferred from ADC internal buffer to a user-defined buffer in the DMA RAM area. Therefore, the application can process several conversion results with minimal software overhead.

5.6.1.4.2 Library Usage Model

Usage Model

The set of interface routines for the ADC Peripheral Library are:

Library Interface Section	Description

5.6.1.4.3 How the Library Works

How the Library Works

The following processes are involved while using an ADC module:

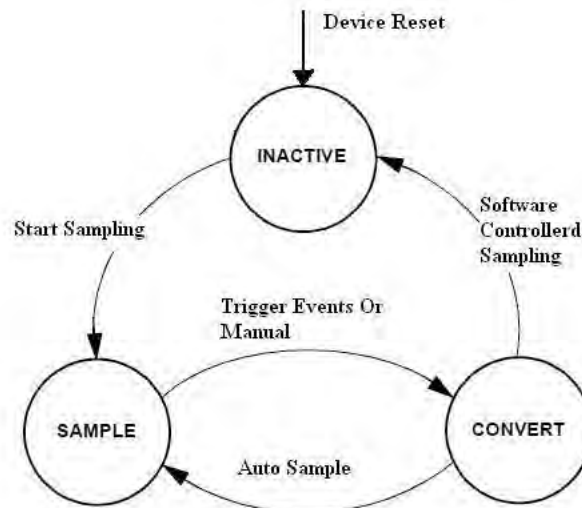
- [Initialization](#)
- [Controlling the sampling process](#)
- [Controlling the conversion process](#)
- [Accessing the results buffer](#)

5.6.1.4.3.1 General

The ADC conversion process can be thought of in terms of a finite state machine. The sample state represents the time that the input channel is connected to the S&H circuit and the signal is passed to the converter input. The convert state is transitory; the module enters this state as soon as it exits the sample state and transitions to a different state when that is done. The inactive state is the default state prior to module initialization and following a software-controlled conversion; it can be avoided in operation by using Auto-Sample mode.

Description

ADC Conversion Sequence or State Machine



5.6.1.4.3.2 Initialization

This topic provides information on the different processes needed to perform an analog-to-digital conversion.

Description

The following processes should be followed for performing an analog-to-digital conversion:

Initialize the ADC Module:

Number	Description	Functions associated
1	Selecting the voltage reference source Idle mode control	PLIB_ADC_VoltageReferenceSelect PLIB_ADC_StopInIdleEnable/ PLIB_ADC_StopInIdleDisable
2	Selecting the ADC conversion clock	PLIB_ADC_ConversionClockSet
3	Input channel selection Configuring MUX A and MUX B inputs, Alternating MUX A and MUX B input selections, Scanning through several inputs	Scan Mask Selection PLIB_ADC_InputScanMaskAdd PLIB_ADC_InputScanMaskRemove Positive Inputs PLIB_ADC_InputSelectPositive PLIB_ADC_MuxChannel0InputPositiveSelect PLIB_ADC_MuxChannel123InputPositiveSelect Negative Inputs PLIB_ADC_InputSelectNegative PLIB_ADC_MuxChannel0InputNegativeSelect PLIB_ADC_MuxChannel123InputNegativeSelect Scan Mode Selection PLIB_ADC_MuxAInputScanEnable PLIB_ADC_MuxAInputScanDisable
4	Enabling the ADC module	PLIB_ADC_Enable
5	Determine how many S&H channels will be used	PLIB_ADC_ChannelGroupSelect
6	Determine how sampling will occur	Sampling Control PLIB_ADC_SamplingModeEnable PLIB_ADC_SamplingModeDisable PLIB_ADC_SampleAcquisitionTimeSet
7	Selecting Manual or Auto-Sampling	PLIB_ADC_SampleAutoStartEnable PLIB_ADC_SampleAutoStartDisable PLIB_ADC_SamplingStart
8	Select conversion trigger and sampling time	PLIB_ADC_ConversionStart PLIB_ADC_ConversionClockSourceSelect PLIB_ADC_ConversionTriggerSourceSelect PLIB_ADC_ConversionStopSequenceEnable/ PLIB_ADC_ConversionStopSequenceDisable
9	Select how conversion results are stored in buffer	PLIB_ADC_ResultBufferModeSelect
10	Select the result format Result sign	PLIB_ADC_ResultFormatSelect PLIB_ADC_ResultSignGet
11	Select the number of readings per interrupt	PLIB_ADC_SamplesPerInterruptSelect
12	Select number of samples in DMA buffer for each ADC module and select how the DMA will access the ADC buffers	PLIB_ADC_DMAEnable/PLIB_ADC_DMADisable PLIB_ADC_DMABufferModeSelect PLIB_ADC_DMAAddressIncrementSelect PLIB_ADC_DMAInputBufferSelect

13	Select the 10-bit or 12-bit mode	PLIB_ADC_ResultSizeSelect
14	Channel pair configuration	PLIB_ADC_PairTriggerSourceSelect PLIB_ADC_PairConversionStart PLIB_ADC_PairInterruptRequestEnable PLIB_ADC_PairInterruptRequestDisable
15	Miscellaneous ADC functions:	
	Asynchronous sampling selection	PLIB_ADC_AsynchronousDedicatedSamplingEnable PLIB_ADC_AsynchronousDedicatedSamplingDisable
	Early interrupt control	PLIB_ADC_PairInterruptAfterFirstConversion PLIB_ADC_PairInterruptAfterSecondConversion
	Conversion order selection	PLIB_ADC_ConversionOrderSelect
	Global software trigger control	PLIB_ADC_GlobalSoftwareTriggerSet
	User ISR jump address	PLIB_ADC_IsrJumpTableBaseAddressSet

Reference Channels Initialization:

Internal reference channels like band gap voltage [ADC_REFERENCE_INPUT_VBG_DIV_BY_1], half of the band gap [ADC_REFERENCE_INPUT_VBG_DIV_BY_2] or one-sixth voltage band gap [ADC_REFERENCE_INPUT_VBG_DIV_BY_6] can be enabled or disabled through the APIs [PLIB_ADC_InternalReferenceChannelEnable](#) and [PLIB_ADC_InternalReferenceChannelDisable](#) respectively with appropriate parameters. The same APIs can be used to control the internal core voltage [ADC_REFERENCE_INPUT_VDDCORE] channels.

ADC Pairs Initialization:

Some ADC modules convert analog inputs in pairs. The analog input pair is a combination of an even and odd numbered analog input, such as AN0 and AN1, AN2, AN3, and so on. The technique of using pairs is particularly useful in power conversion applications that require voltage and current measurement for each PWM control loop.

- An interrupt request can be enabled/disabled using the API [PLIB_ADC_PairInterruptRequestEnable/PLIB_ADC_PairInterruptRequestDisable](#)
- A software trigger can be activated using the API [PLIB_ADC_PairConversionStart](#)
- Each ADC input pair can select an individual software trigger as a trigger source via the API [PLIB_ADC_PairTriggerSourceSelect](#)
- Each ADC input pair can select the global software trigger as a trigger source via the API [PLIB_ADC_GlobalSoftwareTriggerSet](#)

Each analog input uses a dedicated result register to store the converted result. For example, AN0 conversion results are always stored in the ADCBUF0 register and AN1 conversion results are always stored in the ADCBUF1 register.

Conversion Result Handling Using DMA:

DMA can be enabled or disabled using [PLIB_ADC_DMAEnable](#) and [PLIB_ADC_DMADisable](#), respectively. The DMA Buffer Build mode feature controlled through [PLIB_ADC_DMABufferModeSelect](#) determines how the conversion results are filled in the DMA RAM buffer area being used for the ADC. With the parameter ADC_DMA_BUFFER_MODE_ORDER_OF_CONV, DMA buffers are written in the order of conversion; otherwise, with ADC_DMA_BUFFER_MODE_SCATTER_GATHER, DMA buffers are written in Scatter/Gather mode. In the Scatter/Gather mode, the DMA channel must be configured for Peripheral Indirect Addressing. The DMA buffer is divided into consecutive memory blocks corresponding to all available analog inputs (out of AN0-AN31). Each conversion result for a particular analog input is automatically transferred by the ADC module to the corresponding block within the user-defined DMA buffer area. Successive samples for the same analog input are stored in sequence within the block assigned to that input.

The number of samples that need to be stored in the DMA buffer for each analog input is controlled by

[PLIB_ADC_DMAInputBufferSelect](#). The rate at which this internal pointer is incremented when data is written to the DMA buffer is controlled by [PLIB_ADC_DMAAddressIncrementSelect](#).

1. Configure the ADC interrupt (if required):
 - Clear the interrupt status flag
 - Select the ADC interrupt priority
 - Enable the ADC interrupt
2. Configure the DMA Channel.
3. Turn on the ADC module using API [PLIB_ADC_Enable](#).

Example Initialization:

```
// Include all channels in scan
PLIB_ADC_InputScanMaskAdd(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);

// Set the interrupt every 16 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_SAMPLES_PER_INTERRUPT_16);

// Enable scanning of channels
PLIB_ADC_ScanMuxAInputEnable(MY_ADC_INSTANCE);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);
```

Note: Not all functionality is available on all devices. Refer to the "Analog-to-Digital Converter (ADC)" chapter in the specific device data sheet for availability.

5.6.1.4.3.3 Controlling the Sampling Process

This topic describes the modes and related sampling functionality for the sampling process.

Description

The sampling process can be set up using the following two modes:

- **Manual Sampling:** Calling [PLIB_ADC_SamplingStart](#) causes the ADC to begin sampling. One of several options as discussed in "[Controlling the Conversion Process](#)" can be used to end sampling and complete conversions. Sampling does not resume until [PLIB_ADC_SamplingStart](#) is called again.
- **Automatic Sampling:** Setting the ADC in the Auto-Sampling mode using [PLIB_ADC_SampleAutoStartEnable](#) automatically begins sampling a channel whenever a conversion is not active on that channel. One of several options can be used to end sampling and complete conversions, as discussed in "[Controlling the Conversion Process](#)". If the simultaneous sampling is selected using [PLIB_ADC_SamplingModeEnable](#) with parameter `ADC_SAMPLING_MODE_SIMULTANEOUS`, sampling on a channel resumes after the conversion of all channels completes.

Other sampling related functionality:

- **Monitoring Sample Status:** [PLIB_ADC_SamplingIsActive](#) obtains the status as sampling or holding for the ADC module.
- **Aborting a Sample:** While in Manual Sampling mode, calling [PLIB_ADC_SamplingStop](#) will terminate sampling. If the conversion trigger source is selected as `ADC_CONVERSION_TRIGGER_SAMP_CLEAR` using [PLIB_ADC_ConversionTriggerSourceSelect](#), this causes a conversion to start automatically. While in Auto-Sampling mode, calling [PLIB_ADC_SampleAutoStartEnable](#) does not terminate an outgoing sample/convert sequence; however, sampling will not resume after a subsequent conversion.
- **Sampling Modes:** Different sampling modes can be changed using [PLIB_ADC_SamplingModeEnable](#) with the appropriate

parameter such as `ADC_SAMPLING_MODE_ALTERNATE_INPUT` for alternate input mode, `ADC_SAMPLING_MODE_SIMULTANEOUS` for simultaneous sampling mode, or `ADC_SAMPLING_MODE_SEQUENTIAL` for the sequential sampling mode. There is a possibility to combine the sampling modes say the Alternate input mode with either the simultaneous or the sequential sampling modes.

Note: Not all functionality is available on all devices. Refer to the "**Analog-to-Digital Converter (ADC)**" chapter in the specific device data sheet for availability.

5.6.1.4.3.4 Controlling the Conversion Process

The conversion trigger source will terminate sampling and start a selected sequence of conversions. It is also possible to obtain the value of the conversion clock, which is obtained by calling [PLIB_ADC_ConversionClockGet](#). '[PLIB_ADC_ConversionTriggerSourceSelect](#)' selects the source of the conversion trigger.

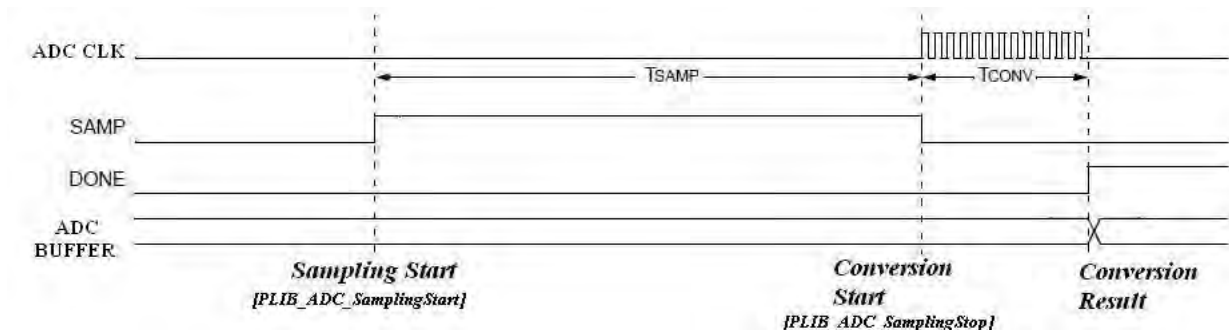
Description

Conversion can be started in one of the following three ways:

1. Manual Conversion Sequence:

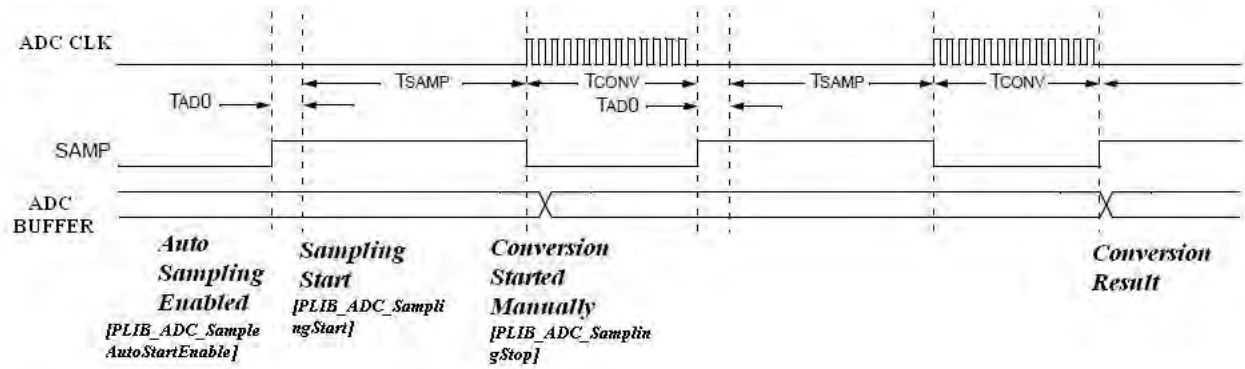
- **Manual Sample Start, Manual Conversion Start:** When `ADC_CONVERSION_TRIGGER_SAMP_CLEAR` is selected using [PLIB_ADC_ConversionTriggerSourceSelect](#), the conversion trigger is under software control. Calls to [PLIB_ADC_SamplingStop](#) will stop the sampling and start the conversion sequence. The user must call [PLIB_ADC_SamplingStart](#) and [PLIB_ADC_SamplingStop](#) in a timed manner, to ensure adequate sampling time.

Converting One Channel, Manual Sample Start, Manual Conversion Start



- **Automatic Sample Start, Manual Conversion Start:** Automatic sampling is initiated using [PLIB_ADC_SampleAutoStartEnable](#), calling [PLIB_ADC_SamplingStop](#) will terminate sampling and start conversion. After the conversion, the sampling starts again automatically. The user must call [PLIB_ADC_SamplingStop](#) in a timed manner, to ensure adequate sampling time. Wait for required acquisition/auto sample time (minimum of 1 TAD), and then call [PLIB_ADC_SamplingStop](#) to start the conversion process.

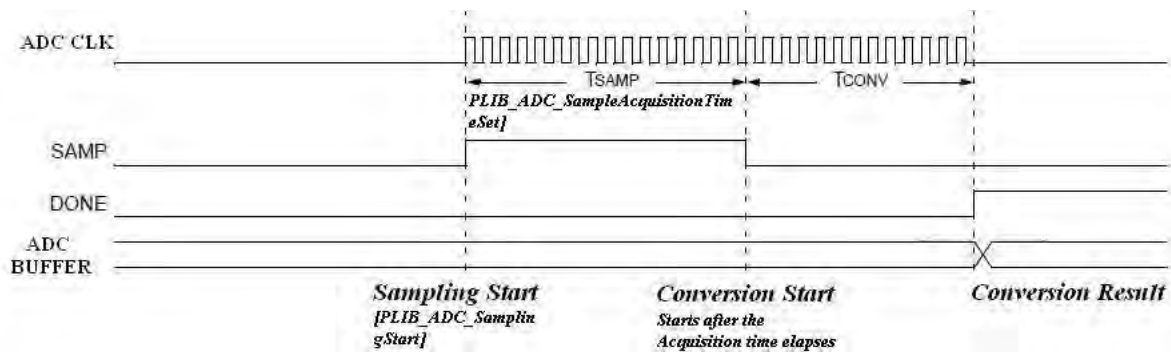
Converting One Channel, Automatic Sample Start, Manual Conversion Start



2. Clocked Conversion Sequence:

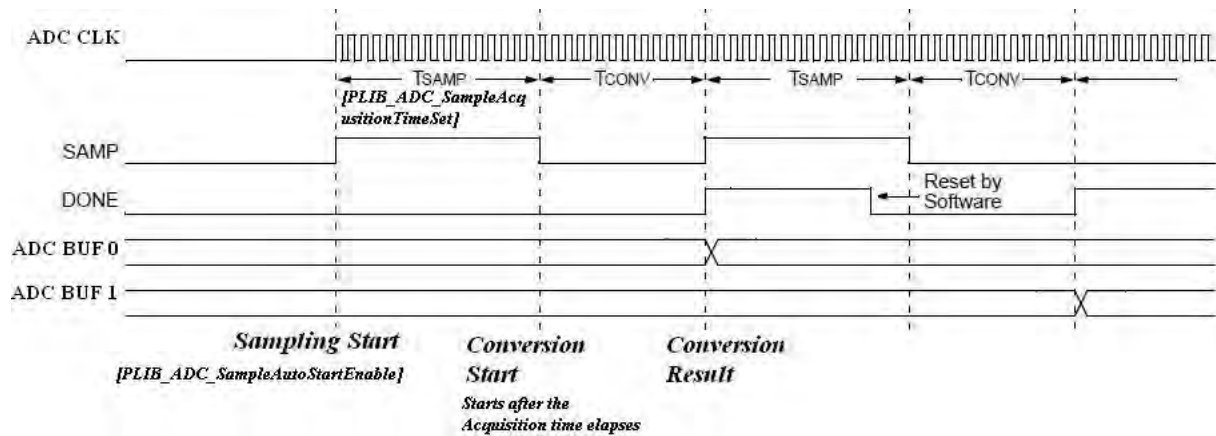
- Manual Sample Start and TAD based Conversion Start:** When `ADC_CONVERSION_TRIGGER_INTERNAL_COUNT` is selected using `PLIB_ADC_ConversionTriggerSourceSelect`, the conversion trigger is under analog-to-digital clock control. `PLIB_ADC_SampleAcquisitionTimeSet` selects the TAD clock cycles between the start of sampling and the start of conversion. [Minimum 1 clock cycle has to be selected to ensure the sampling requirements are met]. `PLIB_ADC_SamplingStart` starts the sampling for the configured acquisition time, and then `PLIB_ADC_SamplingStop` starts the conversion process.

Converting One Channel, Manual Sample Start, TAD-Based Conversion Start



- Auto Sample Start and TAD based Conversion Start or Free Running Sample Conversion:** With the selection of `ADC_CONVERSION_TRIGGER_INTERNAL_COUNT` using `PLIB_ADC_ConversionTriggerSourceSelect` and Automatic sampling initiation using `PLIB_ADC_SampleAutoStartEnable` allows the ADC module to schedule sample/conversion sequences with no intervention by the user or other device resources.

Converting One Channel, Auto-Sample Start, TAD-Based Conversion Start



3. **Event Trigger Conversion Start:** It might be necessary to synchronize the end of sampling and the start of conversion with some other time event. The ADC module may use one of following as conversion trigger sequences:

- **External Pin Trigger:** When `ADC_CONVERSION_TRIGGER_INT0_TRANSITION` is selected using [PLIB_ADC_ConversionTriggerSourceSelect](#)
- **General Purpose Timer Compare Match:** When `ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH` or `ADC_CONVERSION_TRIGGER_TMR5_COMPARE_MATCH` is selected using [PLIB_ADC_ConversionTriggerSourceSelect](#)

Both of the event trigger conversion modes previously described can be used in combination with auto-sampling ([PLIB_ADC_SampleAutoStartEnable](#)) to cause the ADC to synchronize the sample conversion events to the trigger pulse source.

Users should note that some devices have additional conversion trigger sources as part of the enumeration [ADC_CONVERSION_TRIGGER_SOURCE](#).

Other operations during Conversion process:

- **Monitoring Conversion process:** The status of the conversion can be obtained using [PLIB_ADC_ConversionHasCompleted](#)
- **Generating ADC Interrupt:** [PLIB_ADC_SamplesPerInterruptSelect](#) controls the generation of the ADC interrupt. To enable the interrupt it is also essential to enable the ADC interrupt.
- **Aborting the Conversion:** Calling [PLIB_ADC_Disable](#) will abort the current conversion. The result buffer is not updated with the partially completed ADC conversion sequence.
- ADC pair conversion status can be obtained from [PLIB_ADC_PairConversionIsPending](#)

Timing Details:

- **TAD** - The ADC module has a maximum rate at which conversions may be completed. An analog module clock, TAD, controls the conversion timing.
- **TSAMP** - The time required to sample and hold the sampled analog signal, configured through [PLIB_ADC_SampleAcquisitionTimeSet](#)
- **TCONV** - The time required to convert the sampled analog signal, configured through [PLIB_ADC_ConversionClockSet](#). The conversion clock can be verified using [PLIB_ADC_ConversionClockGet](#).

Note: Not all functionality is available on all devices. Refer to the "Analog-to-Digital Converter (ADC)" chapter in the specific device data sheet for availability.

5.6.1.4.3.5 Accessing the Result Buffers

The result buffers can be formatted to the desired format using the function [PLIB_ADC_ResultFormatSelect](#).

Description

As the analog-to-digital conversions are completed, the ADC module writes the results of the conversions into the ADC result buffer. This buffer is a RAM array of 16 words, accessed through the Special Function Register (SFR) space.

User software may attempt to read each ADC conversion result as it is generated; however, this might consume too much CPU time. Generally, to minimize software overhead, the ADC module will fill the buffer with results, and then generate an interrupt when the buffer is filled. There are two different modes for accessing the result buffers.

Note: Refer to the "**Analog-to-Digital Converter (ADC)**" chapter in the specific device data sheet to determine the correct mode for accessing the result buffer for your device.

- **Single Buffer mode:** Conversion results are automatically stored in a dedicated buffer. The module sets its interrupt flag after the conversion is complete, it also marks the conversion status as complete. After the interrupt, the conversion sequence can restart. For 8-bit results, the converted values are available through [PLIB_ADC_ResultGet](#).
- **Multi Buffer mode:** Conversion results are automatically stored in a dedicated position in the result buffer comprising an array of 16 words, allowing for multiple successive readings to be taken before software service is needed. Successive conversions are placed into sequential buffer positions. Alternatively, the buffer can be split into two arrays of 8 words for simultaneous conversion and read operations. The ADC module sets its interrupt flag after a selectable number of conversions, from 1 to 16, when the all buffer positions can be read. After the interrupt, the sequence restarts at the beginning of the buffer. When the interrupt flag is set, scan selections and the output buffer pointer return to their starting positions. The ADC result buffer is a set of 16 words, accessed through [PLIB_ADC_ResultGetByIndex](#), where buffer index can be any value ranging from 0 to 15.
 - [PLIB_ADC_SamplesPerInterruptSelect](#) selects how many analog-to-digital conversions will take place before the CPU is interrupted
 - The buffer fill mode can be selected using [PLIB_ADC_ResultBufferModeSelect](#) with the parameter of the type [ADC_BUFFER_MODE](#)
 - When the conversion result buffer is split ([ADC_BUFFER_MODE_TWO_8WORD_BUFFERS](#) used as the parameter for [PLIB_ADC_ResultBufferModeSelect](#)), [PLIB_ADC_ResultBufferIsFull](#) indicates which half of the buffer is being currently written by the ADC module
- **DMA Buffer mode:** The ADC result buffer is a single word register that stores the ADC conversion result. DMA data transfer can be used to buffer more than one conversion result, before triggering an interrupt. If no DMA is to be used, each result should be read by the user application using [PLIB_ADC_ResultGet](#) before it is overwritten by the next conversion. ADC DMA is enabled through [PLIB_ADC_DMAEnable](#).
 - The DMA buffer build mode selected using [PLIB_ADC_DMABufferModeSelect](#) with the parameter of type [ADC_DMA_BUFFER_MODE](#), which determines how the conversion results are filled in the DMA buffer area. If [ADC_DMA_BUFFER_MODE_SCATTER_GATHER](#) is selected, the ADC module needs to provide a scatter/gather address to the DMA channel based on the index of the analog input and size of the DMA buffer. If [ADC_DMA_BUFFER_MODE_ORDER_OF_CONV](#) is selected, the ADC module provides the address of the non-DMA stand-alone buffer.
 - [PLIB_ADC_DMAInputBufferSelect](#) selects the number of samples that need to be stored in the DMA buffer for each analog input
- **Miscellaneous:**
 - Output result sign (positive or negative) is available through [PLIB_ADC_ResultSignGet](#)
 - ADC pair sample availability can be checked using [PLIB_ADC_PairSamplesAvailable](#)

Note: Not all functionality is available on all devices. Refer to the "**Analog-to-Digital Converter (ADC)**" chapter in the specific device data sheet for availability.

5.6.1.4.3.6 Power-Saving Modes

This topic provides information on the power-saving modes available for use with the ADC module.

Description

Operation in Sleep mode: Operation in Sleep mode requires that the internal RC clock is selected using [PLIB_ADC_ConversionClockSourceSelect](#) with the parameter `ADC_CLOCK_SOURCE_INTERNAL_RC`. If the ADC interrupt is enabled, the device will wake up from Sleep mode on the ADC interrupt. On some microcontrollers, operation in Sleep mode requires that `ADC_CONVERSION_CLOCK_FRC` is selected using [PLIB_ADC_ConversionClockSourceSelect](#). If the ADC interrupt is enabled, the device will wake up from Sleep mode on the ADC interrupt.

Operation in Idle mode: [PLIB_ADC_StopInIdleEnable](#) and [PLIB_ADC_StopInIdleDisable](#) determine if the ADC module stops or continues operation in Idle mode. If [PLIB_ADC_StopInIdleDisable](#) is used, the module will continue operation in Idle mode. If the ADC interrupt is enabled, the device will wake up from Idle mode on the ADC interrupt. If [PLIB_ADC_StopInIdleEnable](#) is used, the module will stop in Idle mode. If the device enters Idle mode in the middle of conversion, the conversion is aborted.

Operation in other Power-Saving modes: If the ADC module is expected to operate in a power-saving mode, configure the acquisition time and the conversion clock using the functions in accordance with the power-saving mode clock that will be used. After the power-saving mode is entered, an analog-to-digital acquisition can be started. Once the acquisition is started, the device can continue to be clocked by the same power-saving source until the conversion has completed. If desired, the device may be placed in the power-saving Idle mode during conversion.

Note: Not all functionality is available on all devices. Refer to the "Analog-to-Digital Converter (ADC)" chapter in the specific device data sheet for availability.

5.6.1.4.3.7 Conversion Sequence Examples

This topic provides examples on how the sampling and conversion will occur in various configurations.

Description

Converting Single Channel, Manual Sample Start, Manual Conversion Start

*// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible values
// from ADC_MODULE_ID*

```
int16_t ADCValue;
```

// Enabling the sampling manually
`PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_SAMP_CLEAR);`

// Disabling ADC Input channels for Scan
`PLIB_ADC_ScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);`

// Connect AN2 as Positive Input
`PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN2);`

// Manual Sample and TAD = 2TCY
`PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 2);`
`PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);`

// Enable the ADC
`PLIB_ADC_Enable(MY_ADC_INSTANCE);`

```
while(1)
{
    // Start Sampling
    PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);

    Delay(); // Ensure, correct sampling time has elapsed before starting conversion.

    // Is Conversion Done ??
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));
}
```

```

    ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);
}

```

Converting Single Channel, Manual Sample Start, TAD-based Conversion Start

```

// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of
the possible values
// from ADC_MODULE_ID

int16_t ADCValue;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_ScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);
// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    // Start Sampling
    PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);

    // Is Conversion Done ??
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);
}

```

Converting Single Channel, Automatic Sample Start, TAD-based Conversion Start

```

// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of
the possible values
// from ADC_MODULE_ID

int16_t ADCValue;
uint8_t index;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_ScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);

// Set the interrupt every 3 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_3SAMPLES_PER_INTERRUPT);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{

```



```

    ADCValue = 0;

    // Auto Start Sampling and then go to conversion
    PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

    // Is Conversion Done ??
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    // Yes, stop sample/convert
    PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

    // Average the 2 ADC values
    for(index = 0; index < 2; index++)
    {
        ADCValue = ADCValue + PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, index);
    }

    ADCValue = ADCValue >> 1;
} // Repeat

```

Converting Single Channel, Automatic Sample Start, Conversion Trigger-based Conversion Start

*// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible values
 // from ADC_MODULE_ID*

```

int16_t ADCValue;

// General Purpose Timer 3 compare match ends sampling and starts conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE,
ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_ScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSelect(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSelect(MY_ADC_INSTANCE, 1);

// Configure the timer to generate period match as per the acquisition requirements
TMR3 = 0x0000; // set TMR3 to time out every 125 ms
PR3 = 0x3FFF;
T3CON = 0x8010;

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    // Auto Start Sampling and then go to conversion
    PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

    // Is Conversion Done ??
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    // Yes, stop sample/convert
    PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

    ADCValue += PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);
}

```

Sampling and Converting a Single Channel Multiple Times

// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible values

```

// from ADC_MODULE_ID

int16_t ADCValue;
uint8_t index;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_ScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);

// Set the interrupt every 16 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    ADCValue = 0;

    // Auto Start Sampling and then go to conversion
    PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

    // Is Conversion Done ??
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    // Yes, stop sample/convert
    PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

    // Average the 16 ADC value
    for(index = 0; index < 16; index++)
    {
        ADCValue = ADCValue + PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, index);
    }

    ADCValue = ADCValue >> 4;
} // Repeat

```

Sampling and Converting all Channels

```

// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of
// the possible values
// from ADC_MODULE_ID

int ADCValue, index;

// Include all channels in scan
PLIB_ADC_ScanMaskAdd(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);

// Set the interrupt every 16 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);

// Enable scanning of channels

```

```

PLIB_ADC_ScanMuxAInputEnable(MY_ADC_INSTANCE);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    ADCValue = 0;

    // Auto Start Sampling and then go to conversion
    PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

    // Is Conversion Done ??
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    // Yes, stop sample/convert
    PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

    // Average the 16 ADC value
    for(index = 0; index < 16; index++)
    {
        ADCValue = ADCValue + PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, index);
    }

    ADCValue = ADCValue >> 4;
} // Repeat

```

Wait for Sample, Manual Conversion Start

```

// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of
the possible values
// from ADC_MODULE_ID

uint8_t ADCValue;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_ScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);

// Connect AN12 as Positive Input
PLIB_ADC_InputSelectPositive(MY_ADC_INSTANCE, ADC_INPUT_POSITIVE_AN12);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    Delay(); // Ensure, correct sampling time has elapsed before starting conversion.

    // Start Sampling
    PLIB_ADC_SamplingStop(MY_ADC_INSTANCE);

    // Is Conversion Done ??
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);
}

```

Wait for Sample, Triggered Conversion Start

```

// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of
the possible values
// from ADC_MODULE_ID

uint8_t ADCValue;

// Connect AN2 as Positive Input
PLIB_ADC_InputSelectPositive(MY_ADC_INSTANCE, ADC_INPUT_POSITIVE_AN2);

// Manual Sample and Conversion Clock
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    Delay(); // Ensure, correct sampling time has elapsed

    PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE,
    ADC_CONVERSION_TRIGGER_INT0_TRANSITION);

    // Is Conversion Done ??
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    ADCValue = PLIB_ADC_ResultGet(MY_ADC_INSTANCE);
}

```

Manual Sample Start, TAD-based Conversion Start

```

// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of
the possible values
// from ADC_MODULE_ID

uint8_t ADCValue;

// Connect AN2 as Positive Input
PLIB_ADC_InputSelectPositive(MY_ADC_INSTANCE, ADC_INPUT_POSITIVE_AN2);

// Sample Time = 1TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 0);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    // Start Sampling
    PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);

    // Is Conversion Done ??
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    ADCValue = PLIB_ADC_ResultGet(MY_ADC_INSTANCE);
}

```

5.6.1.5 Configuring the Library

The library is configured for the supported ADC modules with the multi-buffer interface when the processor is chosen in the MPLAB IDE.

5.6.1.6 Library Interface

Channel Pairs Control

	Name	Description
≡	PLIB_ADC_PairConversionIsPending	Informs whether conversion is pending or complete.
≡	PLIB_ADC_PairConversionStart	Start conversion of the requested ADC pair.
≡	PLIB_ADC_PairInterruptAfterFirstConversion	ADC pair conversion interrupt is generated after the first conversion.
≡	PLIB_ADC_PairInterruptAfterSecondConversion	ADC pair conversion interrupt is generated after the second conversion.
≡	PLIB_ADC_PairInterruptRequestDisable	Disable IRQ generation for the ADC pair.
≡	PLIB_ADC_PairInterruptRequestEnable	Enable IRQ generation for the ADC pair.
≡	PLIB_ADC_PairSampleIsAvailable	Provides status of the ADC pair sample availability in the buffer.
≡	PLIB_ADC_PairSampleStatusClear	Clears the ADC pair ready status flag.
≡	PLIB_ADC_PairTriggerSourceSelect	Selects trigger source for the requested ADC pair.

Conversion Control Logic

	Name	Description
≡	PLIB_ADC_ConversionStart	Starts ADC module manual conversion process.
≡	PLIB_ADC_ConversionHasCompleted	Provides the conversion completion status of the ADC.
≡	PLIB_ADC_ConversionTriggerSourceSelect	Selects the conversion trigger source.
≡	PLIB_ADC_ConversionStopSequenceEnable	Stop conversion sequence (when the first ADC module interrupt is generated).
≡	PLIB_ADC_ConversionStopSequenceDisable	Normal conversion sequence.
≡	PLIB_ADC_ConversionClockSet	Sets the ADC module conversion clock.
≡	PLIB_ADC_ConversionClockGet	Obtains the conversion clock.
≡	PLIB_ADC_ConversionClockSourceSelect	Selects the ADC module conversion clock source.
≡	PLIB_ADC_ConversionTriggerGroupSelect	Selects the conversion trigger source group.
≡	PLIB_ADC_ConversionOrderSelect	Selects the conversion order.

Data Types & Constants

	Name	Description
≡	ADC_MODULE_ID	Identifies the supported ADC modules supported.
≡	ADC_VOLTAGE_REFERENCE	Data type defining the different ADC Voltage Reference by which the ADC can be configured.
≡	ADC_INPUTS_NEGATIVE	Data type defining the different ADC Negative Input Enumeration.
≡	ADC_SAMPLES_PER_INTERRUPT	Data type defining the Samples Per Interrupt Enumeration.
≡	ADC_CHANNEL_GROUP	Data type defines the ADC Channel Group.
≡	ADC_CLOCK_SOURCE	Data type defines the ADC Clock Source Select.
≡	ADC_CONVERSION_TRIGGER_SOURCE	Data type defines the ADC Conversion Trigger Source.
≡	ADC_RESULT_SIZE	Data type defines the ADC Result Size.
≡	ADC_BUFFER_MODE	Data type defines the ADC Buffer Mode.
≡	ADC_RESULT_BUF_STATUS	Data type defines the ADC Result Buffer Status
≡	ADC_MUX	Data type defining the different ADC MUX Enumeration.
≡	ADC_SAMPLING_MODE	Data type defines the ADC Sampling Mode Select.
≡	ADC_INPUTS_SCAN	Data type defines the ADC Scan inputs.

	ADC_REFERENCE_INPUT	Data type defines the ADC Reference Input Enumeration.
	ADC_RESULT_FORMAT	Data type defines the ADC Result Format.
	ADC_DMA_BUFFER_MODE	Data type defines the ADC Buffer Mode.
	ADC_PAIR	Data type defines the ADC Pair.
	ADC_CHANNEL123_INPUTS_NEG	Data type defines the ADC Channel 123 Negative Input Select.
	ADC_CHANNEL123_INPUTS_POS	Data type defines the ADC Channel 123 Positive Input Select.
	ADC_INPUTS_POSITIVE	Data type defines the ADC inputs.
	ADC_CONVERSION_ORDER	Data type defines the ADC Conversion Order.
	ADC_DMA_ADDRESS_INCREMENT	Data type defines the ADC DMA Increment Rate.
	ADC_DMA_INPUT_BUFFER	Data type defines the ADC DMA Buffer Per Input.
	ADC_ACQUISITION_TIME	Data type defining the different ADC acquisition times by which the ADC can be configured.
	ADC_CONVERSION_CLOCK	Data type defines the different ADC Conversion clock
	ADC_SAMPLE	Data type defining the size of the adc sample register.

DMA Transactions

	Name	Description
	PLIB_ADC_DMAAddressIncrementSelect	Selects the increment rate for the DMA address.
	PLIB_ADC_DMABufferModeSelect	DMA Buffer Build mode.
	PLIB_ADC_DMADisable	Disables the ADC Direct Memory Access (DMA).
	PLIB_ADC_DMAEnable	Enables the ADC Direct Memory Access (DMA).
	PLIB_ADC_DMAInputBufferSelect	Selects the number of DMA buffer locations per analog input.

Feature Existence

	Name	Description
	PLIB_ADC_ExistsAsynchronousDedicatedSampling	Identifies whether the AsynchronousDedicatedSampling feature exists on the ADC module
	PLIB_ADC_ExistsCalibrationControl	Identifies whether the CalibrationControl feature exists on the ADC module
	PLIB_ADC_ExistsChannelGroup	Identifies whether the ChannelGroup feature exists on the ADC module
	PLIB_ADC_ExistsConversionClock	Identifies whether the ConversionClock feature exists on the ADC module
	PLIB_ADC_ExistsConversionClockSource	Identifies whether the ConversionClockSource feature exists on the ADC module
	PLIB_ADC_ExistsConversionControl	Identifies whether the ConversionControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionOrder	Identifies whether the ConversionOrder feature exists on the ADC module
	PLIB_ADC_ExistsConversionStatus	Identifies whether the ConversionStatus feature exists on the ADC module
	PLIB_ADC_ExistsConversionStopSequenceControl	Identifies whether the ConversionStopSequenceControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionTriggerGroup	Identifies whether the ConversionTriggerGroup feature exists on the ADC module
	PLIB_ADC_ExistsConversionTriggerSource	Identifies whether the ConversionTriggerSource feature exists on the ADC module
	PLIB_ADC_ExistsDMAAddressIncrement	Identifies whether the DMAAddressIncrement feature exists on the ADC module

	<u>PLIB_ADC_ExistsDMABufferMode</u>	Identifies whether the DMABufferMode feature exists on the ADC module
	<u>PLIB_ADC_ExistsDMABuffersPerAnalogInput</u>	Identifies whether the DMABuffersPerInput feature exists on the ADC module
	<u>PLIB_ADC_ExistsDMAControl</u>	Identifies whether the DMAControl feature exists on the ADC module
	<u>PLIB_ADC_ExistsEnableControl</u>	Identifies whether the EnableControl feature exists on the ADC module.
	<u>PLIB_ADC_ExistsGlobalSoftwareTrigger</u>	Identifies whether the GlobalSoftwareTrigger feature exists on the ADC module
	<u>PLIB_ADC_ExistsInputSelect</u>	Identifies whether the InputSelect feature exists on the ADC module
	<u>PLIB_ADC_ExistsInternalReferenceChannelControl</u>	Identifies whether the InternalReferenceChannelControl feature exists on the ADC module
	<u>PLIB_ADC_ExistsISRJumpTableBaseAddress</u>	Identifies whether the ISRJumpTableBaseAddress feature exists on the ADC module
	<u>PLIB_ADC_ExistsMuxChannel0NegativeInput</u>	Identifies whether the MuxChannel0NegativeInput feature exists on the ADC module
	<u>PLIB_ADC_ExistsMuxChannel0PositiveInput</u>	Identifies whether the MuxChannel0PositiveInput feature exists on the ADC module
	<u>PLIB_ADC_ExistsMuxChannel123NegativeInput</u>	Identifies whether the MuxChannel123NegativeInput feature exists on the ADC module
	<u>PLIB_ADC_ExistsMuxChannel123PositiveInput</u>	Identifies whether the MuxChannel123PositiveInput feature exists on the ADC module
	<u>PLIB_ADC_ExistsMuxInputScanControl</u>	Identifies whether the MuxInputScanControl feature exists on the ADC module
	<u>PLIB_ADC_ExistsMuxInputScanSelect</u>	Identifies whether the MuxInputScanSelect feature exists on the ADC module
	<u>PLIB_ADC_ExistsPairConversionControl</u>	Identifies whether the PairConversionControl feature exists on the ADC module
	<u>PLIB_ADC_ExistsPairInterruptOnConversion</u>	Identifies whether the PairInterruptOnConversion feature exists on the ADC module
	<u>PLIB_ADC_ExistsPairInterruptRequest</u>	Identifies whether the PairInterruptRequest feature exists on the ADC module
	<u>PLIB_ADC_ExistsPairSampleStatus</u>	Identifies whether the PairSampleStatus feature exists on the ADC module
	<u>PLIB_ADC_ExistsPairTriggerSource</u>	Identifies whether the PairTriggerSource feature exists on the ADC module
	<u>PLIB_ADC_ExistsResultBufferFillStatus</u>	Identifies whether the ResultBufferFillStatus feature exists on the ADC module
	<u>PLIB_ADC_ExistsResultBufferMode</u>	Identifies whether the ResultBufferMode feature exists on the ADC module
	<u>PLIB_ADC_ExistsResultFormat</u>	Identifies whether the ResultFormat feature exists on the ADC module
	<u>PLIB_ADC_ExistsResultGet</u>	Identifies whether the ResultGet feature exists on the ADC module
	<u>PLIB_ADC_ExistsResultGetByIndex</u>	Identifies whether the ResultGetByIndex feature exists on the ADC module
	<u>PLIB_ADC_ExistsResultSign</u>	Identifies whether the ResultSign feature exists on the ADC module
	<u>PLIB_ADC_ExistsResultSize</u>	Identifies whether the ResultSize feature exists on the ADC module

	PLIB_ADC_ExistsSampleResolution	Identifies whether the SampleResolution feature exists on the ADC module
	PLIB_ADC_ExistsSamplesPerInterruptSelect	Identifies whether the SamplesPerInterruptSelect feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAcquisitionTime	Identifies whether the SamplingAcquisitionTime feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAutoStart	Identifies whether the SamplingAutoStart feature exists on the ADC module
	PLIB_ADC_ExistsSamplingControl	Identifies whether the SamplingControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingModeControl	Identifies whether the SamplingModeControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingStatus	Identifies whether the SamplingStatus feature exists on the ADC module
	PLIB_ADC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the ADC module
	PLIB_ADC_ExistsVoltageReference	Identifies whether the VoltageReference feature exists on the ADC module

General Configuration

	Name	Description
	PLIB_ADC_Enable	ADC module is enabled (turned ON).
	PLIB_ADC_Disable	ADC module is disabled (turned OFF).
	PLIB_ADC_StopInIdleDisable	Continue ADC module operation when the device is in Idle mode.
	PLIB_ADC_StopInIdleEnable	Discontinue ADC module operation when device enters Idle mode.
	PLIB_ADC_VoltageReferenceSelect	Voltage reference configuration.
	PLIB_ADC_CalibrationEnable	Calibration is performed on the next ADC conversion.
	PLIB_ADC_CalibrationDisable	Normal ADC module operation (no calibration is performed).
	PLIB_ADC_InputSelectPositive	Positive analog input channel selection.
	PLIB_ADC_InputSelectNegative	Analog negative channel selection.
	PLIB_ADC_InputScanMaskAdd	Select ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemove	Omits ADC analog channel for input scan.
	PLIB_ADC_InternalReferenceChannelEnable	Internal reference input is enabled.
	PLIB_ADC_InternalReferenceChannelDisable	Internal reference input is disabled.
	PLIB_ADC_ChannelGroupSelect	Selects channels to be utilized.

Miscellaneous

	Name	Description
	PLIB_ADC_AsynchronousDedicatedSamplingDisable	Disables asynchronous dedicated S&H sampling.
	PLIB_ADC_AsynchronousDedicatedSamplingEnable	Enables asynchronous dedicated S&H sampling.
	PLIB_ADC_GlobalSoftwareTriggerSet	Global software trigger enable/set control.
	PLIB_ADC_IsrJumpTableBaseAddressGet	Gets the base address of the user's ADC Interrupt Service Routine (ISR) jump table.
	PLIB_ADC_IsrJumpTableBaseAddressSet	Sets the base address of the user's ADC Interrupt Service Routine (ISR) jump table.

Mux Selection and Channel Scan

	Name	Description
	PLIB_ADC_MuxAInputScanDisable	Do not scan input selections for CH0+ of MUX A.

	PLIB_ADC_MuxAInputScanEnable	Scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxChannel0InputNegativeSelect	Channel 0 negative input select for multiplexer setting.
	PLIB_ADC_MuxChannel0InputPositiveSelect	Channel 0 positive input select for multiplexer setting.
	PLIB_ADC_MuxChannel123InputNegativeSelect	Channel 1, 2, and 3 negative input select.
	PLIB_ADC_MuxChannel123InputPositiveSelect	Channel 1, 2, and 3 positive input select.

Output Configuration

	Name	Description
	PLIB_ADC_ResultBufferModeSelect	Selects the result buffer mode.
	PLIB_ADC_ResultBufferStatusGet	Provides the buffer fill status.
	PLIB_ADC_ResultFormatSelect	Selects the result format.
	PLIB_ADC_ResultGet	Obtains the ADC sample/result value.
	PLIB_ADC_ResultGetByIndex	Provides the ADC conversion result based on the buffer index.
	PLIB_ADC_ResultSignGet	Obtains the ADC result sign.
	PLIB_ADC_ResultSizeSelect	Selects the ADC module mode/result size.

Sample and Hold Control Logic

	Name	Description
	PLIB_ADC_SampleAcquisitionTimeSet	Sets the ADC acquisition/auto-sample time in TADs.
	PLIB_ADC_SampleAutoStartDisable	Sampling auto-start is disabled.
	PLIB_ADC_SampleAutoStartEnable	Sampling auto-start is enabled.
	PLIB_ADC_SampleMaxGet	Provides the maximum sample value.
	PLIB_ADC_SampleMinGet	Provides the minimum sample value.
	PLIB_ADC_SamplesPerInterruptSelect	Interrupts at the completion of conversion for each nth sample.
	PLIB_ADC_SamplingIsActive	Provides the ADC sampling status.
	PLIB_ADC_SamplingModeSelect	Enable the selected sampling mode.
	PLIB_ADC_SamplingStart	Sampling is enabled.
	PLIB_ADC_SamplingStop	Holding is enabled.

Description

This section lists the interface routines, data types, constants and macros required for ADC library.

5.6.1.6.1 General Configuration

5.6.1.6.1.1 PLIB_ADC_Enable Function

C

```
void PLIB_ADC_Enable(
    ADC_MODULE_ID index
);
```

Description

This function enables (turns ON) the selected ADC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_Enable(MY_ADC_INSTANCE);
```

5.6.1.6.1.2 PLIB_ADC_Disable Function**C**

```
void PLIB_ADC_Disable(  
    ADC_MODULE_ID index  
)
```

Description

This function disables (turns OFF) the selected ADC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_Disable(MY_ADC_INSTANCE);
```

5.6.1.6.1.3 PLIB_ADC_StopInIdleDisable Function**C**

```
void PLIB_ADC_StopInIdleDisable(  
    ADC_MODULE_ID index  
)
```

Description

This function enables the ADC module to continue operation when the device is in Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsStopInIdleControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_StopInIdleDisable(MY_ADC_INSTANCE);
```

5.6.1.6.1.4 PLIB_ADC_StopInIdleEnable Function**C**

```
void PLIB_ADC_StopInIdleEnable(
    ADC_MODULE_ID index
);
```

Description

This function discontinues ADC module operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsStopInIdleControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_StopInIdleEnable(MY_ADC_INSTANCE);
```

5.6.1.6.1.5 PLIB_ADC_VoltageReferenceSelect Function**C**

```
void PLIB_ADC_VoltageReferenceSelect(
    ADC_MODULE_ID index,
    ADC_VOLTAGE_REFERENCE configValue
);
```

Description

This function configures the ADC module voltage reference.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
configValue	One of the possible values from ADC_VOLTAGE_REFERENCE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsVoltageReference](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_VoltageReferenceSelect(MY_ADC_INSTANCE, ADC_REFERENCE_VREFPLUS_TO_AVSS);
```

5.6.1.6.1.6 PLIB_ADC_CalibrationEnable Function**C**

```
void PLIB_ADC_CalibrationEnable(
    ADC_MODULE_ID index
);
```

Description

This function enables calibration to be performed on the next ADC conversion.

When the Calibration bit is enabled, inputs are disconnected and tied to AVss. This sets the inputs of the ADC to zero. Then, the user can perform a conversion. Use of the Calibration mode is not affected if the ADC line has been configured as analog or digital, nor by channel input selection. Any analog input switches are disconnected from the ADC module in this mode. The conversion result is stored by the user software and is used to compensate subsequent conversions. This can be done by adding the two's complement of the result obtained during calibration to all normal ADC conversions.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsCalibrationControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_CalibrationEnable(MY_ADC_INSTANCE);
```

5.6.1.6.1.7 PLIB_ADC_CalibrationDisable Function**C**

```
void PLIB_ADC_CalibrationDisable(  
    ADC_MODULE_ID index  
);
```

Description

This function enables normal ADC module operation without any calibration.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsCalibrationControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_CalibrationDisable(MY_ADC_INSTANCE);
```

5.6.1.6.1.8 PLIB_ADC_InputSelectPositive Function**C**

```
void PLIB_ADC_InputSelectPositive(  
    ADC_MODULE_ID index,  
    ADC_INPUTS_POSITIVE inputs  
);
```

Description

This function selects the positive analog input channel for the ADC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Valid for ADC modules without MUX A, MUX B, etc.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsInputSelect](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_InputSelectPositive(MY_ADC_INSTANCE, ADC_INPUT_POSITIVE_AN2);
```

5.6.1.6.1.9 PLIB_ADC_InputSelectNegative Function**C**

```
void PLIB_ADC_InputSelectNegative(  
    ADC_MODULE_ID index,  
    ADC_INPUTS_NEGATIVE negInput  
);
```

Description

This function selects a negative channel for the ADC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
negInput	One of the possible values from ADC_INPUTS_NEGATIVE

Returns

None.

Remarks

Valid for ADC modules without MUX A, MUX B, etc.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsInputSelect](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_InputSelectNegative(MY_ADC_INSTANCE, ADC_INPUT_NEGATIVE_VREF_MINUS);
```

5.6.1.6.1.10 PLIB_ADC_InputScanMaskAdd Function**C**

```
void PLIB_ADC_InputScanMaskAdd(  
    ADC_MODULE_ID index,  
    ADC_INPUTS_SCAN scanInputs  
);
```

Description

This function selects the ADC analog channel for input scanning.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN . Inputs are added for scanning.

Returns

None.

Remarks

Multiple channels can be added simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsMuxInputScanSelect](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// Single channel addition
PLIB_ADC_InputScanMaskAdd(MY_ADC_INSTANCE, ADC_INPUT_SCAN_AN2);
// Multiple channels addition
PLIB_ADC_InputScanMaskAdd(ADC_ID_1, ADC_INPUT_SCAN_AN2 | ADC_INPUT_SCAN_AN2);
```

5.6.1.6.1.11 PLIB_ADC_InputScanMaskRemove Function**C**

```
void PLIB_ADC_InputScanMaskRemove(
    ADC_MODULE_ID index,
    ADC_INPUTS_SCAN scanInputs
);
```

Description

This function allows the ADC analog channel to be omitted from input scanning.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN . Inputs are removed from scanning.

Returns

None.

Remarks

Multiple channels can be removed simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsMuxInputScanSelect](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
// Single channel addition  
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_AN2);  
// Multiple channels addition  
PLIB_ADC_InputScanMaskRemove(ADC_ID_1, ADC_INPUT_SCAN_AN2 | ADC_INPUT_SCAN_AN3);
```

5.6.1.6.1.12 PLIB_ADC_InternalReferenceChannelEnable Function**C**

```
void PLIB_ADC_InternalReferenceChannelEnable(  
    ADC_MODULE_ID index,  
    ADC_REFERENCE_INPUT input  
);
```

Description

This function enables the internal reference input.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Internal reference can be the band gap or core/regulator output voltage channels.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsInternalReferenceChannelControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_InternalReferenceChannelEnable(MY_ADC_INSTANCE,  
                                         ADC_REFERENCE_INPUT_VBG_DIV_BY_1);
```

5.6.1.6.1.13 PLIB_ADC_InternalReferenceChannelDisable Function**C**

```
void PLIB_ADC_InternalReferenceChannelDisable(  
    ADC_MODULE_ID index,  
    ADC_REFERENCE_INPUT input  
);
```

Description

This function disables the internal reference input.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Internal reference can be the band gap or core/regulator output voltage channels.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsInternalReferenceChannelControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_InternalReferenceChannelDisable(MY_ADC_INSTANCE,
                                         ADC_REFERENCE_INPUT_VBG_DIV_BY_1);
```

5.6.1.6.1.14 PLIB_ADC_ChannelGroupSelect Function**C**

```
void PLIB_ADC_ChannelGroupSelect(
    ADC_MODULE_ID index,
    ADC_CHANNEL_GROUP chlGroup
);
```

Description

This function selects the channels to be utilized by the ADC module.

Preconditions

Automatic sampling must be disabled.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
chlGroup	One of the possible values from ADC_CHANNEL_GROUP

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsChannelGroup](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ChannelGroupSelect(MY_ADC_INSTANCE, ADC_CHANNEL_GROUP_CH0_CH1);
```

5.6.1.6.2 Sample and Hold Control Logic

5.6.1.6.2.1 PLIB_ADC_SampleAcquisitionTimeSet Function

C

```
void PLIB_ADC_SampleAcquisitionTimeSet(  
    ADC_MODULE_ID index,  
    ADC_ACQUISITION_TIME acqTime  
);
```

Description

This function sets the ADC acquisition/auto-sample time in TADs.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
acqTime	Unsigned value of type ADC_ACQUISITION_TIME

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSamplingAcquisitionTime](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 2);
```

5.6.1.6.2.2 PLIB_ADC_SampleAutoStartDisable Function

C

```
void PLIB_ADC_SampleAutoStartDisable(  
    ADC_MODULE_ID index  
);
```

Description

This function disables auto-sampling and enables manual sampling.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSamplingAutoStart](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);
```

5.6.1.6.2.3 PLIB_ADC_SampleAutoStartEnable Function**C**

```
void PLIB_ADC_SampleAutoStartEnable(  
    ADC_MODULE_ID index  
);
```

Description

This function enables auto-sampling. Sampling begins immediately after the last conversion is completed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSamplingAutoStart](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);
```

5.6.1.6.2.4 PLIB_ADC_SampleMaxGet Function**C**

```
ADC_SAMPLE PLIB_ADC_SampleMaxGet(  
    ADC_MODULE_ID index  
);
```

Description

This function provides the maximum sample value based on the supported ADC resolution.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

[ADC_SAMPLE](#) - ADC Sample

Remarks

Possible maximum value that can be obtained from the converted value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSampleResolution](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
ADC_SAMPLE maxSample = PLIB_ADC_SampleMaxGet(MY_ADC_INSTANCE);
```

5.6.1.6.2.5 PLIB_ADC_SampleMinGet Function**C**

```
ADC_SAMPLE PLIB_ADC_SampleMinGet(
    ADC_MODULE_ID index
);
```

Description

This function provides the minimum sample value based on the supported ADC resolution.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

[ADC_SAMPLE](#) - ADC sample

Remarks

Possible minimum value that can be obtained from the converted value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSampleResolution](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
ADC_SAMPLE maxSample = PLIB_ADC_SampleMinGet(MY_ADC_INSTANCE);
```

5.6.1.6.2.6 PLIB_ADC_SamplesPerInterruptSelect Function**C**

```
void PLIB_ADC_SamplesPerInterruptSelect(
    ADC_MODULE_ID index,
    ADC_SAMPLES_PER_INTERRUPT value
);
```

Description

This function interrupts at the completion of conversion for each nth sample/convert sequence.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Possible values from ADC_SAMPLES_PER_INTERRUPT

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSamplesPerInterruptSelect](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);
```

5.6.1.6.2.7 PLIB_ADC_SamplingIsActive Function**C**

```
bool PLIB_ADC_SamplingIsActive(
    ADC_MODULE_ID index
);
```

Description

This function returns the ADC sampling status on whether the ADC is sampling or holding.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

bool - true = ADC S&H circuit is sampling

- false = ADC S&H circuit is holding

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSamplingStatus](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
bool my_status = PLIB_ADC_SamplingIsActive(MY_ADC_INSTANCE);
```

5.6.1.6.2.8 PLIB_ADC_SamplingModeSelect Function**C**

```
void PLIB_ADC_SamplingModeSelect(
    ADC_MODULE_ID index,
    ADC_SAMPLING_MODE mode
);
```

Description

This function selects the sampling mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the possible values from ADC_SAMPLING_MODE

Returns

None.

Remarks

Sampling mode could be alternate input or Simultaneous or Sequential mode. Alternate input can be combined with Simultaneous or Sequential modes.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSamplingModeControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SamplingModeSelect(MY_ADC_INSTANCE, ADC_SAMPLING_MODE_ALTERNATE_INPUT);
```

5.6.1.6.2.9 PLIB_ADC_SamplingStart Function**C**

```
void PLIB_ADC_SamplingStart(
    ADC_MODULE_ID index
);
```

Description

This function starts the ADC Sample and Hold (S&H) circuit to sample the input channel.

Preconditions

Automatic sampling must be disabled.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSamplingControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);
```

5.6.1.6.2.10 PLIB_ADC_SamplingStop Function

C

```
void PLIB_ADC_SamplingStop(  
    ADC_MODULE_ID index  
);
```

Description

This function stops the ADC S&H circuit from sampling and holds the sampled data.

Preconditions

Automatic sampling must be disabled.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsSamplingControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_SamplingStop(MY_ADC_INSTANCE);
```

5.6.1.6.3 Conversion Control Logic

5.6.1.6.3.1 PLIB_ADC_ConversionStart Function

C

```
void PLIB_ADC_ConversionStart(  
    ADC_MODULE_ID index  
);
```

Description

This function starts the ADC module manual conversion process.

Preconditions

Automatic sampling must be disabled.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_ADC_ExistsConversionControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionStart(MY_ADC_INSTANCE);
```

5.6.1.6.3.2 PLIB_ADC_ConversionHasCompleted Function

C

```
bool PLIB_ADC_ConversionHasCompleted(
    ADC_MODULE_ID index
);
```

Description

This function provides the completion status of analog-to-digital conversion.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

bool - true = ADC conversion is done/completed

- false = ADC conversion is in progress or has not started

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsConversionStatus](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
bool my_status = PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE);
```

5.6.1.6.3.3 PLIB_ADC_ConversionTriggerSourceSelect Function

C

```
void PLIB_ADC_ConversionTriggerSourceSelect(
    ADC_MODULE_ID index,
    ADC_CONVERSION_TRIGGER_SOURCE source
);
```

Description

This function selects the ADC module conversion trigger source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values from ADC_CONVERSION_TRIGGER_SOURCE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsConversionTriggerSource](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);
```

5.6.1.6.3.4 PLIB_ADC_ConversionStopSequenceEnable Function**C**

```
void PLIB_ADC_ConversionStopSequenceEnable(  
    ADC_MODULE_ID index  
);
```

Description

This function stops conversions when the first ADC module interrupt is generated. Hardware clears the Automatic Sampling bit when the ADC interrupt is generated.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsConversionStopSequenceControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_ConversionStopSequenceEnable(MY_ADC_INSTANCE);
```

5.6.1.6.3.5 PLIB_ADC_ConversionStopSequenceDisable Function**C**

```
void PLIB_ADC_ConversionStopSequenceDisable(  
    ADC_MODULE_ID index  
);
```

Description

This function enables normal operation, wherein the buffer contents will be overwritten by the next conversion sequence.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsConversionStopSequenceControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_ConversionStopSequencedisable(MY_ADC_INSTANCE);
```

5.6.1.6.3.6 PLIB_ADC_ConversionClockSet Function**C**

```
void PLIB_ADC_ConversionClockSet(  
    ADC_MODULE_ID index,  
    uint32_t baseFrequency,  
    ADC_CONVERSION_CLOCK value  
);
```

Description

This function sets the ADC module conversion clock prescaler.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Unsigned value of type ADC_CONVERSION_CLOCK

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsConversionClock](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 320000000);
```

5.6.1.6.3.7 PLIB_ADC_ConversionClockGet Function**C**

```
ADC_CONVERSION_CLOCK PLIB_ADC_ConversionClockGet(  
    ADC_MODULE_ID index,  
    uint32_t baseFrequency  
);
```

Description

This function obtains the conversion clock that is being used by the ADC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baseFrequencyHz	Input clock frequency to the ADC module in Hz

Returns

uint32_t - Conversion clock value

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsConversionClock](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// baseFrequency is the peripheral input frequency
ADC_CONVERSION_CLOCK conversionClock; // To store the conversion clock value
conversionClock = PLIB_ADC_ConversionClockGet(MY_ADC_INSTANCE, 8000000);
```

5.6.1.6.3.8 PLIB_ADC_ConversionClockSourceSelect Function**C**

```
void PLIB_ADC_ConversionClockSourceSelect(
    ADC_MODULE_ID index,
    ADC_CLOCK_SOURCE source
);
```

Description

This function selects the ADC module conversion clock source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values from ADC_CLOCK_SOURCE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsConversionClockSource](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionClockSourceSelect(MY_ADC_INSTANCE, ADC_CLOCK_SOURCE_SYSTEM_CLOCK);
```

5.6.1.6.3.9 PLIB_ADC_ConversionTriggerGroupSelect Function

C

```
void PLIB_ADC_ConversionTriggerGroupSelect(  
    ADC_MODULE_ID index,  
    ADC_CONVERSION_TRIGGER_GROUP group  
);
```

Description

This function selects the ADC module conversion trigger source group.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
group	One of the possible values from ADC_CONVERSION_TRIGGER_GROUP

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsConversionTriggerGroup](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_ConversionTriggerGroupSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_GROUP_SEL);
```

5.6.1.6.3.10 PLIB_ADC_ConversionOrderSelect Function

C

```
void PLIB_ADC_ConversionOrderSelect(  
    ADC_MODULE_ID index,  
    ADC_CONVERSION_ORDER order  
);
```

Description

This function selects the conversion order of the analog input.

Preconditions

The ADC module must be disabled.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
order	One of the possible values from ADC_CONVERSION_ORDER

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_ADC_ExistsConversionOrder](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionOrderSelect(MY_ADC_INSTANCE, ADC_CONVERSION_ORDER_ODD_FIRST);
```

5.6.1.6.4 DMA Transactions

5.6.1.6.4.1 PLIB_ADC_DMAAddressIncrementSelect Function

C

```
void PLIB_ADC_DMAAddressIncrementSelect(
    ADC_MODULE_ID index,
    ADC_DMA_ADDRESS_INCREMENT incRate
);
```

Description

This function selects the increment rate of the DMA address after completion of incRate number of sample/conversion operations. This gives information on how often the DMA RAM buffer pointer is incremented.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
incRate	One of the possible values from ADC_DMA_ADDRESS_INCREMENT

Returns

None.

Remarks

These represent the same set of bits as does the samples per interrupt selection.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsDMAAddressIncrement](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_DMAAddressIncrementSelect(MY_ADC_INSTANCE, ADC_DMA_ADDRESS_INCREMENT_EVERY_SAMPLE);
```

5.6.1.6.4.2 PLIB_ADC_DMABufferModeSelect Function

C

```
void PLIB_ADC_DMABufferModeSelect(
    ADC_MODULE_ID index,
    ADC_DMA_BUFFER_MODE bufferMode
);
```

Description

This function selects the DMA Buffer Build mode regardless of the DMA buffers being written in the order of conversion or in scatter/gather mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bufferMode	One of the possible values from ADC_DMA_BUFFER_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsDMABufferMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_DMABufferModeSelect(MY_ADC_INSTANCE, ADC_DMA_BUFFER_MODE_ORDER_OF_CONV);
```

5.6.1.6.4.3 PLIB_ADC_DMADisable Function**C**

```
void PLIB_ADC_DMADisable(
    ADC_MODULE_ID index
);
```

Description

This function disables the ADC Direct Memory Access (DMA).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Conversion results are stored in multiple buffers and the DMA will not be used.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsDMAControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_DMADisable(MY_ADC_INSTANCE);
```

5.6.1.6.4.4 PLIB_ADC_DMAEnable Function**C**

```
void PLIB_ADC_DMAEnable(
```

```
    ADC_MODULE_ID index  
);
```

Description

This function enables the ADC Direct Memory Access (DMA).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Conversion results are stored in a single register and are transferred to RAM using DMA.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsDMAControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_DMAEnable(MY_ADC_INSTANCE);
```

5.6.1.6.4.5 PLIB_ADC_DMAInputBufferSelect Function

C

```
void PLIB_ADC_DMAInputBufferSelect(  
    ADC_MODULE_ID index,  
    ADC_DMA_INPUT_BUFFER buffer  
);
```

Description

This function selects the number of DMA buffer locations per analog input and specifies the number of samples that need to be stored in the DMA buffer for each analog input.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
buffer	One of the possible values from ADC_DMA_INPUT_BUFFER

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsDMABuffersPerAnalogInput](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
```

```
// application developer.  
PLIB_ADC_DMAInputBufferSelect(MY_ADC_INSTANCE, ADC_DMA_INPUT_BUFFER_1WORD);
```

5.6.1.6.5 Channel Pairs Control

5.6.1.6.5.1 PLIB_ADC_PairConversionIsPending Function

C

```
bool PLIB_ADC_PairConversionIsPending(  
    ADC_MODULE_ID index,  
    ADC_PAIR pairNum  
);
```

Description

This function provides information on whether the conversion is pending or complete for the ADC pair.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pairNum	One of the possible values from ADC_PAIR

Returns

bool - true = Conversion of a pair of channels is pending. Set when selected trigger is asserted

- false = Conversion is complete

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsPairConversionControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
bool my_status = PLIB_ADC_PairConversionIsPending(MY_ADC_INSTANCE,  
                                                    ADC_PAIR_AN0_AN1);
```

5.6.1.6.5.2 PLIB_ADC_PairConversionStart Function

C

```
void PLIB_ADC_PairConversionStart(  
    ADC_MODULE_ID index,  
    ADC_PAIR pairNum  
);
```

Description

This function starts conversion of the requested ADC pair.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

pairNum	One of the possible values from ADC_PAIR

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsPairConversionControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_PairConversionStart(MY_ADC_INSTANCE, ADC_PAIR_AN0_AN1);
```

5.6.1.6.5.3 PLIB_ADC_PairInterruptAfterFirstConversion Function**C**

```
void PLIB_ADC_PairInterruptAfterFirstConversion(  
    ADC_MODULE_ID index  
);
```

Description

This function generates an interrupt after the first conversion is completed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsPairInterruptOnConversion](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_PairInterruptAfterFirstConversion(MY_ADC_INSTANCE);
```

5.6.1.6.5.4 PLIB_ADC_PairInterruptAfterSecondConversion Function**C**

```
void PLIB_ADC_PairInterruptAfterSecondConversion(  
    ADC_MODULE_ID index  
);
```

Description

This function generates an interrupt after the second conversion is completed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsPairInterruptOnConversion](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_PairInterruptAfterSecondConversion(MY_ADC_INSTANCE);
```

5.6.1.6.5.5 PLIB_ADC_PairInterruptRequestDisable Function**C**

```
void PLIB_ADC_PairInterruptRequestDisable(  
    ADC_MODULE_ID index,  
    ADC_PAIR pairNum  
);
```

Description

This function disables IRQ generation for the requested ADC pair.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pairNum	One of the possible values from ADC_PAIR

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsPairInterruptRequest](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_PairInterruptRequestDisable(MY_ADC_INSTANCE, ADC_PAIR_AN0_AN1);
```

5.6.1.6.5.6 PLIB_ADC_PairInterruptRequestEnable Function**C**

```
void PLIB_ADC_PairInterruptRequestEnable(  
    ADC_MODULE_ID index,  
    ADC_PAIR pairNum  
);
```

Description

This function enables IRQ generation, when requested conversion of channels associated with the pair is completed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pairNum	one of the possible values from ADC_PAIR

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsPairInterruptRequest](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_PairInterruptRequestEnable(MY_ADC_INSTANCE, ADC_PAIR_AN0_AN1);
```

5.6.1.6.5.7 PLIB_ADC_PairSampleIsAvailable Function**C**

```
bool PLIB_ADC_PairSampleIsAvailable(
    ADC_MODULE_ID index,
    ADC_PAIR pairNum
);
```

Description

This function provides information on whether the samples for the ADC pair selected are available in the buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pairNum	One of the possible values from ADC_PAIR

Returns

bool - true = Data is ready in buffer

- false = Data is not ready

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsPairSampleStatus](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
bool my_status;
```

```
my_status = PLIB_ADC_PairSampleIsAvailable(MY_ADC_INSTANCE, ADC_PAIR_AN0_AN1);
```

5.6.1.6.5.8 PLIB_ADC_PairSampleStatusClear Function

C

```
void PLIB_ADC_PairSampleStatusClear(  
    ADC_MODULE_ID index,  
    ADC_PAIR pairNum  
);
```

Description

This function clears a pair conversion ready status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pairNum	One of the possible values from ADC_PAIR

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsPairSampleStatus](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_PairSampleStatusClear(MY_ADC_INSTANCE, ADC_PAIR_AN0_AN1);
```

5.6.1.6.5.9 PLIB_ADC_PairTriggerSourceSelect Function

C

```
void PLIB_ADC_PairTriggerSourceSelect(  
    ADC_MODULE_ID index,  
    ADC_PAIR pairNum,  
    ADC_CONVERSION_TRIGGER_SOURCE trigSrc  
);
```

Description

This function selects the trigger source for the requested ADC pair.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pairNum	One of the possible values from ADC_PAIR

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsPairTriggerSource](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_PairTriggerSourceSelect(MY_ADC_INSTANCE, ADC_PAIR_AN0_AN1,
                                ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH);
```

5.6.1.6.6 Output Configuration

5.6.1.6.6.1 PLIB_ADC_ResultBufferModeSelect Function

C

```
void PLIB_ADC_ResultBufferModeSelect(
    ADC_MODULE_ID index,
    ADC_BUFFER_MODE mode
);
```

Description

This function selects the result buffer mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the possible values from ADC_BUFFER_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsResultBufferMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ResultBufferModeSelect(MY_ADC_INSTANCE,
                                ADC_BUFFER_MODE_TWO_8WORD_BUFFERS);
```

5.6.1.6.6.2 PLIB_ADC_ResultBufferStatusGet Function

C

```
ADC_RESULT_BUF_STATUS PLIB_ADC_ResultBufferStatusGet(
    ADC_MODULE_ID index
);
```

Description

This function obtains the buffer fill status.

Preconditions

ADC multi-buffer support is available and configured.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

bool - true = ADC is currently filling buffer 08-0F, user should access data in 00-07

- false = ADC is currently filling buffer 00-07, user should access data in 08-0F

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsResultBufferFillStatus](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
ADC_RESULT_BUF_STATUS my_status;

my_status = PLIB_ADC_ResultBufferStatusGet(MY_ADC_INSTANCE);
```

5.6.1.6.6.3 PLIB_ADC_ResultFormatSelect Function**C**

```
void PLIB_ADC_ResultFormatSelect(
    ADC_MODULE_ID index,
    ADC_RESULT_FORMAT resultFormat
);
```

Description

This function selects the result format.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
resultFormat	One of the possible values from ADC_RESULT_FORMAT

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsResultFormat](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ResultFormatSelect(MY_ADC_INSTANCE, ADC_RESULT_FORMAT_INTEGER_16BIT);
```

5.6.1.6.6.4 PLIB_ADC_ResultGet Function

C

```
ADC_SAMPLE PLIB_ADC_ResultGet(  
    ADC_MODULE_ID index  
);
```

Description

This function obtains the ADC sample/result value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

ADC_SAMPLE - Right-/Left-justified ADC result

Remarks

Only applicable for 10-bit or 12-bit ADC outputs.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsResultGet](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
ADC_SAMPLE my_res;  
my_res = PLIB_ADC_ResultGet(MY_ADC_INSTANCE);
```

5.6.1.6.6.5 PLIB_ADC_ResultGetByIndex Function

C

```
ADC_SAMPLE PLIB_ADC_ResultGetByIndex(  
    ADC_MODULE_ID index,  
    uint8_t bufferIndex  
);
```

Description

This function provides the ADC module conversion result based on the buffer index.

Preconditions

ADC multi-buffer support available and configured.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bufferIndex	Value ranging from 0 to 15

Returns

int16_t - ADC Conversion result at the respective bufferIndex

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsResultGetByIndex](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
ADC_SAMPLE my_res = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 15);
```

5.6.1.6.6 PLIB_ADC_ResultSignGet Function**C**

```
bool PLIB_ADC_ResultSignGet(  
    ADC_MODULE_ID index  
)
```

Description

This function obtains the ADC result sign status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- bool - true = ADC result is negative
- false = ADC result is positive

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsResultSign](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
bool my_sign = PLIB_ADC_ResultSignGet(MY_ADC_INSTANCE);
```

5.6.1.6.6.7 PLIB_ADC_ResultSizeSelect Function**C**

```
void PLIB_ADC_ResultSizeSelect(  
    ADC_MODULE_ID index,  
    ADC_RESULT_SIZE resSize  
)
```

Description

This function selects the ADC module mode/result size.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
resSize	One of the possible values from ADC_RESULT_SIZE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsResultSize](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_ResultSizeSelect(MY_ADC_INSTANCE, ADC_RESULT_SIZE_10_BITS);
```

5.6.1.6.7 Miscellaneous

5.6.1.6.7.1 PLIB_ADC_AsynchronousDedicatedSamplingDisable Function

C

```
void PLIB_ADC_AsynchronousDedicatedSamplingDisable(  
    ADC_MODULE_ID index  
)
```

Description

This function disables constant sampling, and starts sampling when the trigger event is detected and completes the sampling process in two ADC clock cycles.

Preconditions

The ADC module must be disabled.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsAsynchronousDedicatedSampling](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_AsynchronousDedicatedSamplingDisable(MY_ADC_INSTANCE);
```

5.6.1.6.7.2 PLIB_ADC_AsynchronousDedicatedSamplingEnable Function

C

```
void PLIB_ADC_AsynchronousDedicatedSamplingEnable(  
    ADC_MODULE_ID index  
)
```

```
    ADC_MODULE_ID index
);
```

Description

This function enables constant sampling of the dedicated S&H, and then terminates sampling as soon as the trigger pulse is detected.

Preconditions

The ADC module must be disabled.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsAsynchronousDedicatedSampling](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_AsynchronousDedicatedSamplingEnable(MY_ADC_INSTANCE);
```

5.6.1.6.7.3 PLIB_ADC_GlobalSoftwareTriggerSet Function

C

```
void PLIB_ADC_GlobalSoftwareTriggerSet(
    ADC_MODULE_ID index
);
```

Description

This function enables/sets the global software trigger. It will trigger conversions based on the trigger sources selected.

Preconditions

Global software trigger has to be cleared before setting back.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature/bit is automatically cleared by hardware.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsGlobalSoftwareTrigger](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_GlobalSoftwareTriggerSet(MY_ADC_INSTANCE);
```

5.6.1.6.7.4 PLIB_ADC_IsrJumpTableBaseAddressGet Function

C

```
int PLIB_ADC_IsrJumpTableBaseAddressGet(  
    ADC_MODULE_ID index  
)  
;
```

Description

This function gets the base address of the user's ADC Interrupt Service Routine (ISR) jump table.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

baseAddr - Upper 15 bits of the user's ADC ISR jump table address (bit 0 is a don't care)

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsISRJumpTableBaseAddress](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
unsigned int add;  
add = PLIB_ADC_IsrJumpTableBaseAddressGet(MY_ADC_INSTANCE);
```

5.6.1.6.7.5 PLIB_ADC_IsrJumpTableBaseAddressSet Function

C

```
void PLIB_ADC_IsrJumpTableBaseAddressSet(  
    ADC_MODULE_ID index,  
    int baseAddr  
)  
;
```

Description

This function sets the base address of the user's ADC Interrupt Service Routine (ISR) jump table.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baseAddr	Upper 15 bits of the user's ADC ISR jump table address (bit 0 is a don't care)

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsISRJumpTableBaseAddress](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_IsrJumpTableBaseAddressSet(MY_ADC_INSTANCE, 0x1000);
```

5.6.1.6.8 Data Types & Constants**5.6.1.6.8.1 ADC_MODULE_ID Enumeration****C**

```
typedef enum {
    ADC_ID_1,
    ADC_ID_2,
    ADC_ID_3,
    ADC_ID_4,
    ADC_ID_5,
    ADC_NUMBER_OF_MODULES
} ADC_MODULE_ID;
```

Description

ADC Module ID

This enumeration identifies the available ADC modules.

Members

Members	Description
ADC_ID_1	ADC Module 1 ID
ADC_ID_2	ADC Module 2 ID
ADC_ID_3	ADC Module 3 ID
ADC_ID_4	ADC Module 4 ID
ADC_ID_5	ADC Module 5 ID
ADC_NUMBER_OF_MODULES	Number of available ADC modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules are available on all devices. Refer to the specific data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

5.6.1.6.8.2 ADC_VOLTAGE_REFERENCE Enumeration**C**

```
typedef enum {
    ADC_VREF_POS_TO_VDD,
    ADC_VREF_POS_TO_VREF_PLUS,
    ADC_VREF_NEG_TO_VSS,
    ADC_VREF_NEG_TO_VREF_MINUS,
    ADC_REFERENCE_VDD_TO_AVSS,
    ADC_REFERENCE_VREFPLUS_TO_AVSS,
    ADC_REFERENCE_AVDD_TO_VREF_NEG,
    ADC_REFERENCE_VREFPLUS_TO_VREFNEG
} ADC_VOLTAGE_REFERENCE;
```

Description

ADC Voltage Reference Enumeration

This data type defines the different ADC Voltage Reference by which the ADC can be configured.

Members

Members	Description
ADC_VREF_POS_TO_VDD	Positive voltage reference supplied internally by VDD
ADC_VREF_POS_TO_VREF_PLUS	Positive voltage reference supplied externally through VREF+ pin
ADC_VREF_NEG_TO_VSS	Negative voltage reference supplied internally through VSS
ADC_VREF_NEG_TO_VREF_MINUS	Negative voltage reference supplied externally through VREF- pin
ADC_REFERENCE_VDD_TO_AVSS	Positive voltage reference supplied internally by VDD and Negative voltage reference supplied internally through VSS
ADC_REFERENCE_VREFPLUS_TO_AVSS	Positive voltage reference supplied externally through VREF+ pin and Negative voltage reference supplied internally through VSS
ADC_REFERENCE_AVDD_TO_VREF_NEG	Positive voltage reference supplied internally by VDD and Negative voltage reference supplied externally through VREF- pin
ADC_REFERENCE_VREFPLUS_TO_VREFNEG	Positive voltage reference supplied externally through VREF+ pin and Negative voltage reference supplied externally through VREF- pin

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.3 ADC_INPUTS_NEGATIVE Enumeration**C**

```
typedef enum {
    ADC_INPUT_NEGATIVE_VREF_MINUS,
    ADC_INPUT_NEGATIVE_AN1,
    ADC_INPUT_NEGATIVE_AN2,
    ADC_INPUT_NEGATIVE_AN3,
    ADC_INPUT_NEGATIVE_AN4,
    ADC_INPUT_NEGATIVE_AN5,
    ADC_INPUT_NEGATIVE_AN6
} ADC_INPUTS_NEGATIVE;
```

Description

ADC Negative Input Enumeration

This data type defines the different ADC Negative Input Enumeration.

Members

Members	Description
ADC_INPUT_NEGATIVE_VREF_MINUS	Negative input is Vref
ADC_INPUT_NEGATIVE_AN1	Negative input is AN1
ADC_INPUT_NEGATIVE_AN2	Negative input is AN2
ADC_INPUT_NEGATIVE_AN3	Negative input is AN3
ADC_INPUT_NEGATIVE_AN4	Negative input is AN4
ADC_INPUT_NEGATIVE_AN5	Negative input is AN5
ADC_INPUT_NEGATIVE_AN6	Negative input is AN6

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.4 ADC_SAMPLES_PER_INTERRUPT Enumeration

C

```
typedef enum {
    ADC_1SAMPLE_PER_INTERRUPT,
    ADC_2SAMPLES_PER_INTERRUPT,
    ADC_3SAMPLES_PER_INTERRUPT,
    ADC_4SAMPLES_PER_INTERRUPT,
    ADC_5SAMPLES_PER_INTERRUPT,
    ADC_6SAMPLES_PER_INTERRUPT,
    ADC_7SAMPLES_PER_INTERRUPT,
    ADC_8SAMPLES_PER_INTERRUPT,
    ADC_9SAMPLES_PER_INTERRUPT,
    ADC_10SAMPLES_PER_INTERRUPT,
    ADC_11SAMPLES_PER_INTERRUPT,
    ADC_12SAMPLES_PER_INTERRUPT,
    ADC_13SAMPLES_PER_INTERRUPT,
    ADC_14SAMPLES_PER_INTERRUPT,
    ADC_15SAMPLES_PER_INTERRUPT,
    ADC_16SAMPLES_PER_INTERRUPT,
    ADC_17SAMPLES_PER_INTERRUPT,
    ADC_18SAMPLES_PER_INTERRUPT,
    ADC_19SAMPLES_PER_INTERRUPT,
    ADC_20SAMPLES_PER_INTERRUPT,
    ADC_21SAMPLES_PER_INTERRUPT,
    ADC_22SAMPLES_PER_INTERRUPT,
    ADC_23SAMPLES_PER_INTERRUPT,
    ADC_24SAMPLES_PER_INTERRUPT,
    ADC_25SAMPLES_PER_INTERRUPT,
    ADC_26SAMPLES_PER_INTERRUPT,
    ADC_27SAMPLES_PER_INTERRUPT,
    ADC_28SAMPLES_PER_INTERRUPT,
    ADC_29SAMPLES_PER_INTERRUPT,
    ADC_30SAMPLES_PER_INTERRUPT,
    ADC_31SAMPLES_PER_INTERRUPT,
    ADC_32SAMPLES_PER_INTERRUPT
} ADC_SAMPLES_PER_INTERRUPT;
```

Description

ADC samples per Interrupt Enumeration

This data type defines the Samples Per Interrupt Enumeration.

Members

Members	Description
ADC_1SAMPLE_PER_INTERRUPT	Interrupts at the completion of conversion for each sample/convert sequence
ADC_2SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 2nd sample/convert sequence
ADC_3SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 3rd sample/convert sequence
ADC_4SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 4th sample/convert sequence
ADC_5SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 5th sample/convert sequence
ADC_6SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 6th sample/convert sequence
ADC_7SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 7th sample/convert sequence
ADC_8SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 8th sample/convert sequence
ADC_9SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 9th sample/convert sequence
ADC_10SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 10th sample/convert sequence
ADC_11SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 11th sample/convert sequence
ADC_12SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 12th sample/convert sequence

ADC_13SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 13th sample/convert sequence
ADC_14SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 14th sample/convert sequence
ADC_15SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 15th sample/convert sequence
ADC_16SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 16th sample/convert sequence
ADC_17SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 17th sample/convert sequence
ADC_18SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 18th sample/convert sequence
ADC_19SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 19th sample/convert sequence
ADC_20SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 20th sample/convert sequence
ADC_21SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 21st sample/convert sequence
ADC_22SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 22nd sample/convert sequence
ADC_23SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 23rd sample/convert sequence
ADC_24SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 24th sample/convert sequence
ADC_25SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 25th sample/convert sequence
ADC_26SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 26th sample/convert sequence
ADC_27SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 27th sample/convert sequence
ADC_28SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 28th sample/convert sequence
ADC_29SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 29th sample/convert sequence
ADC_30SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 30th sample/convert sequence
ADC_31SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 31st sample/convert sequence
ADC_32SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 32nd sample/convert sequence

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.5 ADC_CHANNEL_GROUP Enumeration**C**

```
typedef enum {
    ADC_CHANNEL_GROUP_CH0_CH1_CH2_CH3,
    ADC_CHANNEL_GROUP_CH0_CH1,
    ADC_CHANNEL_GROUP_CH0
} ADC_CHANNEL_GROUP;
```

Description

ADC Channel Group

This data type defines the ADC Channel Group.

Members

Members	Description
ADC_CHANNEL_GROUP_CH0_CH1_CH2_CH3	Converts Channels CH0, CH1, CH2 and CH3
ADC_CHANNEL_GROUP_CH0_CH1	Converts Channels CH0, CH1
ADC_CHANNEL_GROUP_CH0	Converts Channel CH0

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.6 ADC_CLOCK_SOURCE Enumeration

C

```
typedef enum {
    ADC_CLOCK_SOURCE_SYSTEM_CLOCK,
    ADC_CLOCK_SOURCE_INTERNAL_RC,
    ADC_CLOCK_SOURCE_AUXILIARY_PLL,
    ADC_CLOCK_SOURCE_PRIMARY_PLL
} ADC_CLOCK_SOURCE;
```

Description

ADC Clock Source Select

This data type defines the ADC Clock Source Select.

Members

Members	Description
ADC_CLOCK_SOURCE_SYSTEM_CLOCK	A/D system clock
ADC_CLOCK_SOURCE_INTERNAL_RC	A/D internal RC clock
ADC_CLOCK_SOURCE_AUXILIARY_PLL	ADC is clocked by the auxiliary PLL (ACLK)
ADC_CLOCK_SOURCE_PRIMARY_PLL	ADC is clock by the primary PLL (FVCO)

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.7 ADC_CONVERSION_TRIGGER_SOURCE Enumeration

C

```
typedef enum {
    ADC_CONVERSION_TRIGGER_SAMP_CLEAR,
    ADC_CONVERSION_TRIGGER_INT0_TRANSITION,
    ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH,
    ADC_CONVERSION_TRIGGER_TMR5_COMPARE_MATCH,
    ADC_CONVERSION_TRIGGER_CTMU_EVENT,
    ADC_CONVERSION_TRIGGER_CTMU_EVENT_ALTERNATIVE,
    ADC_CONVERSION_TRIGGER_INTERNAL_COUNT,
    ADC_CONVERSION_TRIGGER_NO_CONVERSION,
    ADC_CONVERSION_TRIGGER_INDIVIDUAL_SFW_TRIG,
    ADC_CONVERSION_TRIGGER_GLOBAL_SFW_TRIG,
    ADC_CONVERSION_TRIGGER_PWM_PRIMARY_SPECIAL_EVENT,
    ADC_CONVERSION_TRIGGER_PWM_GEN1_PRIMARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN2_PRIMARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN3_PRIMARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN4_PRIMARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN5_PRIMARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN6_PRIMARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN7_PRIMARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN8_PRIMARY,
    ADC_CONVERSION_TRIGGER_TMR1_PERIOD_MATCH,
    ADC_CONVERSION_TRIGGER_PWM_SECONDARY_SPECIAL_EVENT,
    ADC_CONVERSION_TRIGGER_PWM_GEN1_SECONDARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN2_SECONDARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN3_SECONDARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN4_SECONDARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN5_SECONDARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN6_SECONDARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN7_SECONDARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN8_SECONDARY,
    ADC_CONVERSION_TRIGGER_PWM_GEN9_SECONDARY,
}
```



```

ADC_CONVERSION_TRIGGER_PWM_GEN1_CURRENT_LMT_ADC,
ADC_CONVERSION_TRIGGER_PWM_GEN2_CURRENT_LMT_ADC,
ADC_CONVERSION_TRIGGER_PWM_GEN3_CURRENT_LMT_ADC,
ADC_CONVERSION_TRIGGER_PWM_GEN4_CURRENT_LMT_ADC,
ADC_CONVERSION_TRIGGER_PWM_GEN5_CURRENT_LMT_ADC,
ADC_CONVERSION_TRIGGER_PWM_GEN6_CURRENT_LMT_ADC,
ADC_CONVERSION_TRIGGER_PWM_GEN7_CURRENT_LMT_ADC,
ADC_CONVERSION_TRIGGER_PWM_GEN8_CURRENT_LMT_ADC,
ADC_CONVERSION_TRIGGER_TMR2_PERIOD_MATCH
} ADC_CONVERSION_TRIGGER_SOURCE;

```

Description

ADC Conversion Trigger Source

This data type defines the ADC Conversion Trigger Source.

Members

Members	Description
ADC_CONVERSION_TRIGGER_SAMP_CLEAR	Clearing SAMP bit (full program control)
ADC_CONVERSION_TRIGGER_INT0_TRANSITION	Active transition on INT0 pin (basic sync convert)
ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH	Timer3 compare match
ADC_CONVERSION_TRIGGER_TMR5_COMPARE_MATCH	Timer5 compare match
ADC_CONVERSION_TRIGGER_CTMU_EVENT	CTMU event
ADC_CONVERSION_TRIGGER_CTMU_EVENT_ALTERNATIVE	CTMU event Alternative
ADC_CONVERSION_TRIGGER_INTERNAL_COUNT	Internal counter (auto-convert)
ADC_CONVERSION_TRIGGER_NO_CONVERSION	No conversion enabled
ADC_CONVERSION_TRIGGER_INDIVIDUAL_SFW_TRIG	Individual software trigger selected
ADC_CONVERSION_TRIGGER_GLOBAL_SFW_TRIG	Global software trigger selected
ADC_CONVERSION_TRIGGER_PWM_PRIMARY_SPECIAL_EVENT	PWM Special/Primary Special Event Trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN1_PRIMARY	PWM Generator 1 primary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN2_PRIMARY	PWM Generator 2 primary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN3_PRIMARY	PWM Generator 3 primary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN4_PRIMARY	PWM Generator 4 primary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN5_PRIMARY	PWM Generator 5 primary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN6_PRIMARY	PWM Generator 6 primary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN7_PRIMARY	PWM Generator 7 primary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN8_PRIMARY	PWM Generator 8 primary trigger selected
ADC_CONVERSION_TRIGGER_TMR1_PERIOD_MATCH	Timer1 period match
ADC_CONVERSION_TRIGGER_PWM_SECONDARY_SPECIAL_EVENT	PWM secondary special event trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN1_SECONDARY	PWM Generator 1 secondary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN2_SECONDARY	PWM Generator 2 secondary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN3_SECONDARY	PWM Generator 3 secondary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN4_SECONDARY	PWM Generator 4 secondary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN5_SECONDARY	PWM Generator 5 secondary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN6_SECONDARY	PWM Generator 6 secondary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN7_SECONDARY	PWM Generator 7 secondary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN8_SECONDARY	PWM Generator 8 secondary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN9_SECONDARY	PWM Generator 9 secondary trigger selected
ADC_CONVERSION_TRIGGER_PWM_GEN1_CURRENT_LMT_ADC	PWM Generator 1 current-limit ADC trigger

ADC_CONVERSION_TRIGGER_PWM_GEN2_CURRENT_LMT_ADC	PWM Generator 2 current-limit ADC trigger
ADC_CONVERSION_TRIGGER_PWM_GEN3_CURRENT_LMT_ADC	PWM Generator 3 current-limit ADC trigger
ADC_CONVERSION_TRIGGER_PWM_GEN4_CURRENT_LMT_ADC	PWM Generator 4 current-limit ADC trigger
ADC_CONVERSION_TRIGGER_PWM_GEN5_CURRENT_LMT_ADC	PWM Generator 5 current-limit ADC trigger
ADC_CONVERSION_TRIGGER_PWM_GEN6_CURRENT_LMT_ADC	PWM Generator 6 current-limit ADC trigger
ADC_CONVERSION_TRIGGER_PWM_GEN7_CURRENT_LMT_ADC	PWM Generator 7 current-limit ADC trigger
ADC_CONVERSION_TRIGGER_PWM_GEN8_CURRENT_LMT_ADC	PWM Generator 8 current-limit ADC trigger
ADC_CONVERSION_TRIGGER_TMR2_PERIOD_MATCH	Timer2 period match

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.8 ADC_RESULT_SIZE Enumeration

C

```
typedef enum {
    ADC_RESULT_SIZE_8_BITS,
    ADC_RESULT_SIZE_10_BITS,
    ADC_RESULT_SIZE_12_BITS
} ADC_RESULT_SIZE;
```

Description

ADC Result Size

This data type defines the ADC Result Size.

Members

Members	Description
ADC_RESULT_SIZE_8_BITS	Converter Mode/Result Size is 8-bits
ADC_RESULT_SIZE_10_BITS	Converter Mode/Result Size is 10-bits
ADC_RESULT_SIZE_12_BITS	Converter Mode/Result Size is 12-bits

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.9 ADC_BUFFER_MODE Enumeration

C

```
typedef enum {
    ADC_BUFFER_MODE_ONE_16WORD_BUFFER,
    ADC_BUFFER_MODE_TWO_8WORD_BUFFERS
} ADC_BUFFER_MODE;
```

Description

ADC Buffer Mode

This data type defines the ADC Buffer Mode.

Members

Members	Description
ADC_BUFFER_MODE_ONE_16WORD_BUFFER	Buffer configured as one 16-word buffer (ADC1BUF0 to ADC1BUFF)
ADC_BUFFER_MODE_TWO_8WORD_BUFFERS	Buffer configured as two 8-word buffers (ADC1BUF0 to ADC1BUF7 and ADC1BUF8 to ADC1BUFF)

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.10 ADC_RESULT_BUF_STATUS Enumeration**C**

```
typedef enum {  
    ADC_FILLING_BUF_0TO7,  
    ADC_FILLING_BUF_8TOF  
} ADC_RESULT_BUF_STATUS;
```

Description

ADC Result Buffer Status

This data type defines the elements that specify which group of eight buffers the ADC is currently filling.

Members

Members	Description
ADC_FILLING_BUF_0TO7	Buffers 0x0 to 0x7 are getting filled
ADC_FILLING_BUF_8TOF	Buffers 0x8 to 0xF are getting filled

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.11 ADC_MUX Enumeration**C**

```
typedef enum {  
    ADC_MUX_A,  
    ADC_MUX_B  
} ADC_MUX;
```

Description

ADC MUX Enumeration

This data type defines the different ADC MUX Enumeration.

Members

Members	Description
ADC_MUX_A	ADC Mux A Selection
ADC_MUX_B	ADC Mux B Selection

Remarks

None

5.6.1.6.8.12 ADC_SAMPLING_MODE Enumeration**C**

```
typedef enum {  
    ADC_SAMPLING_MODE_SIMULTANEOUS,  
    ADC_SAMPLING_MODE_SEQUENTIAL  
} ADC_SAMPLING_MODE;
```

Description

ADC Sampling Mode Select

This data type defines the ADC Sampling Mode Select.

Members

Members	Description
ADC_SAMPLING_MODE_SIMULTANEOUS	Simultaneous Sampling Mode
ADC_SAMPLING_MODE_SEQUENTIAL	Sequential Sampling Mode

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.13 ADC_INPUTS_SCAN Enumeration**C**

```
typedef enum {
    ADC_INPUT_SCAN_AN0,
    ADC_INPUT_SCAN_AN1,
    ADC_INPUT_SCAN_AN2,
    ADC_INPUT_SCAN_AN3,
    ADC_INPUT_SCAN_AN4,
    ADC_INPUT_SCAN_AN5,
    ADC_INPUT_SCAN_AN6,
    ADC_INPUT_SCAN_AN7,
    ADC_INPUT_SCAN_AN8,
    ADC_INPUT_SCAN_AN9,
    ADC_INPUT_SCAN_AN10,
    ADC_INPUT_SCAN_AN11,
    ADC_INPUT_SCAN_AN12,
    ADC_INPUT_SCAN_AN13,
    ADC_INPUT_SCAN_AN14,
    ADC_INPUT_SCAN_AN15,
    ADC_INPUT_SCAN_AN16,
    ADC_INPUT_SCAN_AN17,
    ADC_INPUT_SCAN_AN18,
    ADC_INPUT_SCAN_AN19,
    ADC_INPUT_SCAN_AN20,
    ADC_INPUT_SCAN_AN21,
    ADC_INPUT_SCAN_AN22,
    ADC_INPUT_SCAN_AN23,
    ADC_INPUT_SCAN_AN24,
    ADC_INPUT_SCAN_AN25,
    ADC_INPUT_SCAN_AN26,
    ADC_INPUT_SCAN_AN27,
    ADC_INPUT_SCAN_AN28,
    ADC_INPUT_SCAN_AN29,
    ADC_INPUT_SCAN_AN30,
    ADC_INPUT_SCAN_AN31,
    ADC_INPUT_SCAN_IVREF,
    ADC_INPUT_SCAN_CTMU,
    ADC_INPUT_SCAN_VSS,
    ADC_INPUT_SCAN_VBG_DIV_BY_6,
    ADC_INPUT_SCAN_VBG_DIV_BY_2,
    ADC_INPUT_SCAN_VBG_DIV_BY_1,
    ADC_INPUT_SCAN_VDDCORE,
    ADC_INPUT_SCAN_ALL
} ADC_INPUTS_SCAN;
```

Description

ADC Scan Inputs Enumeration

This data type defines the ADC Scan inputs.

Members

Members	Description
ADC_INPUT_SCAN_AN0	Scan input is AN0
ADC_INPUT_SCAN_AN1	Scan input is AN1
ADC_INPUT_SCAN_AN2	Scan input is AN2
ADC_INPUT_SCAN_AN3	Scan input is AN3
ADC_INPUT_SCAN_AN4	Scan input is AN4
ADC_INPUT_SCAN_AN5	Scan input is AN5
ADC_INPUT_SCAN_AN6	Scan input is AN6
ADC_INPUT_SCAN_AN7	Scan input is AN7
ADC_INPUT_SCAN_AN8	Scan input is AN8
ADC_INPUT_SCAN_AN9	Scan input is AN9
ADC_INPUT_SCAN_AN10	Scan input is AN10
ADC_INPUT_SCAN_AN11	Scan input is AN11
ADC_INPUT_SCAN_AN12	Scan input is AN12
ADC_INPUT_SCAN_AN13	Scan input is AN13
ADC_INPUT_SCAN_AN14	Scan input is AN14
ADC_INPUT_SCAN_AN15	Scan input is AN15
ADC_INPUT_SCAN_AN16	Scan input is AN16
ADC_INPUT_SCAN_AN17	Scan input is AN17
ADC_INPUT_SCAN_AN18	Scan input is AN18
ADC_INPUT_SCAN_AN19	Scan input is AN19
ADC_INPUT_SCAN_AN20	Scan input is AN20
ADC_INPUT_SCAN_AN21	Scan input is AN21
ADC_INPUT_SCAN_AN22	Scan input is AN22
ADC_INPUT_SCAN_AN23	Scan input is AN23
ADC_INPUT_SCAN_AN24	Scan input is AN24
ADC_INPUT_SCAN_AN25	Scan input is AN25
ADC_INPUT_SCAN_AN26	Scan input is AN26
ADC_INPUT_SCAN_AN27	Scan input is AN27
ADC_INPUT_SCAN_AN28	Scan input is AN28
ADC_INPUT_SCAN_AN29	Scan input is AN29
ADC_INPUT_SCAN_AN30	Scan input is AN30
ADC_INPUT_SCAN_AN31	Scan input is AN31
ADC_INPUT_SCAN_IVREF	Scan input is IVref
ADC_INPUT_SCAN_CTMU	Scan input is CTMU input
ADC_INPUT_SCAN_VSS	Scan input is Vss
ADC_INPUT_SCAN_VBG_DIV_BY_6	Scan input is VBG/6
ADC_INPUT_SCAN_VBG_DIV_BY_2	Scan input is VBG/2
ADC_INPUT_SCAN_VBG_DIV_BY_1	Scan input is VBG
ADC_INPUT_SCAN_VDDCORE	Scan input is VDDCORE

ADC_INPUT_SCAN_ALL	All Inputs

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.14 ADC_REFERENCE_INPUT Enumeration**C**

```
typedef enum {
    ADC_REFERENCE_INPUT_VBG_DIV_BY_1,
    ADC_REFERENCE_INPUT_VBG_DIV_BY_2,
    ADC_REFERENCE_INPUT_VBG_DIV_BY_6,
    ADC_REFERENCE_INPUT_VDDCORE
} ADC_REFERENCE_INPUT;
```

Description

ADC Reference Input Enumeration

This data type defines the ADC Reference Input Enumeration.

Members

Members	Description
ADC_REFERENCE_INPUT_VBG_DIV_BY_1	Internal Band Gap Reference Channel VBG
ADC_REFERENCE_INPUT_VBG_DIV_BY_2	Internal Band Gap Reference Channel VBG/2
ADC_REFERENCE_INPUT_VBG_DIV_BY_6	Internal Band Gap Reference Channel VBG/6
ADC_REFERENCE_INPUT_VDDCORE	Internal Reference Channel VDDCORE

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.15 ADC_RESULT_FORMAT Enumeration**C**

```
typedef enum {
    ADC_RESULT_FORMAT_LEFT_JUSTIFIED,
    ADC_RESULT_FORMAT_RIGHT_JUSTIFIED,
    ADC_RESULT_FORMAT_INTEGER_16BIT,
    ADC_RESULT_FORMAT_SIGNED_INTEGER_16BIT,
    ADC_RESULT_FORMAT_FRACTIONAL_16BIT,
    ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_16BIT,
    ADC_RESULT_FORMAT_INTEGER_32BIT,
    ADC_RESULT_FORMAT_SIGNED_INTEGER_32BIT,
    ADC_RESULT_FORMAT_FRACTIONAL_32BIT,
    ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_32BIT
} ADC_RESULT_FORMAT;
```

Description

ADC Result Format

This data type defines the ADC Result Format.

Members

Members	Description
ADC_RESULT_FORMAT_LEFT_JUSTIFIED	Result Left justified
ADC_RESULT_FORMAT_RIGHT_JUSTIFIED	Result Right justified

ADC_RESULT_FORMAT_INTEGER_16BIT	Integer 16-bit
ADC_RESULT_FORMAT_SIGNED_INTEGER_16BIT	Signed Integer 16-bit
ADC_RESULT_FORMAT_FRACTIONAL_16BIT	Fractional 16-bit
ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_16BIT	Signed Fractional 16-bit
ADC_RESULT_FORMAT_INTEGER_32BIT	Integer 32-bit
ADC_RESULT_FORMAT_SIGNED_INTEGER_32BIT	Signed Integer 32-bit
ADC_RESULT_FORMAT_FRACTIONAL_32BIT	Fractional 32-bit
ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_32BIT	Signed Fractional 32-bit

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.16 ADC_DMA_BUFFER_MODE Enumeration**C**

```
typedef enum {
    ADC_DMA_BUFFER_MODE_SCATTER_GATHER,
    ADC_DMA_BUFFER_MODE_ORDER_OF_CONV
} ADC_DMA_BUFFER_MODE;
```

Description

ADC DMA Buffer Mode

This data type defines the ADC Buffer Mode.

Members

Members	Description
ADC_DMA_BUFFER_MODE_SCATTER_GATHER	DMA buffers are written in Scatter/Gather mode
ADC_DMA_BUFFER_MODE_ORDER_OF_CONV	DMA buffers are written in the order of conversion

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.17 ADC_PAIR Enumeration**C**

```
typedef enum {
    ADC_PAIR_AN0_AN1,
    ADC_PAIR_AN2_AN3,
    ADC_PAIR_AN4_AN5,
    ADC_PAIR_AN6_AN7,
    ADC_PAIR_AN8_AN9,
    ADC_PAIR_AN10_AN11,
    ADC_PAIR_AN12_AN13,
    ADC_PAIR_AN14_AN15,
    ADC_PAIR_AN16_AN17,
    ADC_PAIR_AN18_AN19,
    ADC_PAIR_AN20_AN21,
    ADC_PAIR_AN22_AN23,
    ADC_PAIR_AN24_AN25
} ADC_PAIR;
```

Description

ADC Pair

This data type defines the ADC Pair.

Members

Members	Description
ADC_PAIR_AN0_AN1	Pair 0 - AN0, AN1
ADC_PAIR_AN2_AN3	Pair 1 - AN2, AN3
ADC_PAIR_AN4_AN5	Pair 2 - AN4, AN5
ADC_PAIR_AN6_AN7	Pair 3 - AN6, AN7
ADC_PAIR_AN8_AN9	Pair 4 - AN8, AN9
ADC_PAIR_AN10_AN11	Pair 5 - AN10, AN11
ADC_PAIR_AN12_AN13	Pair 6 - AN12, AN13
ADC_PAIR_AN14_AN15	Pair 7 - AN14, AN15
ADC_PAIR_AN16_AN17	Pair 8 - AN16, AN17
ADC_PAIR_AN18_AN19	Pair 9 - AN18, AN19
ADC_PAIR_AN20_AN21	Pair 10 - AN20, AN21
ADC_PAIR_AN22_AN23	Pair 11 - AN22, AN23
ADC_PAIR_AN24_AN25	Pair 12 - AN24, AN25

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.18 ADC_CHANNEL123_INPUTS_NEG Enumeration**C**

```
typedef enum {
    ADC_CHANNEL123_INPUT_NEG_CH123_VREF_MINUS,
    ADC_CHANNEL123_INPUT_NEG_CH1_AN6_CH2_AN7_CH3_NC,
    ADC_CHANNEL123_INPUT_NEG_CH1_AN6_CH2_AN7_CH3_AN8,
    ADC_CHANNEL123_INPUT_NEG_CH1_AN9_CH2_AN10_CH3_AN11,
    ADC_CHANNEL123_INPUT_NEG_CH1_AN9_CH2_CH3_NC
} ADC_CHANNEL123_INPUTS_NEG;
```

Description

ADC Channel 123 Negative Input Select

This data type defines the ADC Channel 123 Negative Input Select.

Members

Members	Description
ADC_CHANNEL123_INPUT_NEG_CH123_VREF_MINUS	CH1, CH2, CH3 negative input is VREF
ADC_CHANNEL123_INPUT_NEG_CH1_AN6_CH2_AN7_CH3_NC	CH1 negative input is AN6, CH2 negative input is AN7, CH3 negative input is not connected
ADC_CHANNEL123_INPUT_NEG_CH1_AN6_CH2_AN7_CH3_AN8	CH1 negative input is AN6, CH2 negative input is AN7, CH3 negative input is AN8
ADC_CHANNEL123_INPUT_NEG_CH1_AN9_CH2_AN10_CH3_AN11	CH1 negative input is AN9, CH2 negative input is AN10, CH3 negative input is AN11
ADC_CHANNEL123_INPUT_NEG_CH1_AN9_CH2_CH3_NC	CH1 negative input is AN9, CH2 and CH3 negative inputs are not connected

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.19 ADC_CHANNEL123_INPUTS_POS Enumeration

C

```
typedef enum {
    ADC_CHANNEL123_INPUT_POS_CH1_AN3_CH2_CH3_NC,
    ADC_CHANNEL123_INPUT_POS_CH1_AN0_CH2_AN1_CH3_AN2,
    ADC_CHANNEL123_INPUT_POS_CH1_AN3_CH2_AN4_CH3_AN5
} ADC_CHANNEL123_INPUTS_POS;
```

Description

ADC Channel 123 Positive Input Select

This data type defines the ADC Channel 123 Positive Input Select.

Members

Members	Description
ADC_CHANNEL123_INPUT_POS_CH1_AN3_CH2_CH3_NC	CH1 positive input is AN3, CH2 and CH3 positive inputs are not connected
ADC_CHANNEL123_INPUT_POS_CH1_AN0_CH2_AN1_CH3_AN2	CH1 positive input is AN0, CH2 positive input is AN1, CH3 positive input is AN2
ADC_CHANNEL123_INPUT_POS_CH1_AN3_CH2_AN4_CH3_AN5	CH1 positive input is AN3, CH2 positive input is AN4, CH3 positive input is AN5

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.20 ADC_INPUTS_POSITIVE Enumeration

C

```
typedef enum {
    ADC_INPUT_POSITIVE_AN0,
    ADC_INPUT_POSITIVE_AN1,
    ADC_INPUT_POSITIVE_AN2,
    ADC_INPUT_POSITIVE_AN3,
    ADC_INPUT_POSITIVE_AN4,
    ADC_INPUT_POSITIVE_AN5,
    ADC_INPUT_POSITIVE_AN6,
    ADC_INPUT_POSITIVE_AN7,
    ADC_INPUT_POSITIVE_AN8,
    ADC_INPUT_POSITIVE_AN9,
    ADC_INPUT_POSITIVE_AN10,
    ADC_INPUT_POSITIVE_AN11,
    ADC_INPUT_POSITIVE_AN12,
    ADC_INPUT_POSITIVE_AN13,
    ADC_INPUT_POSITIVE_AN14,
    ADC_INPUT_POSITIVE_AN15,
    ADC_INPUT_POSITIVE_AN16,
    ADC_INPUT_POSITIVE_AN17,
    ADC_INPUT_POSITIVE_AN18,
    ADC_INPUT_POSITIVE_AN19,
    ADC_INPUT_POSITIVE_AN20,
    ADC_INPUT_POSITIVE_AN21,
    ADC_INPUT_POSITIVE_AN22,
    ADC_INPUT_POSITIVE_AN23,
    ADC_INPUT_POSITIVE_AN24,
    ADC_INPUT_POSITIVE_AN25,
    ADC_INPUT_POSITIVE_AN26,
    ADC_INPUT_POSITIVE_AN27,
    ADC_INPUT_POSITIVE_AN28,
```

```

ADC_INPUT_POSITIVE_AN29,
ADC_INPUT_POSITIVE_AN30,
ADC_INPUT_POSITIVE_AN31,
ADC_INPUT_POSITIVE_ABS_VOLTAGE_REF,
ADC_INPUT_POSITIVE_MUX_DISCONNECT,
ADC_INPUT_POSITIVE_TEMP_INDICATOR,
ADC_INPUT_POSITIVE_DAC_OUTPUT,
ADC_INPUT_POSITIVE_FVR_BUFFER1_OUTPUT,
ADC_INPUT_POSITIVE_VBG_DIV_BY_6,
ADC_INPUT_POSITIVE_VBG_DIV_BY_2,
ADC_INPUT_POSITIVE_VBG_DIV_BY_1,
ADC_INPUT_POSITIVE_VDDCORE,
ADC_INPUT_POSITIVE_MAX,
ADC_INPUT_POSITIVE_CTMU,
ADC_INPUT_POSITIVE_IVREF,
ADC_INPUT_POSITIVE_OPEN
} ADC_INPUTS_POSITIVE;

```

Description

ADC Inputs Enumeration

This data type defines the ADC inputs.

Members

Members	Description
ADC_INPUT_POSITIVE_AN0	Positive input is AN0
ADC_INPUT_POSITIVE_AN1	Positive input is AN1
ADC_INPUT_POSITIVE_AN2	Positive input is AN2
ADC_INPUT_POSITIVE_AN3	Positive input is AN3
ADC_INPUT_POSITIVE_AN4	Positive input is AN4
ADC_INPUT_POSITIVE_AN5	Positive input is AN5
ADC_INPUT_POSITIVE_AN6	Positive input is AN6
ADC_INPUT_POSITIVE_AN7	Positive input is AN7
ADC_INPUT_POSITIVE_AN8	Positive input is AN8
ADC_INPUT_POSITIVE_AN9	Positive input is AN9
ADC_INPUT_POSITIVE_AN10	Positive input is AN10
ADC_INPUT_POSITIVE_AN11	Positive input is AN11
ADC_INPUT_POSITIVE_AN12	Positive input is AN12
ADC_INPUT_POSITIVE_AN13	Positive input is AN13
ADC_INPUT_POSITIVE_AN14	Positive input is AN14
ADC_INPUT_POSITIVE_AN15	Positive input is AN15
ADC_INPUT_POSITIVE_AN16	Positive input is AN16
ADC_INPUT_POSITIVE_AN17	Positive input is AN17
ADC_INPUT_POSITIVE_AN18	Positive input is AN18
ADC_INPUT_POSITIVE_AN19	Positive input is AN19
ADC_INPUT_POSITIVE_AN20	Positive input is AN20
ADC_INPUT_POSITIVE_AN21	Positive input is AN21
ADC_INPUT_POSITIVE_AN22	Positive input is AN22
ADC_INPUT_POSITIVE_AN23	Positive input is AN23
ADC_INPUT_POSITIVE_AN24	Positive input is AN24
ADC_INPUT_POSITIVE_AN25	Positive input is AN25
ADC_INPUT_POSITIVE_AN26	Positive input is AN26

ADC_INPUT_POSITIVE_AN27	Positive input is AN27
ADC_INPUT_POSITIVE_AN28	Positive input is AN28
ADC_INPUT_POSITIVE_AN29	Positive input is AN29
ADC_INPUT_POSITIVE_AN30	Positive input is AN30
ADC_INPUT_POSITIVE_AN31	Positive input is AN31
ADC_INPUT_POSITIVE_ABS_VOLTAGE_REF	ADC Channel 0.6V Absolute Voltage Reference
ADC_INPUT_POSITIVE_MUX_DISCONNECT	ADC Channel MUX Disconnect
ADC_INPUT_POSITIVE_TEMP_INDICATOR	ADC Channel Temperature Diode/Indicator
ADC_INPUT_POSITIVE_DAC_OUTPUT	ADC Channel DAC Output
ADC_INPUT_POSITIVE_FVR_BUFFER1_OUTPUT	ADC Channel Fixed Voltage Reference Buffer1 Output
ADC_INPUT_POSITIVE_VBG_DIV_BY_6	Positive input is VBG/6
ADC_INPUT_POSITIVE_VBG_DIV_BY_2	Positive input is VBG/2
ADC_INPUT_POSITIVE_VBG_DIV_BY_1	Positive input is VBG
ADC_INPUT_POSITIVE_VDDCORE	Positive input is VDDCORE
ADC_INPUT_POSITIVE_MAX	Maximum Inputs
ADC_INPUT_POSITIVE_CTMU	Positive input is from CTMU
ADC_INPUT_POSITIVE_IVREF	Positive input is IVref
ADC_INPUT_POSITIVE_OPEN	Positive input is Open

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.21 ADC_CONVERSION_ORDER Enumeration**C**

```
typedef enum {
    ADC_CONVERSION_ORDER_ODD_FIRST,
    ADC_CONVERSION_ORDER_EVEN_FIRST
} ADC_CONVERSION_ORDER;
```

Description

ADC Conversion Order

This data type defines the ADC Conversion Order.

Members

Members	Description
ADC_CONVERSION_ORDER_ODD_FIRST	Odd numbered analog input is converted first, followed by conversion of even numbered input
ADC_CONVERSION_ORDER_EVEN_FIRST	Even numbered analog input is converted first, followed by conversion of odd numbered input

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.22 ADC_DMA_ADDRESS_INCREMENT Enumeration**C**

```
typedef enum {
    ADC_DMA_ADDRESS_INCREMENT_EVERY_SAMPLE,
    ADC_DMA_ADDRESS_INCREMENT_EVERY_2ND_SAMPLE,
```

```

ADC_DMA_ADDRESS_INCREMENT EVERY 3RD SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 4TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 5TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 6TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 7TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 8TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 9TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 10TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 11TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 12TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 13TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 14TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 15TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 16TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 17TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 18TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 19TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 20TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 21ST SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 22ND SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 23RD SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 24TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 25TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 26TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 27TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 28TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 29TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 30TH SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 31ST SAMPLE,
ADC_DMA_ADDRESS_INCREMENT EVERY 32ND SAMPLE
} ADC_DMA_ADDRESS_INCREMENT;

```

Description

ADC DMA Increment Rate

This data type defines the ADC DMA Increment Rate.

Members

Members	Description
ADC_DMA_ADDRESS_INCREMENT EVERY SAMPLE	Increments the DMA address after completion of every sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY 2ND SAMPLE	Increments the DMA address after completion of every 2nd sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY 3RD SAMPLE	Increments the DMA address after completion of every 3rd sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY 4TH SAMPLE	Increments the DMA address after completion of every 4th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY 5TH SAMPLE	Increments the DMA address after completion of every 5th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY 6TH SAMPLE	Increments the DMA address after completion of every 6th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY 7TH SAMPLE	Increments the DMA address after completion of every 7th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY 8TH SAMPLE	Increments the DMA address after completion of every 8th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY 9TH SAMPLE	Increments the DMA address after completion of every 9th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY 10TH SAMPLE	Increments the DMA address after completion of every 10th sample/conversion operation

ADC_DMA_ADDRESS_INCREMENT EVERY_11TH_SAMPLE	Increments the DMA address after completion of every 11th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_12TH_SAMPLE	Increments the DMA address after completion of every 12th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_13TH_SAMPLE	Increments the DMA address after completion of every 13th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_14TH_SAMPLE	Increments the DMA address after completion of every 14th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_15TH_SAMPLE	Increments the DMA address after completion of every 15th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_16TH_SAMPLE	Increments the DMA address after completion of every 16th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_17TH_SAMPLE	Increments the DMA address after completion of every 17th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_18TH_SAMPLE	Increments the DMA address after completion of every 18th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_19TH_SAMPLE	Increments the DMA address after completion of every 19th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_20TH_SAMPLE	Increments the DMA address after completion of every 20th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_21ST_SAMPLE	Increments the DMA address after completion of every 21st sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_22ND_SAMPLE	Increments the DMA address after completion of every 22nd sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_23RD_SAMPLE	Increments the DMA address after completion of every 23rd sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_24TH_SAMPLE	Increments the DMA address after completion of every 24th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_25TH_SAMPLE	Increments the DMA address after completion of every 25th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_26TH_SAMPLE	Increments the DMA address after completion of every 26th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_27TH_SAMPLE	Increments the DMA address after completion of every 27th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_28TH_SAMPLE	Increments the DMA address after completion of every 28th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_29TH_SAMPLE	Increments the DMA address after completion of every 29th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_30TH_SAMPLE	Increments the DMA address after completion of every 30th sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_31ST_SAMPLE	Increments the DMA address after completion of every 31st sample/conversion operation
ADC_DMA_ADDRESS_INCREMENT EVERY_32ND_SAMPLE	Increments the DMA address after completion of every 32nd sample/conversion operation

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.23 ADC_DMA_INPUT_BUFFER Enumeration**C**

```
typedef enum {
```

```

ADC_DMA_INPUT_BUFFER_1WORD,
ADC_DMA_INPUT_BUFFER_2WORDS,
ADC_DMA_INPUT_BUFFER_4WORDS,
ADC_DMA_INPUT_BUFFER_8WORDS,
ADC_DMA_INPUT_BUFFER_16WORDS,
ADC_DMA_INPUT_BUFFER_32WORDS,
ADC_DMA_INPUT_BUFFER_64WORDS,
ADC_DMA_INPUT_BUFFER_128WORDS
} ADC_DMA_INPUT_BUFFER;

```

Description

ADC DMA Buffer Per Input

This data type defines the ADC DMA Buffer Per Input.

Members

Members	Description
ADC_DMA_INPUT_BUFFER_1WORD	Allocates 1 word of buffer to each analog input
ADC_DMA_INPUT_BUFFER_2WORDS	Allocates 2 words of buffer to each analog input
ADC_DMA_INPUT_BUFFER_4WORDS	Allocates 4 words of buffer to each analog input
ADC_DMA_INPUT_BUFFER_8WORDS	Allocates 8 words of buffer to each analog input
ADC_DMA_INPUT_BUFFER_16WORDS	Allocates 16 words of buffer to each analog input
ADC_DMA_INPUT_BUFFER_32WORDS	Allocates 32 words of buffer to each analog input
ADC_DMA_INPUT_BUFFER_64WORDS	Allocates 64 words of buffer to each analog input
ADC_DMA_INPUT_BUFFER_128WORDS	Allocates 128 words of buffer to each analog input

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

5.6.1.6.8.24 ADC_ACQUISITION_TIME Type**C**

```
typedef unsigned int ADC_ACQUISITION_TIME;
```

Description

ADC Acquisition Time Selection Data Type

This data type defines the different ADC acquisition times by which the ADC can be configured.

Remarks

None

5.6.1.6.8.25 ADC_CONVERSION_CLOCK Type**C**

```
typedef unsigned int ADC_CONVERSION_CLOCK;
```

Description

ADC Conversion Clock Selection Data Type

This data type defines the different ADC Conversion clock

Remarks

None

5.6.1.6.8.26 ADC_SAMPLE Type

C

```
typedef unsigned int ADC_SAMPLE;
```

Description

ADC Sample size

This data type defines the size of the adc sample register.

Remarks

None

5.6.1.6.9 Feature Existence

5.6.1.6.9.1 PLIB_ADC_ExistsAsynchronousDedicatedSampling Function

C

```
bool PLIB_ADC_ExistsAsynchronousDedicatedSampling(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the AsynchronousDedicatedSampling feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_AsynchronousDedicatedSamplingEnable](#)
- [PLIB_ADC_AsynchronousDedicatedSamplingDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The AsynchronousDedicatedSampling feature is supported on the device
- false = The AsynchronousDedicatedSampling feature is not supported on the device

Remarks

None.

5.6.1.6.9.2 PLIB_ADC_ExistsCalibrationControl Function

C

```
bool PLIB_ADC_ExistsCalibrationControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the CalibrationControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_CalibrationEnable](#)
- [PLIB_ADC_CalibrationDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The CalibrationControl feature is supported on the device
- false = The CalibrationControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.3 PLIB_ADC_ExistsChannelGroup Function

C

```
bool PLIB_ADC_ExistsChannelGroup(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelGroup feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ChannelGroupSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ChannelGroup feature is supported on the device
- false = The ChannelGroup feature is not supported on the device

Remarks

None.

5.6.1.6.9.4 PLIB_ADC_ExistsConversionClock Function

C

```
bool PLIB_ADC_ExistsConversionClock(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ConversionClock feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ConversionClockSet](#)
- [PLIB_ADC_ConversionClockGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ConversionClock feature is supported on the device
- false = The ConversionClock feature is not supported on the device

Remarks

None.

5.6.1.6.9.5 PLIB_ADC_ExistsConversionClockSource Function

C

```
bool PLIB_ADC_ExistsConversionClockSource(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ConversionClockSource feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ConversionClockSourceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ConversionClockSource feature is supported on the device
- false = The ConversionClockSource feature is not supported on the device

Remarks

None.

5.6.1.6.9.6 PLIB_ADC_ExistsConversionControl Function

C

```
bool PLIB_ADC_ExistsConversionControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ConversionControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ConversionStart](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = When ConversionControl feature is supported on the device
- false = When ConversionControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.7 PLIB_ADC_ExistsConversionOrder Function**C**

```
bool PLIB_ADC_ExistsConversionOrder(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ConversionOrder feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ConversionOrderSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ConversionOrder feature is supported on the device
- false = The ConversionOrder feature is not supported on the device

Remarks

None.

5.6.1.6.9.8 PLIB_ADC_ExistsConversionStatus Function**C**

```
bool PLIB_ADC_ExistsConversionStatus(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ConversionStatus feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ConversionHasCompleted](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ConversionStatus feature is supported on the device
- false = The ConversionStatus feature is not supported on the device

Remarks

None.

5.6.1.6.9.9 PLIB_ADC_ExistsConversionStopSequenceControl Function**C**

```
bool PLIB_ADC_ExistsConversionStopSequenceControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ConversionStopSequenceControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ConversionStopSequenceEnable](#)
- [PLIB_ADC_ConversionStopSequenceDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ConversionStopSequenceControl feature is supported on the device
- false = The ConversionStopSequenceControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.10 PLIB_ADC_ExistsConversionTriggerGroup Function**C**

```
bool PLIB_ADC_ExistsConversionTriggerGroup(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ConversionTriggerGroup feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ConversionTriggerGroupSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ConversionTriggerGroup feature is supported on the device
- false = The ConversionTriggerGroup feature is not supported on the device

Remarks

None.

5.6.1.6.9.11 PLIB_ADC_ExistsConversionTriggerSource Function**C**

```
bool PLIB_ADC_ExistsConversionTriggerSource(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ConversionTriggerSource feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ConversionTriggerSourceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ConversionTriggerSource feature is supported on the device
- false = The ConversionTriggerSource feature is not supported on the device

Remarks

None.

5.6.1.6.9.12 PLIB_ADC_ExistsDMAAddressIncrement Function**C**

```
bool PLIB_ADC_ExistsDMAAddressIncrement(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the DMAAddressIncrement feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_DMAAddressIncrementSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The DMAAddressIncrement feature is supported on the device
- false = The DMAAddressIncrement feature is not supported on the device

Remarks

None.

5.6.1.6.9.13 PLIB_ADC_ExistsDMABufferMode Function**C**

```
bool PLIB_ADC_ExistsDMABufferMode(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the DMABufferMode feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_DMABufferModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The DMABufferMode feature is supported on the device
- false = The DMABufferMode feature is not supported on the device

Remarks

None.

5.6.1.6.9.14 PLIB_ADC_ExistsDMABuffersPerAnalogInput Function**C**

```
bool PLIB_ADC_ExistsDMABuffersPerAnalogInput(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the DMABuffersPerInput feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_DMAInputBufferSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The DMABuffersPerInput feature is supported on the device
- false = The DMABuffersPerInput feature is not supported on the device

Remarks

None.

5.6.1.6.9.15 PLIB_ADC_ExistsDMAControl Function**C**

```
bool PLIB_ADC_ExistsDMAControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the DMAControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_DMAEnable](#)
- [PLIB_ADC_DMADisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The DMAControl feature is supported on the device
- false = The DMAControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.16 PLIB_ADC_ExistsEnableControl Function**C**

```
bool PLIB_ADC_ExistsEnableControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_Enable](#)
- [PLIB_ADC_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The EnableControl feature is supported on the device
- false = The EnableControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.17 PLIB_ADC_ExistsGlobalSoftwareTrigger Function**C**

```
bool PLIB_ADC_ExistsGlobalSoftwareTrigger(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the GlobalSoftwareTrigger feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_GlobalSoftwareTriggerSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The GlobalSoftwareTrigger feature is supported on the device
- false = The GlobalSoftwareTrigger feature is not supported on the device

Remarks

None.

5.6.1.6.9.18 PLIB_ADC_ExistsInputSelect Function**C**

```
bool PLIB_ADC_ExistsInputSelect(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the InputSelect feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_InputSelectPositive](#)
- [PLIB_ADC_InputSelectNegative](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The InputSelect feature is supported on the device
- false = The InputSelect feature is not supported on the device

Remarks

None.

5.6.1.6.9.19 PLIB_ADC_ExistsInternalReferenceChannelControl Function**C**

```
bool PLIB_ADC_ExistsInternalReferenceChannelControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the InternalReferenceChannelControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_InternalReferenceChannelEnable](#)
- [PLIB_ADC_InternalReferenceChannelDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The InternalReferenceChannelControl feature is supported on the device
- false = The InternalReferenceChannelControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.20 PLIB_ADC_ExistsISRJumpTableBaseAddress Function**C**

```
bool PLIB_ADC_ExistsISRJumpTableBaseAddress(  
    ADC_MODULE_ID index  
);
```


Description

This function identifies whether the `ISRJumpTableBaseAddress` feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_IsrJumpTableBaseAddressSet](#)
- [PLIB_ADC_IsrJumpTableBaseAddressGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The `ISRJumpTableBaseAddress` feature is supported on the device
- false = The `ISRJumpTableBaseAddress` feature is not supported on the device

Remarks

None.

5.6.1.6.9.21 PLIB_ADC_ExistsMuxChannel0NegativeInput Function**C**

```
bool PLIB_ADC_ExistsMuxChannel0NegativeInput(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the `MuxChannel0NegativeInput` feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_MuxChannel0InputNegativeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The `MuxChannel0NegativeInput` feature is supported on the device
- false = The `MuxChannel0NegativeInput` feature is not supported on the device

Remarks

None.

5.6.1.6.9.22 PLIB_ADC_ExistsMuxChannel0PositiveInput Function**C**

```
bool PLIB_ADC_ExistsMuxChannel0PositiveInput(  
    ADC_MODULE_ID index
```

```
);
```

Description

This function identifies whether the MuxChannel0PositiveInput feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_MuxChannel0InputPositiveSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The MuxChannel0PositiveInput feature is supported on the device
- false = The MuxChannel0PositiveInput feature is not supported on the device

Remarks

None.

5.6.1.6.9.23 PLIB_ADC_ExistsMuxChannel123NegativeInput Function

C

```
bool PLIB_ADC_ExistsMuxChannel123NegativeInput(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the MuxChannel123NegativeInput feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_MuxChannel123InputNegativeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The MuxChannel123NegativeInput feature is supported on the device
- false = The MuxChannel123NegativeInput feature is not supported on the device

Remarks

None.

5.6.1.6.9.24 PLIB_ADC_ExistsMuxChannel123PositiveInput Function

C

```
bool PLIB_ADC_ExistsMuxChannel123PositiveInput(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the MuxChannel123PositiveInput feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_MuxChannel123InputPositiveSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The MuxChannel123PositiveInput feature is supported on the device
- false = The MuxChannel123PositiveInput feature is not supported on the device

Remarks

None.

5.6.1.6.9.25 PLIB_ADC_ExistsMuxInputScanControl Function**C**

```
bool PLIB_ADC_ExistsMuxInputScanControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the MuxInputScanControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_MuxAInputScanEnable](#)
- [PLIB_ADC_MuxAInputScanDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The MuxInputScanControl feature is supported on the device
- false = The MuxInputScanControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.26 PLIB_ADC_ExistsMuxInputScanSelect Function**C**

```
bool PLIB_ADC_ExistsMuxInputScanSelect(  
    ADC_MODULE_ID index
```

```
);
```

Description

This function identifies whether the MuxInputScanSelect feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_InputScanMaskAdd](#)
- [PLIB_ADC_InputScanMaskRemove](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The MuxInputScanSelect feature is supported on the device
- false = The MuxInputScanSelect feature is not supported on the device

Remarks

None.

5.6.1.6.9.27 PLIB_ADC_ExistsPairConversionControl Function

C

```
bool PLIB_ADC_ExistsPairConversionControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the PairConversionControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_PairConversionStart](#)
- [PLIB_ADC_PairConversionIsPending](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PairConversionControl feature is supported on the device
- false = The PairConversionControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.28 PLIB_ADC_ExistsPairInterruptOnConversion Function

C

```
bool PLIB_ADC_ExistsPairInterruptOnConversion(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the PairInterruptOnConversion feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_PairInterruptAfterFirstConversion](#)
- [PLIB_ADC_PairInterruptAfterSecondConversion](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PairInterruptOnConversion feature is supported on the device
- false = The PairInterruptOnConversion feature is not supported on the device

Remarks

None.

5.6.1.6.9.29 PLIB_ADC_ExistsPairInterruptRequest Function

C

```
bool PLIB_ADC_ExistsPairInterruptRequest(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the PairInterruptRequest feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_PairInterruptRequestEnable](#)
- [PLIB_ADC_PairInterruptRequestDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PairInterruptRequest feature is supported on the device
- false = The PairInterruptRequest feature is not supported on the device

Remarks

None.

5.6.1.6.9.30 PLIB_ADC_ExistsPairSampleStatus Function**C**

```
bool PLIB_ADC_ExistsPairSampleStatus(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the PairSampleStatus feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_PairSampleStatusClear](#)
- [PLIB_ADC_PairSamplesAvailable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PairSampleStatus feature is supported on the device
- false = The PairSampleStatus feature is not supported on the device

Remarks

None.

5.6.1.6.9.31 PLIB_ADC_ExistsPairTriggerSource Function**C**

```
bool PLIB_ADC_ExistsPairTriggerSource(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the PairTriggerSource feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_PairTriggerSourceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PairTriggerSource feature is supported on the device

- false = The PairTriggerSource feature is not supported on the device

Remarks

None.

5.6.1.6.9.32 PLIB_ADC_ExistsResultBufferFillStatus Function**C**

```
bool PLIB_ADC_ExistsResultBufferFillStatus(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ResultBufferFillStatus feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ResultBufferStatusGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ResultBufferFillStatus feature is supported on the device
- false = The ResultBufferFillStatus feature is not supported on the device

Remarks

None.

5.6.1.6.9.33 PLIB_ADC_ExistsResultBufferMode Function**C**

```
bool PLIB_ADC_ExistsResultBufferMode(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ResultBufferMode feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ResultBufferModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ResultBufferMode feature is supported on the device
- false = The ResultBufferMode feature is not supported on the device

Remarks

None.

5.6.1.6.9.34 PLIB_ADC_ExistsResultFormat Function**C**

```
bool PLIB_ADC_ExistsResultFormat(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ResultFormat feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ResultFormatSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ResultFormat feature is supported on the device
- false = The ResultFormat feature is not supported on the device

Remarks

None.

5.6.1.6.9.35 PLIB_ADC_ExistsResultGet Function**C**

```
bool PLIB_ADC_ExistsResultGet(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ResultGet feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ResultGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ResultGet feature is supported on the device
- false = The ResultGet feature is not supported on the device

Remarks

None.

5.6.1.6.9.36 PLIB_ADC_ExistsResultGetByIndex Function**C**

```
bool PLIB_ADC_ExistsResultGetByIndex(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ResultGetByIndex feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ResultGetByIndex](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ResultGetByIndex feature is supported on the device
- false = The ResultGetByIndex feature is not supported on the device

Remarks

None.

5.6.1.6.9.37 PLIB_ADC_ExistsResultSign Function**C**

```
bool PLIB_ADC_ExistsResultSign(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ResultSign feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ResultSignGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ResultSign feature is supported on the device
- false = The ResultSign feature is not supported on the device

Remarks

None.

5.6.1.6.9.38 PLIB_ADC_ExistsResultSize Function**C**

```
bool PLIB_ADC_ExistsResultSize(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the ResultSize feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_ResultSizeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ResultSize feature is supported on the device
- false = The ResultSize feature is not supported on the device

Remarks

None.

5.6.1.6.9.39 PLIB_ADC_ExistsSampleResolution Function**C**

```
bool PLIB_ADC_ExistsSampleResolution(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the SampleResolution feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_SampleMaxGet](#)
- [PLIB_ADC_SampleMinGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The SampleResolution feature is supported on the device

- false = The SampleResolution feature is not supported on the device

Remarks

None.

5.6.1.6.9.40 PLIB_ADC_ExistsSamplesPerInterruptSelect Function**C**

```
bool PLIB_ADC_ExistsSamplesPerInterruptSelect(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the SamplesPerInterruptSelect feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_SamplesPerInterruptSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The SamplesPerInterruptSelect feature is supported on the device
- false = The SamplesPerInterruptSelect feature is not supported on the device

Remarks

None.

5.6.1.6.9.41 PLIB_ADC_ExistsSamplingAcquisitionTime Function**C**

```
bool PLIB_ADC_ExistsSamplingAcquisitionTime(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the SamplingAcquisitionTime feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_SampleAcquisitionTimeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The SamplingAcquisitionTime feature is supported on the device
- false = The SamplingAcquisitionTime feature is not supported on the device

Remarks

None.

5.6.1.6.9.42 PLIB_ADC_ExistsSamplingAutoStart Function**C**

```
bool PLIB_ADC_ExistsSamplingAutoStart(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the SamplingAutoStart feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_SampleAutoStartEnable](#)
- [PLIB_ADC_SampleAutoStartDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The SamplingAutoStart feature is supported on the device
- false = The SamplingAutoStart feature is not supported on the device

Remarks

None.

5.6.1.6.9.43 PLIB_ADC_ExistsSamplingControl Function**C**

```
bool PLIB_ADC_ExistsSamplingControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the SamplingControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_SamplingStart](#)
- [PLIB_ADC_SamplingStop](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The SamplingControl feature is supported on the device
- false = The SamplingControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.44 PLIB_ADC_ExistsSamplingModeControl Function**C**

```
bool PLIB_ADC_ExistsSamplingModeControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the SamplingModeControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_SamplingModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The SamplingModeControl feature is supported on the device
- false = The SamplingModeControl feature is not supported on the device

Remarks

None.

5.6.1.6.9.45 PLIB_ADC_ExistsSamplingStatus Function**C**

```
bool PLIB_ADC_ExistsSamplingStatus(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the SamplingStatus feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_SamplingIsActive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The SamplingStatus feature is supported on the device
- false = The SamplingStatus feature is not supported on the device

Remarks

None.

5.6.1.6.9.46 PLIB_ADC_ExistsStopInIdleControl Function**C**

```
bool PLIB_ADC_ExistsStopInIdleControl(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_StopInIdleEnable](#)
- [PLIB_ADC_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The StopInIdle feature is supported on the device
- false = The StopInIdle feature is not supported on the device

Remarks

None.

5.6.1.6.9.47 PLIB_ADC_ExistsVoltageReference Function**C**

```
bool PLIB_ADC_ExistsVoltageReference(  
    ADC_MODULE_ID index  
);
```

Description

This function identifies whether the VoltageReference feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- [PLIB_ADC_VoltageReferenceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The VoltageReference feature is supported on the device
- false = The VoltageReference feature is not supported on the device

Remarks

None.

5.6.1.6.10 Mux Selection and Channel Scan

5.6.1.6.10.1 PLIB_ADC_MuxAInputScanDisable Function

C

```
void PLIB_ADC_MuxAInputScanDisable(  
    ADC_MODULE_ID index  
);
```

Description

This function disables scan input for CH0+ of MUX A.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsMuxInputScanControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_MuxAInputScanDisable(MY_ADC_INSTANCE);
```

5.6.1.6.10.2 PLIB_ADC_MuxAInputScanEnable Function

C

```
void PLIB_ADC_MuxAInputScanEnable(  
    ADC_MODULE_ID index  
);
```

Description

This function enables scan input for CH0+ of MUX A.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsMuxInputScanControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_MuxAInputScanEnable(MY_ADC_INSTANCE);
```

5.6.1.6.10.3 PLIB_ADC_MuxChannel0InputNegativeSelect Function**C**

```
void PLIB_ADC_MuxChannel0InputNegativeSelect(
    ADC_MODULE_ID index,
    ADC_MUX muxType,
    ADC_INPUTS_NEGATIVE input
);
```

Description

This function selects the negative input for channel 0 of MUX A or MUX B.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
muxType	One of the possible values from ADC_MUX

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsMuxChannel0NegativeInput](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_MuxChannel0InputNegativeSelect(MY_ADC_INSTANCE, ADC_MUX_A,
ADC_INPUT_NEGATIVE_VREF_MINUS);
```


5.6.1.6.10.4 PLIB_ADC_MuxChannel0InputPositiveSelect Function

C

```
void PLIB_ADC_MuxChannel0InputPositiveSelect(  
    ADC_MODULE_ID index,  
    ADC_MUX muxType,  
    ADC_INPUTS_POSITIVE input  
);
```

Description

This function selects the positive input for channel 0 of MUX A or MUX B.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
muxType	One of the possible values from ADC_MUX

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsMuxChannel0PositiveInput](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_MuxChannel0InputPositiveSelect(MY_ADC_INSTANCE, ADC_MUX_A, ADC_INPUT_POSITIVE_AN2);
```

5.6.1.6.10.5 PLIB_ADC_MuxChannel123InputNegativeSelect Function

C

```
void PLIB_ADC_MuxChannel123InputNegativeSelect(  
    ADC_MODULE_ID index,  
    ADC_MUX muxType,  
    ADC_CHANNEL123_INPUTS_NEG input  
);
```

Description

This function selects the negative input for channels 1, 2, and 3 of MUX A or MUX B.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
muxType	One of the possible values from ADC_MUX

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsMuxChannel123NegativeInput](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_MuxChannel123InputNegativeSelect(MY_ADC_INSTANCE, ADC_MUX_A,  
ADC_CHANNEL123_INPUT_NEG_CH123_VREF_MINUS);
```

5.6.1.6.10.6 PLIB_ADC_MuxChannel123InputPositiveSelect Function**C**

```
void PLIB_ADC_MuxChannel123InputPositiveSelect(  
    ADC_MODULE_ID index,  
    ADC_MUX muxType,  
    ADC_CHANNEL123_INPUTS_POS input  
);
```

Description

This function selects the positive input for channels 1, 2, and 3 of MUX A or MUX B.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
muxType	One of the possible values from ADC_MUX

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_ADC_ExistsMuxChannel123PositiveInput](#) in your application to determine whether this feature is available.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the  
// application developer.  
PLIB_ADC_MuxChannel123InputPositiveSelect(MY_ADC_INSTANCE, ADC_MUX_A,  
ADC_CHANNEL123_INPUT_POS_CH1_AN0_CH2_AN1_CH3_AN2);
```

5.6.1.7 Files**Files**

Name	Description
plib_adc.h	ADC PLIB interface header for definitions common to the ADC peripheral.

Description

5.6.1.7.1 plib_adc.h

ADC Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the ADC PLIB for all families of Microchip microcontrollers. The definitions in this file are common to the ADC peripheral.

Functions

	Name	Description
≡	<u>PLIB_ADC_AsynchronousDedicatedSamplingDisable</u>	Disables asynchronous dedicated S&H sampling.
≡	<u>PLIB_ADC_AsynchronousDedicatedSamplingEnable</u>	Enables asynchronous dedicated S&H sampling.
≡	<u>PLIB_ADC_CalibrationDisable</u>	Normal ADC module operation (no calibration is performed).
≡	<u>PLIB_ADC_CalibrationEnable</u>	Calibration is performed on the next ADC conversion.
≡	<u>PLIB_ADC_ChannelGroupSelect</u>	Selects channels to be utilized.
≡	<u>PLIB_ADC_ConversionClockGet</u>	Obtains the conversion clock.
≡	<u>PLIB_ADC_ConversionClockSet</u>	Sets the ADC module conversion clock.
≡	<u>PLIB_ADC_ConversionClockSourceSelect</u>	Selects the ADC module conversion clock source.
≡	<u>PLIB_ADC_ConversionHasCompleted</u>	Provides the conversion completion status of the ADC.
≡	<u>PLIB_ADC_ConversionOrderSelect</u>	Selects the conversion order.
≡	<u>PLIB_ADC_ConversionStart</u>	Starts ADC module manual conversion process.
≡	<u>PLIB_ADC_ConversionStopSequenceDisable</u>	Normal conversion sequence.
≡	<u>PLIB_ADC_ConversionStopSequenceEnable</u>	Stop conversion sequence (when the first ADC module interrupt is generated).
≡	<u>PLIB_ADC_ConversionTriggerGroupSelect</u>	Selects the conversion trigger source group.
≡	<u>PLIB_ADC_ConversionTriggerSourceSelect</u>	Selects the conversion trigger source.
≡	<u>PLIB_ADC_Disable</u>	ADC module is disabled (turned OFF).
≡	<u>PLIB_ADC_DMAAddressIncrementSelect</u>	Selects the increment rate for the DMA address.
≡	<u>PLIB_ADC_DMABufferModeSelect</u>	DMA Buffer Build mode.
≡	<u>PLIB_ADC_DMADisable</u>	Disables the ADC Direct Memory Access (DMA).
≡	<u>PLIB_ADC_DMAEnable</u>	Enables the ADC Direct Memory Access (DMA).
≡	<u>PLIB_ADC_DMAInputBufferSelect</u>	Selects the number of DMA buffer locations per analog input.
≡	<u>PLIB_ADC_Enable</u>	ADC module is enabled (turned ON).
≡	<u>PLIB_ADC_ExistsAsynchronousDedicatedSampling</u>	Identifies whether the AsynchronousDedicatedSampling feature exists on the ADC module
≡	<u>PLIB_ADC_ExistsCalibrationControl</u>	Identifies whether the CalibrationControl feature exists on the ADC module
≡	<u>PLIB_ADC_ExistsChannelGroup</u>	Identifies whether the ChannelGroup feature exists on the ADC module
≡	<u>PLIB_ADC_ExistsConversionClock</u>	Identifies whether the ConversionClock feature exists on the ADC module
≡	<u>PLIB_ADC_ExistsConversionClockSource</u>	Identifies whether the ConversionClockSource feature exists on the ADC module
≡	<u>PLIB_ADC_ExistsConversionControl</u>	Identifies whether the ConversionControl feature exists on the ADC module
≡	<u>PLIB_ADC_ExistsConversionOrder</u>	Identifies whether the ConversionOrder feature exists on the ADC module
≡	<u>PLIB_ADC_ExistsConversionStatus</u>	Identifies whether the ConversionStatus feature exists on the ADC module

	PLIB_ADC_ExistsConversionStopSequenceControl	Identifies whether the ConversionStopSequenceControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionTriggerGroup	Identifies whether the ConversionTriggerGroup feature exists on the ADC module
	PLIB_ADC_ExistsConversionTriggerSource	Identifies whether the ConversionTriggerSource feature exists on the ADC module
	PLIB_ADC_ExistsDMAAddressIncrement	Identifies whether the DMAAddressIncrement feature exists on the ADC module
	PLIB_ADC_ExistsDMABufferMode	Identifies whether the DMABufferMode feature exists on the ADC module
	PLIB_ADC_ExistsDMABuffersPerAnalogInput	Identifies whether the DMABuffersPerInput feature exists on the ADC module
	PLIB_ADC_ExistsDMAControl	Identifies whether the DMAControl feature exists on the ADC module
	PLIB_ADC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADC module.
	PLIB_ADC_ExistsGlobalSoftwareTrigger	Identifies whether the GlobalSoftwareTrigger feature exists on the ADC module
	PLIB_ADC_ExistsInputSelect	Identifies whether the InputSelect feature exists on the ADC module
	PLIB_ADC_ExistsInternalReferenceChannelControl	Identifies whether the InternalReferenceChannelControl feature exists on the ADC module
	PLIB_ADC_ExistsISRJumpTableBaseAddress	Identifies whether the ISRJumpTableBaseAddress feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0NegativeInput	Identifies whether the MuxChannel0NegativeInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0PositiveInput	Identifies whether the MuxChannel0PositiveInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel123NegativeInput	Identifies whether the MuxChannel123NegativeInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel123PositiveInput	Identifies whether the MuxChannel123PositiveInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanControl	Identifies whether the MuxInputScanControl feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanSelect	Identifies whether the MuxInputScanSelect feature exists on the ADC module
	PLIB_ADC_ExistsPairConversionControl	Identifies whether the PairConversionControl feature exists on the ADC module
	PLIB_ADC_ExistsPairInterruptOnConversion	Identifies whether the PairInterruptOnConversion feature exists on the ADC module
	PLIB_ADC_ExistsPairInterruptRequest	Identifies whether the PairInterruptRequest feature exists on the ADC module
	PLIB_ADC_ExistsPairSampleStatus	Identifies whether the PairSampleStatus feature exists on the ADC module
	PLIB_ADC_ExistsPairTriggerSource	Identifies whether the PairTriggerSource feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferFillStatus	Identifies whether the ResultBufferFillStatus feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferMode	Identifies whether the ResultBufferMode feature exists on the ADC module
	PLIB_ADC_ExistsResultFormat	Identifies whether the ResultFormat feature exists on the ADC module

	PLIB_ADC_ExistsResultGet	Identifies whether the ResultGet feature exists on the ADC module
	PLIB_ADC_ExistsResultGetByIndex	Identifies whether the ResultGetByIndex feature exists on the ADC module
	PLIB_ADC_ExistsResultSign	Identifies whether the ResultSign feature exists on the ADC module
	PLIB_ADC_ExistsResultSize	Identifies whether the ResultSize feature exists on the ADC module
	PLIB_ADC_ExistsSampleResolution	Identifies whether the SampleResolution feature exists on the ADC module
	PLIB_ADC_ExistsSamplesPerInterruptSelect	Identifies whether the SamplesPerInterruptSelect feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAcquisitionTime	Identifies whether the SamplingAcquisitionTime feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAutoStart	Identifies whether the SamplingAutoStart feature exists on the ADC module
	PLIB_ADC_ExistsSamplingControl	Identifies whether the SamplingControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingModeControl	Identifies whether the SamplingModeControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingStatus	Identifies whether the SamplingStatus feature exists on the ADC module
	PLIB_ADC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the ADC module
	PLIB_ADC_ExistsVoltageReference	Identifies whether the VoltageReference feature exists on the ADC module
	PLIB_ADC_GlobalSoftwareTriggerSet	Global software trigger enable/set control.
	PLIB_ADC_InputScanMaskAdd	Select ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemove	Omits ADC analog channel for input scan.
	PLIB_ADC_InputSelectNegative	Analog negative channel selection.
	PLIB_ADC_InputSelectPositive	Positive analog input channel selection.
	PLIB_ADC_InternalReferenceChannelDisable	Internal reference input is disabled.
	PLIB_ADC_InternalReferenceChannelEnable	Internal reference input is enabled.
	PLIB_ADC_IsrJumpTableBaseAddressGet	Gets the base address of the user's ADC Interrupt Service Routine (ISR) jump table.
	PLIB_ADC_IsrJumpTableBaseAddressSet	Sets the base address of the user's ADC Interrupt Service Routine (ISR) jump table.
	PLIB_ADC_MuxAInputScanDisable	Do not scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxAInputScanEnable	Scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxChannel0InputNegativeSelect	Channel 0 negative input select for multiplexer setting.
	PLIB_ADC_MuxChannel0InputPositiveSelect	Channel 0 positive input select for multiplexer setting.
	PLIB_ADC_MuxChannel123InputNegativeSelect	Channel 1, 2, and 3 negative input select.
	PLIB_ADC_MuxChannel123InputPositiveSelect	Channel 1, 2, and 3 positive input select.
	PLIB_ADC_PairConversionIsPending	Informs whether conversion is pending or complete.
	PLIB_ADC_PairConversionStart	Start conversion of the requested ADC pair.
	PLIB_ADC_PairInterruptAfterFirstConversion	ADC pair conversion interrupt is generated after the first conversion.
	PLIB_ADC_PairInterruptAfterSecondConversion	ADC pair conversion interrupt is generated after the second conversion.

	PLIB_ADC_PairInterruptRequestDisable	Disable IRQ generation for the ADC pair.
	PLIB_ADC_PairInterruptRequestEnable	Enable IRQ generation for the ADC pair.
	PLIB_ADC_PairSamplesAvailable	Provides status of the ADC pair sample availability in the buffer.
	PLIB_ADC_PairSampleStatusClear	Clears the ADC pair ready status flag.
	PLIB_ADC_PairTriggerSourceSelect	Selects trigger source for the requested ADC pair.
	PLIB_ADC_ResultBufferModeSelect	Selects the result buffer mode.
	PLIB_ADC_ResultBufferStatusGet	Provides the buffer fill status.
	PLIB_ADC_ResultFormatSelect	Selects the result format.
	PLIB_ADC_ResultGet	Obtains the ADC sample/result value.
	PLIB_ADC_ResultGetByIndex	Provides the ADC conversion result based on the buffer index.
	PLIB_ADC_ResultSignGet	Obtains the ADC result sign.
	PLIB_ADC_ResultSizeSelect	Selects the ADC module mode/result size.
	PLIB_ADC_SampleAcquisitionTimeSet	Sets the ADC acquisition/auto-sample time in TADs.
	PLIB_ADC_SampleAutoStartDisable	Sampling auto-start is disabled.
	PLIB_ADC_SampleAutoStartEnable	Sampling auto-start is enabled.
	PLIB_ADC_SampleMaxGet	Provides the maximum sample value.
	PLIB_ADC_SampleMinGet	Provides the minimum sample value.
	PLIB_ADC_SamplesPerInterruptSelect	Interrupts at the completion of conversion for each nth sample.
	PLIB_ADC_SamplingIsActive	Provides the ADC sampling status.
	PLIB_ADC_SamplingModeSelect	Enable the selected sampling mode.
	PLIB_ADC_SamplingStart	Sampling is enabled.
	PLIB_ADC_SamplingStop	Holding is enabled.
	PLIB_ADC_StopInIdleDisable	Continue ADC module operation when the device is in Idle mode.
	PLIB_ADC_StopInIdleEnable	Discontinue ADC module operation when device enters Idle mode.
	PLIB_ADC_VoltageReferenceSelect	Voltage reference configuration.

Types

	Name	Description
	ADC_ACQUISITION_TIME	Data type defining the different ADC acquisition times by which the ADC can be configured.
	ADC_CONVERSION_CLOCK	Data type defines the different ADC Conversion clock
	ADC_SAMPLE	Data type defining the size of the adc sample register.

File Name

plib_adc.h

Company

Microchip Technology Inc.

5.6.2 ADCP Peripheral Library

This topic describes the Pipelined Analog-to-Digital Converter peripheral library interface.

Description

This section contains the list of topics.

5.6.2.1 Introduction

Pipelined Analog-to-Digital Converter Peripheral Library for Microchip Microcontrollers

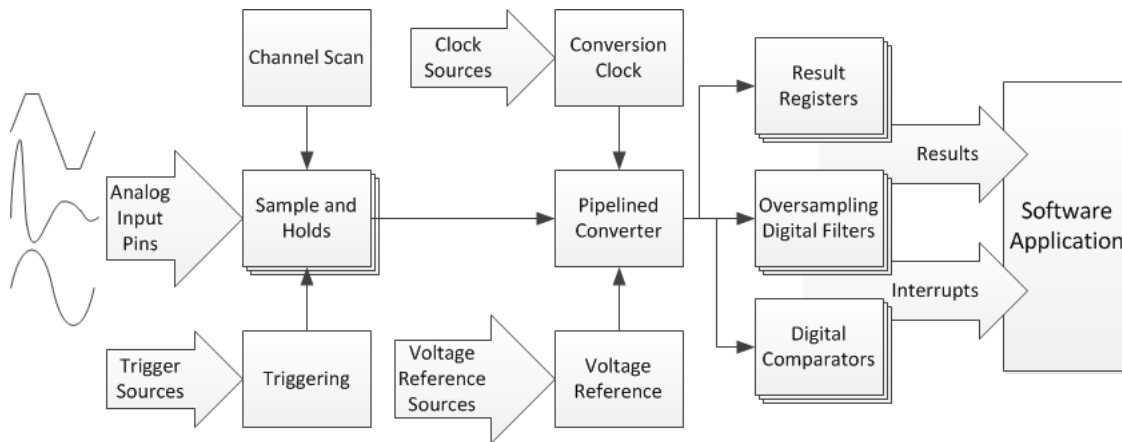
This library provides a low-level abstraction of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Pipelined Analog-to-Digital Converter (ADCP) peripheral is used to convert an analog input into a digital number that represents that input voltage. When the input is periodically converted, the result is a sequence of digital values that have converted a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal. This particular ADC architecture is pipelined allowing it to simultaneously convert up to 6 samples, each at a different stage of conversion resulting in a low overall latency of multiple sampled inputs.

The structure of the ADCP is shown in the diagram below.

1. Analog input pins [connect](#) to the sample and hold (S&H) circuits. These circuits capture the analog signal so that it can be passed to the pipeline converter.
2. The S&H circuits are controlled by predefined trigger sources. These trigger sources switch the sample and hold circuit to hold and queue the conversion of the sample.
3. Channel scan is available which allows a single trigger to start a sample/conversion sequence of multiple analog inputs using a predefined scan list.
4. A conversion clock is required for the pipeline converter. This clock determines the conversion speed and latency and effects power consumption.
5. A voltage reference is required for the pipeline converter. This voltage reference determines allowable input range of the analog signal.
6. Result registers store the conversion results and can be read by the software application. Software is notified of ready results by either polling or through the use of interrupts.
7. Optional Oversampling Digital Filters can be used to provide increased measurement accuracy at the sacrifice of sample rate. Software is notified of ready oversampling results by either polling or through the use of interrupts.
8. Optional digital comparators can be used to test conversion results independent of software and generate interrupts based on predefined conditions.



5.6.2.2 Release Notes

MPLAB Harmony Version: v0.70b ADCP Peripheral Library Version : 0.70b Release Date: 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.2.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.2.4 Using the Library

This topic describes the basic architecture of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library and provides information and examples on how to use it.

Interface Header File: `plib_adc.h`

The interface to the ADCP Peripheral library is defined in the "[plib_adcp.h](#)" header file. This file is included by the "peripheral.h" file. Any C language source (.c) file that uses the Pipelined Analog to Digital Converter (ADCP) Peripheral library should include "peripheral.h".

Library File:

The ADCP Peripheral library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the Peripheral interacts with the framework.

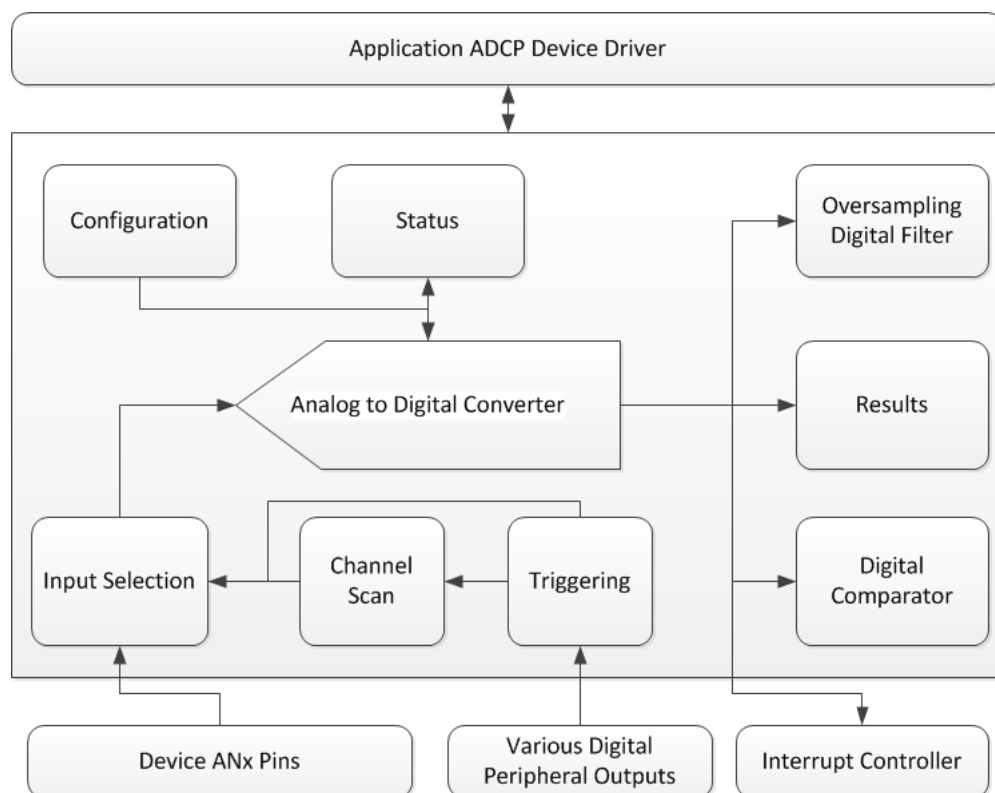
5.6.2.4.1 Hardware Abstraction Model

The interface provided is a super set of all the functionality of the available Pipelined Analog-to-Digital Converter (ADCP) on the device. Refer to the specific device data sheet or related family reference manual section to determine the set of functions that are supported for each ADCP module on the device.

This library provides the low-level abstraction of the ADCP module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Pipelined Analog-to-Digital Converter (ADCP) Software Abstraction Block Diagram



5.6.2.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the ADCP module.

Library Interface Section	Description
Interface Routines - Configuration	These library functions are used to configure the ADC, enable and disable it, initiate a calibration and manage low power states.
Interface Routines - Status	These library functions return status information related to the ADC.
Interface Routines - Input Selection	These library functions select and configure the ANx inputs to the ADC.
Interface Routines - Channel Scan	These library functions are used to configure the channel scan feature.
Interface Routines - Triggering	These library functions are used to configure and initiate conversion triggers.
Interface Routines - Analog Input Conversion Results	These library function are used to retrieve individual conversion results for ANx pins.
Interface Routines - Digital Comparator	These library functions are used to configure and query the digital comparators.
Interface Routines - Oversampling Digital Filter	These library functions are used to configure the Oversampling Digital Filter and fetch results from it.

5.6.2.4.3 How the Library Works

Core Functionality

Name	Description
Example - Simultaneous Sampling Three Class 1 Inputs	Simultaneous sampling is used when the application requires capture of more than one signal at the same instance of time. Simultaneous sampling requires the use of multiple Class 1 inputs where each is assigned the same trigger source. The following example illustrates simultaneous sampling of AN0, AN1 and AN2 using the global software trigger as the trigger source. The dedicated sample and holds (S&Hs) are configured for single-ended input and a unipolar (unsigned) output. No interrupts are used so the results are polled for ready status.
Example - Channel Scanning	Channel scanning is used in applications where precise timing of the sample to an external source is not needed. Channel scanning allows a large number of analog inputs to be sampled and converted in sequence each time a single trigger occurs. The following example illustrates how to configure channel scanning of multiple Class 2 and Class 3 inputs using the shared S&H. This example uses inputs AN5, AN11 - AN14. AN5 and AN11 are Class 2 inputs. AN13 and AN 14 are Class 3 inputs. The global software trigger is used to initiate the scan. A 4 TAD sample time... more

Other Functionality

Name	Description
Example - Digital Oversampling Filter	The Digital Oversampling filter can be used to increase resolution of the conversion result at the expense of throughput. The example below shows how two extra bits of resolution can be obtained using 16X oversampling (sixteen samples are used to create one higher resolution result). AN0 is used for the input. The sampling time for the retriggers is set to 3.5 TAD. Example:
Example - Digital Comparator	The Digital Comparator is used to automatically evaluate results as they are output by the converter. The following example illustrates an automated test of AN5 for values $\geq 80\%$ of full scale or $< 20\%$ of full scale. A count is incremented each time an event occurs. Example:

Description

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

5.6.2.4.3.1 Core Functionality**5.6.2.4.3.1.1 Example - Simultaneous Sampling Three Class 1 Inputs**

Simultaneous sampling is used when the application requires capture of more than one signal at the same instance of time. Simultaneous sampling requires the use of multiple Class 1 inputs where each is assigned the same trigger source. The following example illustrates simultaneous sampling of AN0, AN1 and AN2 using the global software trigger as the trigger source. The dedicated sample and holds (S&Hs) are configured for single-ended input and a unipolar (unsigned) output. No interrupts are used so the results are polled for ready status.

Example:

```
int32_t results[3] ; // storage for results

// This structure is used to simplify testing for a ready channel
AN_READY anReady ;

// Configure the ADC
// AVdd/AVss reference, ADC clock is system clock (SYSCLK) divided by 8
// No need to specify sample time for class 2/3 inputs or oversampling
// sample time as Class 2/3 inputs and oversampling is not used.
PLIB_ADCP_Configure ( ADCP_ID_1 ,
                      ADCP_VREF_AVDD_AVSS , // AVdd and AVss as reference
                      FALSE ,                // No VREF boost
                      FALSE ,                // Do not use fractional format
                      FALSE ,                // Do not stop in idle
                      ADCP_CLK_SRC_SYSCLK , // SYSCLK is the clock source
                      4 ,                    // TAD = 1/SYSCLK * 2 * 4
                                              // or ADC Clock = SYSCLK / (2 * 4)
                      0 ,                    // Oversampling is not used.
                      0 ,                    // No early interrupt.
                      0 );                  // No class 2 or 3 channels are used

// Set up the triggers
// Configure AN0 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure ( ADCP_ID_1 ,
                                    ADCP_CLASS12_AN0 ,
                                    ADCP_TRG_SRC_SOFTWARE ) ;

// Configure AN1 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure ( ADCP_ID_1 ,
                                    ADCP_CLASS12_AN1 ,
                                    ADCP_TRG_SRC_SOFTWARE ) ;

// Configure AN2 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure ( ADCP_ID_1 ,
                                    ADCP_CLASS12_AN2 ,
                                    ADCP_TRG_SRC_SOFTWARE ) ;

// Set S&H modes
PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH0 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;
PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH1 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;
PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH2 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;
// Enable the ADC Module
PLIB_ADCP_Enable ( ADCP_ID_1 ) ;

// Wait for a ready status
while ( !PLIB_ADCP_ModuleIsReady ( ADCP_ID_1 ) ) ;

while ( 1 )
```

```

{
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger ( ADCP_ID_1 ) ;

    // Wait for a data results to be ready
    // if lowest priority channel is done then all are done.
    do
    {
        anReady = PLIB_ADCP_ResultReady ( ADCP_ID_1 ) ;
    }
    while (anReady.AN2 == 0) ;

    // fetch AN0
    if (anReady.AN0 == 1)
        results[0] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN0 ) ;

    // fetch AN1
    if (anReady.AN1 == 1)
        results[1] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN1 ) ;

    // fetch AN2
    if (anReady.AN2 == 1)
        results[2] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN2 ) ;
}

```

5.6.2.4.3.1.2 Example - Channel Scanning

Channel scanning is used in applications where precise timing of the sample to an external source is not needed. Channel scanning allows a large number of analog inputs to be sampled and converted in sequence each time a single trigger occurs. The following example illustrates how to configure channel scanning of multiple Class 2 and Class 3 inputs using the shared S&H. This example uses inputs AN5, AN11 - AN14. AN5 and AN11 are Class 2 inputs. AN13 and AN 14 are Class 3 inputs. The global software trigger is used to initiate the scan. A 4 TAD sample time is specified for the shared S&H. The shared S&H is configured for single-ended operation and a unipolar output. No interrupts are used so the results are polled for ready status.

Example:

```

int32_t results[5] ; // storage for results

// This structure is used to simplify testing of the ready status
AN_READY anReady ;

// This structure is used to simplify specifying channels for scan
// AN5, AN11, AN12, AN13 and AN14 are selected for scanning.
AN_SELECT anSelect ;
anSelect.highWord = 0 ;
anSelect.lowWord = 0 ;
anSelect.AN5 = 1 ;
anSelect.AN11 = 1 ;
anSelect.AN12 = 1 ;
anSelect.AN13 = 1 ;
anSelect.AN14 = 1 ;

// Configure the ADC
// AVdd/AVss reference, ADC clock is system clock (SYSCLK) divided by 8
// The sample for the shared S&H must be adequate for all inputs.
// In this example it is set to 4 TAD.
// No need to specify sample time for oversampling as it is not used.
PLIB_ADCP_Configure ( ADCP_ID_1 ,
    ADCP_VREF_AVDD_AVSS , // AVdd and AVss as reference
    FALSE ,               // No VREF boost
    FALSE ,               // Do not use fractional format
    FALSE ,               // Do not stop in idle
    ADCP_CLK_SRC_SYSCLK , // SYSCLK is the clock source
    4 ,                   // TAD = 1/SYSCLK * 2 * 4
                        // or ADC Clock = SYSCLK / (2 * 4)

```

```

        0 ,           // Oversampling is not used.
        0 ,           // No early interrupt.
        3 ) ;         // 3 + 1 = 4 TAD for Class 2 and 3
                        // Sample Time.
// Specify the inputs to include in the Channel Scan using the selections made when
// initializing anSelect. Use the software trigger to initiate the scan.
PLIB_ADCP_ChannelScanConfigure ( ADCP_ID_1 ,
                                anSelect.lowWord , anSelect.highWord ,
                                ADCP_SCAN_TRG_SRC_SOFTWARE ) ;

// Set S&H 5 (shared S&H) to use the single ended, unipolar mode
PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH5 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;

// Enable the ADC Module
PLIB_ADCP_Enable ( ADCP_ID_1 ) ;

// Wait for a ready status
while ( !PLIB_ADCP_ModuleIsReady ( ADCP_ID_1 ) ) ;

while (1)
{
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger ( ADCP_ID_1 ) ;

    // Wait for a data results to be ready
    // if lowest priority channel is done then all are done.
    do
    {
        anReady = PLIB_ADCP_ResultReady ( ADCP_ID_1 ) ;
    }
    while (anReady.AN14 == 0) ;

    // fetch AN5
    if (anReady.AN5 == 1)
        results[0] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN5 ) ;

    // fetch AN11
    if (anReady.AN11 == 1)
        results[1] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN11 ) ;

    // fetch AN12
    if (anReady.AN12 == 1)
        results[2] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN12 ) ;

    // fetch AN13
    if (anReady.AN13 == 1)
        results[3] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN13 ) ;

    // fetch AN14
    if (anReady.AN14 == 1)
        results[4] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN14 ) ;
}

```

5.6.2.4.3.2 Other Functionality

5.6.2.4.3.2.1 Example - Digital Oversampling Filter

The Digital Oversampling filter can be used to increase resolution of the conversion result at the expense of throughput. The example below shows how two extra bits of resolution can be obtained using 16X oversampling (sixteen samples are used to create one higher resolution result). AN0 is used for the input. The sampling time for the retriggers is set to 3.5 TAD.

Example:

```
int32_t result; // storage for result
```

```

// Configure the ADC
// AVdd/AVss reference, ADC clock is system clock (SYSCLK) divided by 8
// No need to specify sample time for Class 2/3 inputs as they are not used.
// Oversampling sample time is set to 3.5 TAD.
PLIB_ADCP_Configure(ADCP_ID_1,
                    ADCP_VREF_AVDD_AVSS, // AVdd and AVss as reference
                    FALSE,                // No VREF boost
                    FALSE,                // Do not use fractional format
                    FALSE,                // Do not stop in idle
                    ADCP_CLK_SRC_SYSCLK,  // SYSCLK is the clock source
                    4,                    // TAD = 1/SYSCLK * 2 * 4
                    2,                    // or ADC Clock = SYSCLK / (2 * 4)
                    0,                    // 2 + 1.5 = 3.5 TAD between
                    0,                    // oversampling triggers
                    0);                  // No early interrupt.
// No Class 2 and 3 are used.

// Set up the triggers
// Configure AN0 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure(ADCP_ID_1, ADCP_CLASS12_AN0, ADCP_TRG_SRC_SOFTWARE);

// Set S&H modes
// Set S&H 0 to use the single ended, unipolar mode
PLIB_ADCP_SHModeSelect(ADCP_ID_1, ADCP_SH0, ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR);

// Configure the Oversampling Filter
PLIB_ADCP_OsampDigFilterConfig(ADCP_ID_1,          // ADCP module ID
                               ADCP_ODFLTR1,        // Filter ID
                               ADCP_AN0,            // Oversample AN0
                               ADCP_ODFLTR_16X,     // 16 x oversampling
                               FALSE);              // No Global Int Enable

// Enable ODFLTR0
PLIB_ADCP_OsampDigFilterEnable(ADCP_ID_1, ADCP_ODFLTR1);

// Enable the ADC Module
PLIB_ADCP_Enable(ADCP_ID_1);

// Wait for a ready status
while (!PLIB_ADCP_ModuleIsReady(ADCP_ID_1));

while (1) {
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger(ADCP_ID_1);

    // Wait for data to be ready
    while (PLIB_ADCP_OsampDigFilterDataRdy(ADCP_ID_1, ADCP_ODFLTR1) == FALSE);

    // get the result
    result = PLIB_ADCP_OsampDigFilterDataGet(ADCP_ID_1, ADCP_ODFLTR1);
}

```

5.6.2.4.3.2.2 Example - Digital Comparator

The Digital Comparator is used to automatically evaluate results as they are output by the converter. The following example illustrates an automated

test of AN5 for values $\geq 80\%$ of full scale or $< 20\%$ of full scale. A count is incremented each time an event occurs.

Example:

```

int32_t result; // storage for result
int32_t count = 0;

```

```

/* This structure is used to simplify testing for a ready channel */
AN_READY anReady;

// Configure the ADC
// AVdd/AVss reference, ADC clock is system clock (SYSCLK) divided by 8
// The sample for the shared S&H must be adequate for all inputs.
// In this example it is set to 4 TAD.
// No need to specify sample time for oversampling as it is not used.
PLIB_ADCP_Configure(ADCP_ID_1,
                    ADCP_VREF_AVDD_AVSS, // AVdd and AVss as reference
                    FALSE, // No VREF boost
                    FALSE, // Do not use fractional format
                    FALSE, // Do not stop in idle
                    ADCP_CLK_SRC_SYSCLK, // SYSCLK is the clock source
                    4, // TAD = 1/SYSCLK * 2 * 4 or
                    // ADC Clock = SYSCLK / (2 * 4)
                    0, // no oversampling
                    0, // No early interrupt.
                    3); // 3 + 1 = 4 TAD for Class 2 and 3 Sample Time.

// Set up the triggers
// Configure AN5 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure(ADCP_ID_1, ADCP_CLASS12_AN5, ADCP_TRG_SRC_SOFTWARE);

// Set S&H modes
// Set S&H 5 to use the single ended, unipolar mode
PLIB_ADCP_SHModeSelect(ADCP_ID_1, ADCP_SH5, ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR);

// Configure the Digital Comparator
// Creates an event when the reading of AN5 is less than 20% or
// greater than or equal to 80% of the full scale 12-bit output.
PLIB_ADCP_DigCmpConfig(ADCP_ID_1, // ADCP module ID
                       ADCP_DCMP1, // Comparator ID
                       FALSE, // No Global Int Enable
                       FALSE, // no test for between low and high
                       TRUE, // test for greater than or equal to high
                       FALSE, // no test for less than high
                       FALSE, // no test for greater than or equal to low
                       TRUE, // test for less than low
                       1 << 5, // enable AN5 for comparison
                       0xFFF - (0xFFF / 5), // high limit is 80% of full scale
                       0xFFF / 5); // low limit is 20% of full scale

// Enable DigCmp1
PLIB_ADCP_DigCmpEnable(ADCP_ID_1, ADCP_DCMP1);

// Enable the ADC Module
PLIB_ADCP_Enable(ADCP_ID_1);

// Wait for a ready status
while (!PLIB_ADCP_ModuleIsReady(ADCP_ID_1));

while (1) {
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger(ADCP_ID_1);

    // wait for conversion to complete
    do {
        anReady = PLIB_ADCP_ResultReady(ADCP_ID_1);
    } while (anReady.AN5 == 0);

    // Check result and increment count if an out of range event
    // occurred on digital comparator 1 for AN5
    if (PLIB_ADCP_DigCmpAidGet(ADCP_ID_1, ADCP_DCMP1) == 5) {
        count++;
    }
}

```

```

    }
}

```

5.6.2.4.4 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.2.5 Library Interface

Data Types

	Name	Description
	ADCP_MODULE_ID	Identifies the number of ADC Modules Supported.
	ADCP_VREF_SOURCE	Data type defines the ADCP VREF Source Select
	ADCP_CLOCK_SOURCE	Data type defines the ADCP Clock Source Select
	ADCP_CLASS12_INPUT_ID	Type for identifying the Class 1 and Class 2 ADC Inputs
	ADCP_SH_ID	Identifies the S&Hs supported regardless of type.
	ADCP_DSH_ID	Identifies the Dedicated S&Hs supported
	ADCP_SH_MODE	Data type defines the available modes for the S&H
	ADCP_INPUT_ID	Type for identifying the available ADC Inputs
	ADCP_DCMP_ID	Identifies the Digital Comparators supported.
	ADCP_ODFLTR_ID	Identifies the Oversampling Digital Filter supported.
	ADCP_ODFLTR_OSR	Identifies the Oversampling Digital Filter oversampling ratios supported.
	ADCP_SCAN_TRG_SRC	Data type defines the ADCP Channel Scan Trigger Source Selections
	ADCP_TRG_SRC	Data type defines the ADCP Trigger Source Selections
	AN_SELECT	This union of structures is provided to simply selection of analog inputs for inclusion in a particular function.
	AN_READY	This union (identical to the AN_SELECT union) is used as the return value by the <code>PLIB_ADCP_Result_Ready()</code> function.

Interface Routines - Channel Scan

	Name	Description
	PLIB_ADCP_ChannelScanConfigure	Selects input to include in Channel Scan Mode
	PLIB_ADCP_ExistsChannelScan	Identifies whether the ChannelScan feature exists on the ADCP module

Interface Routines - Configuration

	Name	Description
	PLIB_ADCP_Configure	Configures the Pipelined ADC converter including the ADC calibration registers.
	PLIB_ADCP_Enable	Enables the Pipelined ADC module
	PLIB_ADCP_Disable	Pipelined ADC module is disabled (turned OFF).
	PLIB_ADCP_CalibrationStart	Initiates Pipelined ADC Calibration.
	PLIB_ADCP_LowPowerStateEnter	Places the Pipelined ADC module in a low power state.
	PLIB_ADCP_LowPowerStateExit	Takes the Pipelined ADC module out of low power state and puts it into an operational mode.
	PLIB_ADCP_LowPowerStateGet	Returns the state of the low power setting.
	PLIB_ADCP_ExistsCalibration	Identifies whether the Calibration feature exists on the ADCP module
	PLIB_ADCP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCP module

	PLIB_ADCP_ExistsLowPowerControl	Identifies whether the LowPowerControl feature exists on the ADCP module
	PLIB_ADCP_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCP module

Interface Routines - Digital Comparator

	Name	Description
	PLIB_ADCP_DigCmpEnable	Enables the Digital Comparator in the Pipelined ADC module
	PLIB_ADCP_DigCmpDisable	Disables the Digital Comparator in the Pipelined ADC module
	PLIB_ADCP_DigCmpAldGet	Returns the Analog Input ID of for the Comparator Event
	PLIB_ADCP_DigCmpConfig	Configures the Digital Comparator on the Pipelined ADC converter.
	PLIB_ADCP_ExistsDigCmp	Identifies whether the DigitalComparator feature exists on the ADCP module

Interface Routines - Individual Analog Input Conversion Results

	Name	Description
	PLIB_ADCP_ResultReady	Returns the ADC conversion result ready bits for the module.
	PLIB_ADCP_ResultGet	Returns a Pipelined ADC conversion result.
	PLIB_ADCP_ResultReadyGrpIntConfigure	Selects input to include in global interrupt.
	PLIB_ADCP_ExistsConversionResults	Identifies whether the ConversionResults feature exists on the ADCP module

Interface Routines - Input Selection

	Name	Description
	PLIB_ADCP_AlternateInputSelect	Selects the alternate physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_DefaultInputSelect	Selects the default physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_ExistsInputSelect	Identifies whether the InputSelect feature exists on the ADCP module
	PLIB_ADCP_ExistsModeSelect	Identifies whether the ModeSelect feature exists on the ADCP module
	PLIB_ADCP_SHModeSelect	Selects the mode for the specified S&H.

Interface Routines - Oversampling Digital Filter

	Name	Description
	PLIB_ADCP_OsampDigFilterEnable	Enables the Oversampling Digital Filter in the Pipelined ADC module
	PLIB_ADCP_OsampDigFilterDisable	Disables the Oversampling Digital Filter in the Pipelined ADC module
	PLIB_ADCP_OsampDigFilterDataRdy	Used to determine if the Oversampling Digital Filter has data ready.
	PLIB_ADCP_OsampDigFilterDataGet	Used to fetch the data result from the Oversampling Digital Filter.
	PLIB_ADCP_ExistsOsampDigFilter	Identifies whether the OsampDigitalFilter feature exists on the ADCP module
	PLIB_ADCP_OsampDigFilterConfig	Configures the Oversampling Digital Filter on the Pipelined ADC converter.

Interface Routines - Status

	Name	Description
	PLIB_ADCP_ModuleIsReady	Returns the overall ready status of the module.
	PLIB_ADCP_ExistsReadyStatus	Identifies whether the ReadyStatus feature exists on the ADCP module

Interface Routines - Triggering

	Name	Description
	PLIB_ADCP_GlobalSoftwareTrigger	Initiate a Global Software Trigger on the specified module.
	PLIB_ADCP_IndividualTrigger	Triggers an individual channel independent of the configured trigger source
	PLIB_ADCP_Class12TriggerConfigure	Configures a Class 1 or Class 2 Trigger Source.
	PLIB_ADCP_ExistsTriggering	Identifies whether the Triggering feature exists on the ADCP module

Description

This section describes the Application Programming Interface (API) functions of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral.

Refer to each section for a detailed description.

5.6.2.5.1 Interface Routines - Configuration**5.6.2.5.1.1 PLIB_ADCP_Configure Function**

C

```
void PLIB_ADCP_Configure(
    ADCP_MODULE_ID index,
    ADCP_VREF_SOURCE voltageRefSelect,
    bool boostVref,
    bool fractionalOutputOn,
    bool stopInIdle,
    ADCP_CLOCK_SOURCE adcClockSource,
    int8_t adcClockDivider,
    int8_t oversampleDigFilterSamTime,
    int8_t earlyIntEnable,
    int8_t class2and3SampleTime
);
```

Description

Configures all ADC parameters common to all inputs. This configuration must occur prior to enabling the ADC and therefore must be called when the ADC is disabled.

Preconditions

The module is disabled when calling this function.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
voltageRefSelect	VREF Source Selection
boostVref	Enables the VREF boost if TRUE.
Use when VREFH	$VREFL < .64 (AVDD - AVSS)$
fractionalOutputOn	sets output to a fractional format if TRUE
stopInIdle	sets ADC to stop when device is in idle mode if TRUE
adcClockSource	Clock source selection
adcClockDivider	Clock source divider. Values range from 0 to 254.
oversampleDigFilterSamTime	Sets the delay (sample time) between automatically generated oversampling filter triggers. Values range from 0 to 31 for 1.5 TAD to 32.5 TAD respectively.
earlyIntEnable	Sets the number of clocks prior to the actual arrival of data that the ADRDY bit is set. Range is 0 to 7. Used to generate an early interrupt.
class2and3SampleTime	Set the sample time for Class 2 and Class 3 inputs. Range is 0 to 255 for a sample time of 1 to 256 respectively.

Returns

none

Remarks

This function must be called when the ADC is disabled.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
```

```
// Configure the ADC
PLIB_ADCP_Configure( MY_ADCP_INSTANCE,
    ADCP_VREF_VREFP_VREFN, // VREF+ and VREF- as reference
    FALSE, // No VREF boost
    TRUE, // Use fractionl format
    TRUE, // Do stop in idle
    ADCP_CLK_SRC_SYSCLK, // SYSCLK is the clock source
    3, // TAD = 1/SYSCLK * 2 * 3
    // or ADC Clock = SYSCLK / (2 * 3)
    2, // 2 + 1.5 = 3.5 TAD between
    // oversampling triggers
    0, // No early interrupt.
    3 ); // 3 + 1 = 4 TAD for Class 2 and 3
    // Sample Time.
```

```
// Enable the ADC
PLIB_ADCP_Enable(MY_ADCP_INSTANCE);
```

5.6.2.5.1.2 PLIB_ADCP_Enable Function**C**

```
void PLIB_ADCP_Enable(
    ADCP_MODULE_ID index
);
```

Description

This function enables (turns ON) the selected Pipelined ADC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance to be configured

Returns

None.

Remarks

None

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
PLIB_ADCP_Enable(MY_ADCP_INSTANCE);
```

5.6.2.5.1.3 PLIB_ADCP_Disable Function**C**

```
void PLIB_ADCP_Disable(
    ADCP_MODULE_ID index
```

```
);
```

Description

This function disables (turns OFF) the selected Pipelined ADC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance to be configured

Returns

None.

Remarks

Not all functionality is available on all devices. Please refer to the specific device data sheet for the list of available features.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the  
// application developer.  
PLIB_ADCP_Disable(MY_ADCP_INSTANCE);
```

5.6.2.5.1.4 PLIB_ADCP_CalibrationStart Function**C**

```
void PLIB_ADCP_CalibrationStart(  
    ADCP_MODULE_ID index  
);
```

Description

This function initiates calibration of the module. During calibration the module cannot perform conversion.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance to be configured

Returns

None.

Remarks

Calibration is complete when [PLIB_ADCP_ModuleIsReady\(\)](#) returns a TRUE result.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the  
// application developer.  
PLIB_ADCP_CalibrationStart(MY_ADCP_INSTANCE);  
while (!PLIB_ADCP_ModuleIsReady(MY_ADCP_INSTANCE));
```

5.6.2.5.1.5 PLIB_ADCP_LowPowerStateEnter Function**C**

```
void PLIB_ADCP_LowPowerStateEnter(  

```

```
    ADCP_MODULE_ID index
);
```

Description

Places the Pipelined ADC module in a low power state. The feature is used in lieu of disabling the ADC when power reduction is needed. The Pipelined ADC can come out of low power state and be operational much faster since recalibration is not required.

Preconditions

The Pipelined ADC must be enabled.

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Returns

None.

Remarks

None.

Example

```
PLIB_ADCP_LowPowerStateEnter(MY_ADCP_INSTANCE);
```

5.6.2.5.1.6 PLIB_ADCP_LowPowerStateExit Function**C**

```
void PLIB_ADCP_LowPowerStateExit(
    ADCP_MODULE_ID index
);
```

Description

Takes the Pipelined ADC module out of low power state and puts it into an operational mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Returns

None.

Remarks

The first five conversions following the exit from ADC Low-power mode may be subject to lower accuracy than specified in the device data sheet.

Example

```
PLIB_ADCP_LowPowerStateExit(MY_ADCP_INSTANCE);
```

5.6.2.5.1.7 PLIB_ADCP_LowPowerStateGet Function**C**

```
bool PLIB_ADCP_LowPowerStateGet(
    ADCP_MODULE_ID index
);
```

Description

Returns the state of the low power setting.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Returns

A Boolean indicating the state of the low power setting.

Remarks

None.

Example

```
bool lowPowerSate = PLIB_ADCP_LowPowerStateGet(MY_ADCP_INSTANCE);
```

5.6.2.5.1.8 PLIB_ADCP_ExistsCalibration Function**C**

```
bool PLIB_ADCP_ExistsCalibration(  
    ADCP_MODULE_ID index  
);
```

Description

This function identifies whether the Calibration feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_CalibrationStart](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the Calibration feature:

- true - When Calibration feature is supported on the device
- false - When Calibration feature is not supported on the device

Remarks

None.

5.6.2.5.1.9 PLIB_ADCP_ExistsEnableControl Function**C**

```
bool PLIB_ADCP_ExistsEnableControl(  
    ADCP_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_Enable](#)
- [PLIB_ADCP_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the EnableControl feature:

- true - When EnableControl feature is supported on the device
- false - When EnableControl feature is not supported on the device

Remarks

None.

5.6.2.5.1.10 PLIB_ADCP_ExistsLowPowerControl Function

C

```
bool PLIB_ADCP_ExistsLowPowerControl(  
    ADCP_MODULE_ID index  
);
```

Description

This function identifies whether the LowPowerControl feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_LowPowerStateEnter](#)
- [PLIB_ADCP_LowPowerStateExit](#)
- [PLIB_ADCP_LowPowerStateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the LowPowerControl feature:

- true - When LowPowerControl feature is supported on the device
- false - When LowPowerControl feature is not supported on the device

Remarks

None.

5.6.2.5.1.11 PLIB_ADCP_ExistsConfiguration Function

C

```
bool PLIB_ADCP_ExistsConfiguration(  
    ADCP_MODULE_ID index  
);
```

Description

This function identifies whether the Configuration feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_Configure](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the Configuration feature:

- true - When Configuration feature is supported on the device
- false - When Configuration feature is not supported on the device

Remarks

None.

5.6.2.5.2 Interface Routines - Status

5.6.2.5.2.1 PLIB_ADCP_ModuleIsReady Function

C

```
bool PLIB_ADCP_ModuleIsReady(  
    ADCP_MODULE_ID index  
);
```

Description

Returns the ready status of the Pipelined ADC. The ADC must be ready before operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Returns

boolean indicating ready status.

Remarks

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Enable the module
PLIB_ADCP_Enable(MY_ADCP_INSTANCE);

// wait for calibration to complete
while (!PLIB_ADCP_ModuleIsReady(MY_ADCP_INSTANCE));

// begin sampling
...
```

5.6.2.5.2 PLIB_ADCP_ExistsReadyStatus Function**C**

```
bool PLIB_ADCP_ExistsReadyStatus(
    ADCP_MODULE_ID index
);
```

Description

This function identifies whether the ReadyStatus feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_ModuleIsReady](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the ReadyStatus feature:

- true - When ReadyStatus feature is supported on the device
- false - When ReadyStatus feature is not supported on the device

Remarks

None.

5.6.2.5.3 Interface Routines - Input Selection**5.6.2.5.3.1 PLIB_ADCP_AlternateInputSelect Function****C**

```
void PLIB_ADCP_AlternateInputSelect(
    ADCP_MODULE_ID index,
    ADCP_DSH_ID id
);
```

Description

Selects the alternate physical input for the specified S&H.

Preconditions

The function only applies to dedicated S&Hs (Class 1 Inputs).

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	An ADCP_DSH_ID type indicating which dedicated Class 1 S&H to configure for its alternate input source.

Returns

none.

Remarks

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the  
// application developer.
```

```
// Set S&H 1 to use the alternate input.  
PLIB_ADCP_AlternateInputSelect(MY_ADCP_INSTANCE, ADCP_DSH1)
```

5.6.2.5.3.2 PLIB_ADCP_DefaultInputSelect Function**C**

```
void PLIB_ADCP_DefaultInputSelect(  
    ADCP_MODULE_ID index,  
    ADCP_DSH_ID id  
);
```

Description

Selects the default physical input for the specified S&H.

Preconditions

The function only applies to dedicated S&Hs (Class 1 Inputs).

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	An ADCP_DSH_ID type indicating which dedicated Class 1 S&H to configure for its alternate input source.

Returns

none.

Remarks

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the  
// application developer.
```

```
// Set S&H 1 to use the default input.  
PLIB_ADCP_DefaultInputSelect(MY_ADCP_INSTANCE, ADCP_DSH1)
```

5.6.2.5.3.3 PLIB_ADCP_ExistsInputSelect Function

C

```
bool PLIB_ADCP_ExistsInputSelect(  
    ADCP_MODULE_ID index  
);
```

Description

This function identifies whether the InputSelect feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_DedicatedSHInputSelect

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the InputSelect feature:

- true - When InputSelect feature is supported on the device
- false - When InputSelect feature is not supported on the device

Remarks

None.

5.6.2.5.3.4 PLIB_ADCP_ExistsModeSelect Function

C

```
bool PLIB_ADCP_ExistsModeSelect(  
    ADCP_MODULE_ID index  
);
```

Description

This function identifies whether the ModeSelect feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_SHModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the ModeSelect feature:

- true - When ModeSelect feature is supported on the device
- false - When ModeSelect feature is not supported on the device

Remarks

None.

5.6.2.5.3.5 PLIB_ADCP_SHModeSelect Function**C**

```
void PLIB_ADCP_SHModeSelect(
    ADCP_MODULE_ID index,
    ADCP_DSH_ID id,
    ADCP_SH_MODE mode
);
```

Description

Selects the mode (single ended or differential) and encoding (unipolar or two's compliment) for the specified S&H.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	An ADCP_DSH_ID type indicating which dedicated S&H to configure to use its alternate source.
mode	An ADCP_SH_MODE type indicating the mode selection.

Returns

none.

Remarks

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Set S&H 1 to use the single ended, two's complement mode
// selection.
PLIB_ADCP_SHModeSelect(MY_ADCP_INSTANCE,
    ADCP_SH1,
    ADCP_SH_MODE_SINGLE_ENDED_TWOS_COMP)
```

5.6.2.5.4 Interface Routines - Channel Scan**5.6.2.5.4.1 PLIB_ADCP_ChannelScanConfigure Function****C**

```
void PLIB_ADCP_ChannelScanConfigure(
    ADCP_MODULE_ID index,
    uint32_t lowEnable,
    uint32_t highEnable,
    ADCP_SCAN_TRG_SRC triggerSource
);
```

Description

Selects inputs, as determined by the low and high enable scan list for inclusion in the channel scan sequence. If the input is a

Class 1 or Class 2 type, it will also select the trigger source for that input to be the scan trigger, which is required if included in channel scanning.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
lowEnable	bit mask for selecting low order scan channels
highEnable	bit mask for selecting high order scan channels
triggerSource	Trigger source used to initiate channel scan.

Returns

none.

Remarks

The type def [AN_SELECT](#) can be used to create a variable to simplify selection of the inputs to include in the channel scan. This typedef creates bit field structures for each ANx input that are joined with unions for the low and high 32-bit words. See the code example for Channel scan in the ADCP PLIB help documentation for details on its use.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure Channel Scanning
// Channels 10 through 13
// Trigger on Timer 1 match.
PLIB_ADCP_ChannelScanConfigure(MY_ADCP_INSTANCE,
                                0XF000, // AN12 - AN15
                                0X0F00, // AN24 - AN27
                                ADCP_SCAN_TRG_SRC_TMR1_MATCH)
```

5.6.2.5.4.2 PLIB_ADCP_ExistsChannelScan Function

C

```
bool PLIB_ADCP_ExistsChannelScan(
    ADCP_MODULE_ID index
);
```

Description

This function identifies whether the ChannelScan feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_ChannelScanConfigure](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the ChannelScan feature:

- true - When ChannelScan feature is supported on the device
- false - When ChannelScan feature is not supported on the device

Remarks

None.

5.6.2.5.5 Interface Routines - Triggering

5.6.2.5.5.1 PLIB_ADCP_GlobalSoftwareTrigger Function

C

```
void PLIB_ADCP_GlobalSoftwareTrigger(  
    ADCP_MODULE_ID index  
);
```

Description

All inputs or scan list that is configured to trigger on the global software trigger are triggered.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Returns

None.

Remarks

None.

Example

```
PLIB_ADCP_GlobalSoftwareTrigger(MY_ADCP_INSTANCE);
```

5.6.2.5.5.2 PLIB_ADCP_IndividualTrigger Function

C

```
void PLIB_ADCP_IndividualTrigger(  
    ADCP_MODULE_ID index,  
    ADCP_INPUT_ID inputId  
);
```

Description

Forces a trigger of an individual Class 1 or Class 2 channel independent of its configured trigger source.

Preconditions

The function only applies to Class 1 and Class 2 inputs.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
inputId	An ADCP_INPUT_ID type indicating which input to trigger.

Returns

none.

Remarks

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Force a trigger of AN5
PLIB_ADCP_IndividualTrigger(MY_ADCP_INSTANCE, ADCP_CLASS12_AN5)
```

5.6.2.5.5.3 PLIB_ADCP_Class12TriggerConfigure Function**C**

```
void PLIB_ADCP_Class12TriggerConfigure(
    ADCP_MODULE_ID index,
    ADCP_CLASS12_INPUT_ID inputId,
    ADCP_TRG_SRC triggerSource
);
```

Description

Configures the trigger source for Class 1 or Class 2 inputs. A call to this function is not required when the input is being used as part of a channel scan as the channel scan configure function also configures all trigger sources.

Preconditions

The function only applies to Class 1 and Class 2 inputs.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
inputId	Class 1 or Class 2 input to configure the trigger for.
triggerSource	Trigger source to use for this input.

Returns

none.

Remarks

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure AN0 for triggering from INT0.
PLIB_ADCP_Class12TriggerConfigure(MY_ADCP_INSTANCE,
    ADCP_CLASS12_AN0,
    ADCP_TRG_SRC_INT0);
```

5.6.2.5.5.4 PLIB_ADCP_ExistsTriggering Function**C**

```
bool PLIB_ADCP_ExistsTriggering(
    ADCP_MODULE_ID index
);
```

Description

This function identifies whether the Triggering feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_Class12TriggerConfigure](#)
- [PLIB_ADCP_GlobalSoftwareTrigger](#)
- [PLIB_ADCP_IndividualTrigger](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the Triggering feature:

- true - When Triggering feature is supported on the device
- false - When Triggering feature is not supported on the device

Remarks

None.

5.6.2.5.6 Interface Routines - Individual Analog Input Conversion Results

5.6.2.5.6.1 PLIB_ADCP_ResultReady Function

C

```
AN_READY PLIB_ADCP_ResultReady(  
    ADCP_MODULE_ID index  
);
```

Description

Returns a result indicating which analog inputs have conversion results ready.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Returns

A [AN_READY](#) type indicating conversion result status.

Remarks

This function returns individual conversion results. It does not return results from the module.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the  
// application developer.
```



```
AN_READY      MyRdyStatus;

// check for data and process it
if ((MyRdyStatus = PLIB_ADCP_ResultReady(MY_ADCP_INSTANCE)) != 0) {
    // fetch the results and process
}
```

5.6.2.5.6.2 PLIB_ADCP_ResultGet Function

C

```
int32_t PLIB_ADCP_ResultGet(
    ADCP_MODULE_ID index,
    ADCP_INPUT_ID inputId
);
```

Description

Returns the specified Pipelined ADC analog input conversion result.

Preconditions

A valid conversion is ready to be fetched.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
inputId	Identifier for the input

Returns

The conversion result expressed as a 32 bit integer.

Remarks

None.

Example

```
int32_t      result;
...
// fetch result for AN31
result = PLIB_ADCP_ResultGet( MY_ADCP_INSTANCE, ADCP_AN31 );
```

5.6.2.5.6.3 PLIB_ADCP_ResultReadyGrpIntConfigure Function

C

```
void PLIB_ADCP_ResultReadyGrpIntConfigure(
    ADCP_MODULE_ID index,
    uint32_t lowEnable,
    uint32_t highEnable
);
```

Description

Selects inputs, as determined by the input mask channel scan list for inclusion in the global or global interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
lowEnable	bit mask for selecting low order scan channels

highEnable	bit mask for selecting high order scan channels

Returns

none.

Remarks

The type def [AN_SELECT](#) can be used to create a variable to simplify selection of the inputs to include in the global interrupt. This typedef creates bit field structures for each ANx input that are joined with unions for the low and high 32-bit words. See the code example for Channel scan in the ADCP PLIB help documentation for details on its use.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the  
// application developer.  
  
// Configure Global Interrupts  
// Analog inputs 10 through 13 are included in the global interrupt.  
PLIB_ADCP_ResultReadyGrpIntConfigure(MY_ADCP_INSTANCE,  
                                     0x0F00, // inputs AN8 - AN11  
                                     0x000F ); // inputs AN16 - AN19
```

5.6.2.5.6.4 PLIB_ADCP_ExistsConversionResults Function**C**

```
bool PLIB_ADCP_ExistsConversionResults(  
    ADCP_MODULE_ID index  
);
```

Description

This function identifies whether the ConversionResults feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_ResultReady](#)
- [PLIB_ADCP_ResultGet](#)
- [PLIB_ADCP_ResultReadyGrpIntConfigure](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the ConversionResults feature:

- true - When ConversionResults feature is supported on the device
- false - When ConversionResults feature is not supported on the device

Remarks

None.

5.6.2.5.7 Interface Routines - Digital Comparator

5.6.2.5.7.1 PLIB_ADCP_DigCmpEnable Function

C

```
void PLIB_ADCP_DigCmpEnable(  
    ADCP_MODULE_ID index,  
    ADCP_DCMP_ID id  
);
```

Description

This function enables (turns ON) the selected Digital Comparator in the specified Pipelined ADC module.

Preconditions

The digital comparator should be configured using the [PLIB_ADCP_Configure\(\)](#) function prior to enabling.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in the ADCP module.

Returns

None.

Remarks

None

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the  
// application developer.  
  
// Enable Digital Comparator 1  
PLIB_ADCP_DigCmpEnable(MY_ADCP_INSTANCE, ADCP_DCMP1);
```

5.6.2.5.7.2 PLIB_ADCP_DigCmpDisable Function

C

```
void PLIB_ADCP_DigCmpDisable(  
    ADCP_MODULE_ID index,  
    ADCP_DCMP_ID id  
);
```

Description

This function Disables (turns OFF) the selected Digital Comparator in the specified Pipelined ADC module.

Preconditions

none.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in the ADCP module.

Returns

None.

Remarks

None

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Disable Digital Comparator 1
PLIB_ADCP_DigCmpDisable(MY_ADCP_INSTANCE, ADCP_DCMP1);
```

5.6.2.5.7.3 PLIB_ADCP_DigCmpAidGet Function**C**

```
int16_t PLIB_ADCP_DigCmpAidGet(
    ADCP_MODULE_ID index,
    ADCP_DCMP_ID id
);
```

Description

This function tests the DigCmp Event Flag and if true, returns the ANx Identifier for the input which caused the event or -1 if there was no pending DigCmp event. The ID value returned corresponds to numeric portion of the analog input ID, ANx.

Preconditions

none.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in the ADCP module.

Returns

Value x of type int16_t where x is the index to the analog input ANx which caused the event or -1 if no event has occurred.

Remarks

None

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
int16_t DigCmpResult;

// Get Digital Comparator 1 result.
DigCmpResult = PLIB_ADCP_DigCmpAidGet(MY_ADCP_INSTANCE, ADCP_DCMP1);
if (DigCmpResult != -1) {
    // process event
}
```

5.6.2.5.7.4 PLIB_ADCP_DigCmpConfig Function**C**

```
void PLIB_ADCP_DigCmpConfig(
    ADCP_MODULE_ID index,
    ADCP_DCMP_ID id,
    bool globalIntEnable,
    bool inBetweenOrEqual,
    bool greaterEqualHi,
    bool lessThanHi,
    bool greaterEqualLo,
```

```

    bool lessThanLo,
    uint32_t inputEnable,
    int32_t hiLimit,
    int32_t loLimit
);

```

Description

Configures all parameters for the Digital Comparator module of the pipelined ADC.

Preconditions

The module is disabled when calling this function.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in this device
globalIntEnable	When set, comparator events are included in the Global Interrupt
inBetweenOrEqual	Event is generated when result is greater than or equal to loLimit and less than hiLimit
greaterEqualHi	Event is generated when result is greater than or equal to hiLimit
lessThanHi	Event is generated when result is less than hiLimit
greaterEqualLo	Event is generated when result is greater than or equal to loLimit
lessThanLo	Event is generated when result is less than loLimit
inputEnable	Bit field which determines which analog inputs are tested by this comparator module. Bit 0 applies to AN0 and bit 31 to AN31.
hiLimit	High limit in the same format as the conversion result.
loLimit	Low limit in the same format as the conversion result.

Returns

none

Remarks

This function must be called when the ADC is disabled. The format of hiLimit and loLimit must match the output format of the channel(s) specified in inputEnable.

Example

```

// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure the Digital Comparator
// Creates an event when the reading of AN0 is between 20% to 80% of the
// full scale 12-bit output.
PLIB_ADCP_DigCmpConfig( MY_ADCP_INSTANCE,    // ADCP module ID
                        ADCP_DCMP1,          // Comparator ID
                        FALSE,                // Global Int Enable
                        TRUE,                 // test for between low and high
                        FALSE,                // no test for greater then equal to high
                        FALSE,                // no test for less than high
                        FALSE,                // no test for greater then equal to low
                        FALSE,                // no test for less than low
                        1 << 3,               // enable AN3
                        0x0CCD,               // high limit, 80% of full scale
                        0x0333);              // low limit, 20% of full scale

// Enable Digital Comparator 1
PLIB_ADCP_DigCmpEnable(MY_ADCP_INSTANCE, ADCP_DCMP1);

```

5.6.2.5.7.5 PLIB_ADCP_ExistsDigCmp Function

C

```
bool PLIB_ADCP_ExistsDigCmp(  
    ADCP_MODULE_ID index  
);
```

Description

This function identifies whether the DigitalComparator feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_DigCmpConfig](#)
- [PLIB_ADCP_DigCmpEnable](#)
- [PLIB_ADCP_DigCmpDisable](#)
- [PLIB_ADCP_DigCmpAldGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the DigitalComparator feature:

- true - When DigitalComparator feature is supported on the device
- false - When DigitalComparator feature is not supported on the device

Remarks

None.

5.6.2.5.8 Interface Routines - Oversampling Digital Filter

5.6.2.5.8.1 PLIB_ADCP_OsampDigFilterEnable Function

C

```
void PLIB_ADCP_OsampDigFilterEnable(  
    ADCP_MODULE_ID index,  
    ADCP_ODFLTR_ID id  
);
```

Description

This function enables (turns ON) the selected Oversampling Digital Filter in the specified Pipelined ADC module.

Preconditions

The Oversampling Digital Filter should be configured using the [PLIB_ADCP_Configure\(\)](#) function prior to enabling.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in the ADCP module.

Returns

None.

Remarks

None

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the  
// application developer.  
  
// Enable OsampDigFilter1  
PLIB_ADCP_OsampDigFilterEnable(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
```

5.6.2.5.8.2 PLIB_ADCP_OsampDigFilterDisable Function**C**

```
void PLIB_ADCP_OsampDigFilterDisable(  
    ADCP_MODULE_ID index,  
    ADCP_ODFLTR_ID id  
);
```

Description

This function Disables (turns OFF) the selected Oversampling Digital Filter in the specified Pipelined ADC module.

Preconditions

none.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in the ADCP module.

Returns

None.

Remarks

None

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the  
// application developer.  
  
// Disable OsampDigFilter1  
PLIB_ADCP_OsampDigFilterDisable(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
```

5.6.2.5.8.3 PLIB_ADCP_OsampDigFilterDataRdy Function**C**

```
bool PLIB_ADCP_OsampDigFilterDataRdy(  
    ADCP_MODULE_ID index,  
    ADCP_ODFLTR_ID id  
);
```

Description

Used to determine if the ADCP digital filter has data ready. A TRUE is returned when data is available which can be fetched using [PLIB_ADCP_OsampDigFilterDataGet\(\)](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in this device

Returns

A boolean TRUE if data is ready or FALSE if not. .

Remarks

None.

Example

```
if (PLIB_ADCP_OsampDigFilterDataRdy(MY_ADCP_INSTANCE, ADCP_ODFLTR_ID_0)) {  
    // fetch and process data  
}
```

5.6.2.5.8.4 PLIB_ADCP_OsampDigFilterDataGet Function**C**

```
int16_t PLIB_ADCP_OsampDigFilterDataGet(  
    ADCP_MODULE_ID index,  
    ADCP_ODFLTR_ID id  
);
```

Description

This function is used to fetch data from the Oversampling Digital Filter. The format of the data is determined by the oversampling setting configuration defined in the call of [PLIB_ADCP_OsampDigFilterConfig\(\)](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the ADCP instance
dfiltrID	Identifier for the digital Filter in this device

Returns

A 16 bit result in the format specified by the filter's oversampling setting

Remarks

None.

Example

```
int16_t myODFLTRResult;  
  
if (PLIB_ADCP_OsampDigFilterDataRdy(MY_ADCP_INSTANCE, ADCP_ODFLTR1)) {  
    // fetch data  
    myODFLTRResult = PLIB_ADCP_OsampDigFilterDataGet(MY_ADCP_INSTANCE, ADCP_ODFLTR1);  
    // process result  
    ...  
}
```


5.6.2.5.8.5 PLIB_ADCP_ExistsOsampDigFilter Function

C

```
bool PLIB_ADCP_ExistsOsampDigFilter(  
    ADCP_MODULE_ID index  
);
```

Description

This function identifies whether the OsampDigitalFilter feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- [PLIB_ADCP_OsampDigFilterConfig](#)
- [PLIB_ADCP_OsampDigFilterEnable](#)
- [PLIB_ADCP_OsampDigFilterDisable](#)
- [PLIB_ADCP_OsampDigFilterDataRdy](#)
- [PLIB_ADCP_OsampDigFilterDataGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the OsampDigitalFilter feature:

- true - When OsampDigitalFilter feature is supported on the device
- false - When OsampDigitalFilter feature is not supported on the device

Remarks

None.

5.6.2.5.8.6 PLIB_ADCP_OsampDigFilterConfig Function

C

```
void PLIB_ADCP_OsampDigFilterConfig(  
    ADCP_MODULE_ID index,  
    ADCP_ODFLTR_ID id,  
    ADCP_INPUT_ID inputId,  
    ADCP_ODFLTR_SAMPLE_RATIO oversampleRatio,  
    bool globalIntEnable  
);
```

Description

Configures all parameters for the Oversampling Digital Filter module of the pipelined ADC.

Preconditions

The Oversampling Digital Filter module is disabled when calling this function.

Parameters

Parameters	Description
index	Identifier for the ADCP instance

id	Identifier for the digital Filter in this device
inputId	Identifier for the analog input to be oversampled
oversampleRatio	Enumerator specifying the oversampling ratio
globalIntEnable	When set, Filter events are included in the Global Interrupt

Returns

none

Remarks

This function must be called when the ADC is disabled.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure the Oversampling Digital Filter
// AN4 is oversampled at a 16X rate. No global interrupt is enabled.
PLIB_ADCP_OsampDigFilterConfig( MY_ADCP_INSTANCE,    // ADCP module ID
                                ADCP_ODFLTR1,        // Filter ID
                                ADCP_AN4,            // Oversample AN4
                                ADCP_ODFLTR_16X,     // 16 x oversampling
                                FALSE );              // No Global Int Enable

// Enable OsampDigFilter1
PLIB_ADCP_OsampDigFilterEnable(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
```

5.6.2.5.9 Data Types

5.6.2.5.9.1 ADCP_MODULE_ID Enumeration

C

```
typedef enum {
    ADCP_ID_1,
    ADC_NUMBER_OF_MODULES
} ADCP_MODULE_ID;
```

Description

ADC Module ID

This enumeration identifies the available ADC modules.

Members

Members	Description
ADCP_ID_1	ADC Module 1 ID
ADC_NUMBER_OF_MODULES	Number of available ADC modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

5.6.2.5.9.2 ADCP_VREF_SOURCE Enumeration

C

```
typedef enum {
    ADCP_VREF_AVDD_AVSS,
    ADCP_VREF_VREFP_AVSS,
    ADCP_VREF_AVDD_VREFN,
    ADCP_VREF_VREFP_VREFN,
} ADCP_VREF_SOURCE;
```

Description

ADCP VREF Source Select

This data type defines the ADCP Reference Voltage (VREF) Source Select

Members

Members	Description
ADCP_VREF_AVDD_AVSS	Reference voltage set to AVDD and AVSS
ADCP_VREF_VREFP_AVSS	Reference voltage set to VREF positive and AVSS
ADCP_VREF_AVDD_VREFN	Reference voltage set to AVDD and VREF negative
ADCP_VREF_VREFP_VREFN	Reference voltage set to VREF positive and VREF negative

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.3 ADCP_CLOCK_SOURCE Enumeration

C

```
typedef enum {
    ADCP_CLK_SRC_NONE,
    ADCP_CLK_SRC_SYSCLK,
    ADCP_CLK_SRC_RFCLK3,
    ADCP_CLK_SRC_FRC,
} ADCP_CLOCK_SOURCE;
```

Description

ADCP Clock Source Select

This enumeration data type defines the ADCP Clock Source Select

Members

Members	Description
ADCP_CLK_SRC_NONE	TAD clock disabled
ADCP_CLK_SRC_SYSCLK	TAD clock set to SYSCLK (TCY)
ADCP_CLK_SRC_RFCLK3	TAD clock set to REFCLK3
ADCP_CLK_SRC_FRC	TAD clock set to FRC

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.4 ADCP_CLASS12_INPUT_ID Enumeration

C

```
typedef enum {
```

```
ADCP_CLASS12_AN0,  
ADCP_CLASS12_AN1,  
ADCP_CLASS12_AN2,  
ADCP_CLASS12_AN3,  
ADCP_CLASS12_AN4,  
ADCP_CLASS12_AN5,  
ADCP_CLASS12_AN6,  
ADCP_CLASS12_AN7,  
ADCP_CLASS12_AN8,  
ADCP_CLASS12_AN9,  
ADCP_CLASS12_AN10,  
ADCP_CLASS12_AN11  
} ADCP_CLASS12_INPUT_ID;
```

Description

ADC Class 1 and Class 2 Input ID

Used to identify the Class 1 and Class 2 ADC analog inputs

Members

Members	Description
ADCP_CLASS12_AN0	Analog Input AN0
ADCP_CLASS12_AN1	Analog Input AN1
ADCP_CLASS12_AN2	Analog Input AN2
ADCP_CLASS12_AN3	Analog Input AN3
ADCP_CLASS12_AN4	Analog Input AN4
ADCP_CLASS12_AN5	Analog Input AN5
ADCP_CLASS12_AN6	Analog Input AN6
ADCP_CLASS12_AN7	Analog Input AN7
ADCP_CLASS12_AN8	Analog Input AN8
ADCP_CLASS12_AN9	Analog Input AN9
ADCP_CLASS12_AN10	Analog Input AN10
ADCP_CLASS12_AN11	Analog Input AN11

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.5 ADCP_SH_ID Enumeration

C

```
typedef enum {  
    ADCP_SH0,  
    ADCP_SH1,  
    ADCP_SH2,  
    ADCP_SH3,  
    ADCP_SH4,  
    ADCP_SH5,  
    ADCP_NUMBER_OF_SH  
} ADCP_SH_ID;
```

Description

ADCP S&H Select

This enumeration identifies all supported S&Hs.

Members

Members	Description
ADCP_SH0	S&H 0
ADCP_SH1	S&H 1
ADCP_SH2	S&H 2
ADCP_SH3	S&H 3
ADCP_SH4	S&H 4
ADCP_SH5	S&H 5
ADCP_NUMBER_OF_SH	Number of S&Hs

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.6 ADCP_DSH_ID Enumeration**C**

```
typedef enum {
    ADCP_DSH0,
    ADCP_DSH1,
    ADCP_DSH2,
    ADCP_DSH3,
    ADCP_DSH4,
    ADCP_NUMBER_OF_DSH
} ADCP_DSH_ID;
```

Description

ADCP Dedicated S&H Select

This enumeration identifies the supported Dedicated S&Hs.

Members

Members	Description
ADCP_DSH0	Dedicated S&H 0
ADCP_DSH1	Dedicated S&H 1
ADCP_DSH2	Dedicated S&H 2
ADCP_DSH3	Dedicated S&H 3
ADCP_DSH4	Dedicated S&H 4
ADCP_NUMBER_OF_DSH	Number of Dedicated S&Hs

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.7 ADCP_SH_MODE Enumeration**C**

```
typedef enum {
    ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR,
    ADCP_SH_MODE_SINGLE_ENDED_TWOS_COMP,
    ADCP_SH_MODE_DIFFERENTIAL_UNIPOLAR,
    ADCP_SH_MODE_DIFFERENTIAL_TWOS_COMP
} ADCP_SH_MODE;
```

Description

ADCP S&H Mode

Data type defines the available modes for the S&H

Members

Members	Description
ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR	Sinlge Ended input, Unipolar encoded
ADCP_SH_MODE_SINGLE_ENDED_TWOS_COMP	Sinlge Ended input, two's compliment encoded
ADCP_SH_MODE_DIFFERENTIAL_UNIPOLAR	Differential input, Unipolar encoded
ADCP_SH_MODE_DIFFERENTIAL_TWOS_COMP	Differential input, two's compliment encoded

Remarks

None

5.6.2.5.9.8 ADCP_INPUT_ID Enumeration

C

```
typedef enum {
    ADCP_AN0,
    ADCP_AN1,
    ADCP_AN2,
    ADCP_AN3,
    ADCP_AN4,
    ADCP_AN5,
    ADCP_AN6,
    ADCP_AN7,
    ADCP_AN8,
    ADCP_AN9,
    ADCP_AN10,
    ADCP_AN11,
    ADCP_AN12,
    ADCP_AN13,
    ADCP_AN14,
    ADCP_AN15,
    ADCP_AN16,
    ADCP_AN17,
    ADCP_AN18,
    ADCP_AN19,
    ADCP_AN20,
    ADCP_AN21,
    ADCP_AN22,
    ADCP_AN23,
    ADCP_AN24,
    ADCP_AN25,
    ADCP_AN26,
    ADCP_AN27,
    ADCP_AN28,
    ADCP_AN29,
    ADCP_AN30,
    ADCP_AN31,
    ADCP_AN32,
    ADCP_AN33,
    ADCP_AN34,
    ADCP_AN35,
    ADCP_AN36,
    ADCP_AN37,
    ADCP_AN38,
    ADCP_AN39,
    ADCP_AN40,
```

```

ADCP_AN41,
ADCP_AN42,
ADCP_AN43,
ADCP_AN44,
ADCP_AN45,
ADCP_AN46,
ADCP_AN47,
ADCP_AN48,
ADCP_AN49,
ADCP_AN50,
ADCP_AN51,
ADCP_AN52,
ADCP_AN53,
ADCP_AN54,
ADCP_AN55,
ADCP_AN56,
ADCP_AN57,
ADCP_AN58,
ADCP_AN59,
ADCP_AN60,
ADCP_AN61,
ADCP_AN62,
ADCP_AN63,
ADCP_IVREF,
ADCP_IVTEMP
} ADCP_INPUT_ID;

```

Description

ADC Input ID

Type for identifying the available ADC Inputs, regardless of Class.

Members

Members	Description
ADCP_AN0	Analog Input AN0
ADCP_AN1	Analog Input AN1
ADCP_AN2	Analog Input AN2
ADCP_AN3	Analog Input AN3
ADCP_AN4	Analog Input AN4
ADCP_AN5	Analog Input AN5
ADCP_AN6	Analog Input AN6
ADCP_AN7	Analog Input AN7
ADCP_AN8	Analog Input AN8
ADCP_AN9	Analog Input AN9
ADCP_AN10	Analog Input AN10
ADCP_AN11	Analog Input AN11
ADCP_AN12	Analog Input AN12
ADCP_AN13	Analog Input AN13
ADCP_AN14	Analog Input AN14
ADCP_AN15	Analog Input AN15
ADCP_AN16	Analog Input AN16
ADCP_AN17	Analog Input AN17
ADCP_AN18	Analog Input AN18
ADCP_AN19	Analog Input AN19
ADCP_AN20	Analog Input AN20

ADCP_AN21	Analog Input AN21
ADCP_AN22	Analog Input AN22
ADCP_AN23	Analog Input AN23
ADCP_AN24	Analog Input AN24
ADCP_AN25	Analog Input AN25
ADCP_AN26	Analog Input AN26
ADCP_AN27	Analog Input AN27
ADCP_AN28	Analog Input AN28
ADCP_AN29	Analog Input AN29
ADCP_AN30	Analog Input AN30
ADCP_AN31	Analog Input AN31
ADCP_AN32	Analog Input AN32
ADCP_AN33	Analog Input AN33
ADCP_AN34	Analog Input AN34
ADCP_AN35	Analog Input AN35
ADCP_AN36	Analog Input AN36
ADCP_AN37	Analog Input AN37
ADCP_AN38	Analog Input AN38
ADCP_AN39	Analog Input AN39
ADCP_AN40	Analog Input AN40
ADCP_AN41	Analog Input AN41
ADCP_AN42	Analog Input AN42
ADCP_AN43	Analog Input AN43
ADCP_AN44	Analog Input AN44
ADCP_AN45	Analog Input AN45
ADCP_AN46	Analog Input AN46
ADCP_AN47	Analog Input AN47
ADCP_AN48	Analog Input AN48
ADCP_AN49	Analog Input AN49
ADCP_AN50	Analog Input AN50
ADCP_AN51	Analog Input AN51
ADCP_AN52	Analog Input AN52
ADCP_AN53	Analog Input AN53
ADCP_AN54	Analog Input AN54
ADCP_AN55	Analog Input AN55
ADCP_AN56	Analog Input AN56
ADCP_AN57	Analog Input AN57
ADCP_AN58	Analog Input AN58
ADCP_AN59	Analog Input AN59
ADCP_AN60	Analog Input AN60
ADCP_AN61	Analog Input AN61
ADCP_AN62	Analog Input AN62
ADCP_AN63	Analog Input AN63
ADCP_IVREF	Analog Input for Internal Voltage Reference
ADCP_IVTEMP	Analog Input for Internal Temperature Sensor

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.9 ADCP_DCMP_ID Enumeration

C

```
typedef enum {
    ADCP_DCMP1,
    ADCP_DCMP2,
    ADCP_DCMP3,
    ADCP_DCMP4,
    ADCP_DCMP5,
    ADCP_DCMP6,
    ADCP_NUMBER_OF_DCMP
} ADCP_DCMP_ID;
```

Description

ADCP Digital Comparator

This enumeration identifies all supported digital comparators for this ADCP module.

Members

Members	Description
ADCP_DCMP1	DCMP1
ADCP_DCMP2	DCMP2
ADCP_DCMP3	DCMP3
ADCP_DCMP4	DCMP4
ADCP_DCMP5	DCMP5
ADCP_DCMP6	DCMP6
ADCP_NUMBER_OF_DCMP	Number of Digital Comparators

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.10 ADCP_ODFLTR_ID Enumeration

C

```
typedef enum {
    ADCP_ODFLTR1,
    ADCP_ODFLTR2,
    ADCP_ODFLTR3,
    ADCP_ODFLTR4,
    ADCP_ODFLTR5,
    ADCP_ODFLTR6,
    ADCP_NUMBER_OF_ODFLTR
} ADCP_ODFLTR_ID;
```

Description

ADCP Oversampling Digital Filter

This enumeration identifies all supported digital Filters for this ADCP module.

Members

Members	Description
ADCP_ODFLTR1	ODFLTR1

ADCP_ODFLTR2	ODFLTR2
ADCP_ODFLTR3	ODFLTR3
ADCP_ODFLTR4	ODFLTR4
ADCP_ODFLTR5	ODFLTR5
ADCP_ODFLTR6	ODFLTR6
ADCP_NUMBER_OF_ODFLTR	Number of Oversampling Digital Filters

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.11 ADCP_ODFLTR_OSR Enumeration**C**

```
typedef enum {
    ADCP_ODFLTR_4X,
    ADCP_ODFLTR_16X,
    ADCP_ODFLTR_64X,
    ADCP_ODFLTR_256X,
    ADCP_ODFLTR_2X,
    ADCP_ODFLTR_8X,
    ADCP_ODFLTR_32X,
    ADCP_ODFLTR_128X
} ADCP_ODFLTR_OSR;
```

Description

ADCP Oversampling Ratio

ADCP Oversampling Digital Filter oversampling Ratio

This enumeration identifies all supported digital Filter oversampling ratios for this ADCP module. Oversampling ratios determine the number of samples used to generate a single output and the resulting resolution and format.

Members

Members	Description
ADCP_ODFLTR_4X	4x oversampling, shift sum 1 bit to right, output data is 13 bits
ADCP_ODFLTR_16X	16x oversampling, shift sum 2 bits to right, output data is 14 bits
ADCP_ODFLTR_64X	64x oversampling, shift sum 3 bits to right, output data is 15 bits
ADCP_ODFLTR_256X	256x oversampling, shift sum 4 bits to right, output data is 16 bits
ADCP_ODFLTR_2X	2x oversampling, shift sum 0 bits to right, output data is in 12.1 format
ADCP_ODFLTR_8X	8x oversampling, shift sum 1 bit to right, output data is in 13.1 format
ADCP_ODFLTR_32X	32x oversampling, shift sum 2 bits to right, output data is in 14.1 format
ADCP_ODFLTR_128X	128x oversampling, shift sum 3 bit to right, output data is in 15.1 format

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.12 ADCP_SCAN_TRG_SRC Enumeration**C**

```
typedef enum {
    ADCP_SCAN_TRG_SRC_NONE,
    ADCP_SCAN_TRG_SRC_SOFTWARE,
    ADCP_SCAN_TRG_SRC_INT0,
    ADCP_SCAN_TRG_SRC_TMR1_MATCH,
```

```

ADCP_SCAN_TRG_SRC_TMR3_MATCH,
ADCP_SCAN_TRG_SRC_TMR5_MATCH,
ADCP_SCAN_TRG_SRC_OCMP1,
ADCP_SCAN_TRG_SRC_OCMP3,
ADCP_SCAN_TRG_SRC_OCMP5,
ADCP_SCAN_TRG_SRC_COMP1_COUT,
ADCP_SCAN_TRG_SRC_COMP2_COUT
} ADCP_SCAN_TRG_SRC;

```

Description

ADCP Channel Scan Trigger Source Select

This data type defines the ADCP Channel Scan Trigger Source Selections

Members

Members	Description
ADCP_SCAN_TRG_SRC_NONE	No scan trigger source is selected
ADCP_SCAN_TRG_SRC_SOFTWARE	Global Software trigger selected as scan trigger source
ADCP_SCAN_TRG_SRC_INT0	Interrupt 0 (INT0) selected as scan trigger source
ADCP_SCAN_TRG_SRC_TMR1_MATCH	Timer 1 Match (TMR1) selected as scan trigger source
ADCP_SCAN_TRG_SRC_TMR3_MATCH	Timer 3 Match (TMR3) selected as scan trigger source
ADCP_SCAN_TRG_SRC_TMR5_MATCH	Timer 5 Match (TMR5) selected as scan trigger source
ADCP_SCAN_TRG_SRC_OCMP1	Output Compare 1 (OCMP1) selected as scan trigger source
ADCP_SCAN_TRG_SRC_OCMP3	Output Compare 3 (OCMP3) selected as scan trigger source
ADCP_SCAN_TRG_SRC_OCMP5	Output Compare 5 (OCMP5) selected as scan trigger source
ADCP_SCAN_TRG_SRC_COMP1_COUT	Analog Comparator 1 (COUT) selected as scan trigger source
ADCP_SCAN_TRG_SRC_COMP2_COUT	Analog Comparator 2 (COUT) selected as scan trigger source

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.13 ADCP_TRG_SRC Enumeration**C**

```

typedef enum {
    ADCP_TRG_SRC_NONE,
    ADCP_TRG_SRC_SOFTWARE,
    ADCP_TRG_SRC_SCAN_TRG,
    ADCP_TRG_SRC_INT0,
    ADCP_TRG_SRC_TMR1_MATCH,
    ADCP_TRG_SRC_TMR3_MATCH,
    ADCP_TRG_SRC_TMR5_MATCH,
    ADCP_TRG_SRC_OCMP1,
    ADCP_TRG_SRC_OCMP3,
    ADCP_TRG_SRC_OCMP5,
    ADCP_TRG_SRC_COMP1_COUT,
    ADCP_TRG_SRC_COMP2_COUT
} ADCP_TRG_SRC;

```

Description

ADCP Trigger Source Select

This data type defines the ADCP Trigger Source Selections

Members

Members	Description
ADCP_TRG_SRC_NONE	No trigger source is selected
ADCP_TRG_SRC_SOFTWARE	Global Software trigger selected as the trigger source
ADCP_TRG_SRC_SCAN_TRG	Use the Channel Scan Trigger as the trigger source (input is part of ch scan)
ADCP_TRG_SRC_INT0	Interrupt 0 (INT0) selected as the trigger source
ADCP_TRG_SRC_TMR1_MATCH	Timer 1 Match (TMR1) selected as the trigger source
ADCP_TRG_SRC_TMR3_MATCH	Timer 3 Match (TMR3) selected as the trigger source
ADCP_TRG_SRC_TMR5_MATCH	Timer 5 Match (TMR5) selected as the trigger source
ADCP_TRG_SRC_OCMP1	Output Compare 1 (OCMP1) selected as the trigger source
ADCP_TRG_SRC_OCMP3	Output Compare 3 (OCMP3) selected as the trigger source
ADCP_TRG_SRC_OCMP5	Output Compare 5 (OCMP5) selected as the trigger source
ADCP_TRG_SRC_COMP1_COUT	Analog Comparator 1 (COUT) selected as the trigger source
ADCP_TRG_SRC_COMP2_COUT	Analog Comparator 2 (COUT) selected as the trigger source

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.2.5.9.14 AN_SELECT Union**C**

```

union AN_SELECT {
    int ret_val;
    struct {
        uint32_t lowWord;
        uint32_t highWord;
    };
    struct {
        uint32_t AN0 : 1;
        uint32_t AN1 : 1;
        uint32_t AN2 : 1;
        uint32_t AN3 : 1;
        uint32_t AN4 : 1;
        uint32_t AN5 : 1;
        uint32_t AN6 : 1;
        uint32_t AN7 : 1;
        uint32_t AN8 : 1;
        uint32_t AN9 : 1;
        uint32_t AN10 : 1;
        uint32_t AN11 : 1;
        uint32_t AN12 : 1;
        uint32_t AN13 : 1;
        uint32_t AN14 : 1;
        uint32_t AN15 : 1;
        uint32_t AN16 : 1;
        uint32_t AN17 : 1;
        uint32_t AN18 : 1;
        uint32_t AN19 : 1;
        uint32_t AN20 : 1;
        uint32_t AN21 : 1;
        uint32_t AN22 : 1;
        uint32_t AN23 : 1;
        uint32_t AN24 : 1;
        uint32_t AN25 : 1;
        uint32_t AN26 : 1;
        uint32_t AN27 : 1;
        uint32_t AN28 : 1;
    };
}

```

```

uint32_t AN29 : 1;
uint32_t AN30 : 1;
uint32_t AN31 : 1;
uint32_t AN32 : 1;
uint32_t AN33 : 1;
uint32_t AN34 : 1;
uint32_t AN35 : 1;
uint32_t AN36 : 1;
uint32_t AN37 : 1;
uint32_t AN38 : 1;
uint32_t AN39 : 1;
uint32_t AN40 : 1;
uint32_t AN41 : 1;
uint32_t AN42 : 1;
uint32_t AN43 : 1;
uint32_t AN44 : 1;
uint32_t AN45 : 1;
uint32_t AN46 : 1;
uint32_t AN47 : 1;
uint32_t AN48 : 1;
uint32_t AN49 : 1;
uint32_t AN50 : 1;
uint32_t AN51 : 1;
uint32_t AN52 : 1;
uint32_t AN53 : 1;
uint32_t AN54 : 1;
uint32_t AN55 : 1;
uint32_t AN56 : 1;
uint32_t AN57 : 1;
uint32_t AN58 : 1;
uint32_t AN59 : 1;
uint32_t AN60 : 1;
uint32_t AN61 : 1;
uint32_t AN62 : 1;
uint32_t AN63 : 1;
};
};
};

```

Description

ADCP AN Select Union

Structure labels identify the analog inputs associated with each bit when initializing the structure. Unsigned 32 bit integers, lowWord and highWord are used as arguments to the function.

Remarks

See the Channel Scanning example in the ADCP Help documentation regarding use of this type define.

5.6.2.5.9.15 AN_READY Type**C**

```
typedef AN_SELECT AN_READY;
```

Description

ADCP AN Ready Union

Structure labels identify the analog inputs associated with each bit for testing ready status of individual inputs. Unsigned 32 bit integers, lowWord and highWord allow for testing ready status of groups of bits using a mask.

Remarks

See the Simultaneous Sampling or Channel Scanning examples in the ADCP Help documentation regarding use of this type define.

5.6.2.6 Files

Files

Name	Description
plib_adcp.h	ADCP PLIB Interface Header for ADCP common definitions

Description

5.6.2.6.1 plib_adcp.h

ADCP - Analog to Digital Converter, Pipelined Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the ADCP PLIB for all families of Microchip microcontrollers..The definitions in this file are common to ADCP peripheral.

Functions

	Name	Description
≡	PLIB_ADCP_AlternateInputSelect	Selects the alternate physical input for the specified dedicated (Class 1) S&H.
≡	PLIB_ADCP_CalibrationStart	Initiates Pipelined ADC Calibration.
≡	PLIB_ADCP_ChannelScanConfigure	Selects input to include in Channel Scan Mode
≡	PLIB_ADCP_Class12TriggerConfigure	Configures a Class 1 or Class 2 Trigger Source.
≡	PLIB_ADCP_Configure	Configures the Pipelined ADC converter including the ADC calibration registers.
≡	PLIB_ADCP_DefaultInputSelect	Selects the default physical input for the specified dedicated (Class 1) S&H.
≡	PLIB_ADCP_DigCmpAldGet	Returns the Analog Input ID of for the Comparator Event
≡	PLIB_ADCP_DigCmpConfig	Configures the Digital Comparator on the Pipelined ADC converter.
≡	PLIB_ADCP_DigCmpDisable	Disables the Digital Comparator in the Pipelined ADC module
≡	PLIB_ADCP_DigCmpEnable	Enables the Digital Comparator in the Pipelined ADC module
≡	PLIB_ADCP_Disable	Pipelined ADC module is disabled (turned OFF).
≡	PLIB_ADCP_Enable	Enables the Pipelined ADC module
≡	PLIB_ADCP_ExistsCalibration	Identifies whether the Calibration feature exists on the ADCP module
≡	PLIB_ADCP_ExistsChannelScan	Identifies whether the ChannelScan feature exists on the ADCP module
≡	PLIB_ADCP_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCP module
≡	PLIB_ADCP_ExistsConversionResults	Identifies whether the ConversionResults feature exists on the ADCP module
≡	PLIB_ADCP_ExistsDigCmp	Identifies whether the DigitalComparator feature exists on the ADCP module
≡	PLIB_ADCP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCP module
≡	PLIB_ADCP_ExistsInputSelect	Identifies whether the InputSelect feature exists on the ADCP module
≡	PLIB_ADCP_ExistsLowPowerControl	Identifies whether the LowPowerControl feature exists on the ADCP module
≡	PLIB_ADCP_ExistsModeSelect	Identifies whether the ModeSelect feature exists on the ADCP module
≡	PLIB_ADCP_ExistsOsampDigFilter	Identifies whether the OsampDigitalFilter feature exists on the ADCP module

	PLIB_ADCP_ExistsReadyStatus	Identifies whether the ReadyStatus feature exists on the ADCP module
	PLIB_ADCP_ExistsTriggering	Identifies whether the Triggering feature exists on the ADCP module
	PLIB_ADCP_GlobalSoftwareTrigger	Initiate a Global Software Trigger on the specified module.
	PLIB_ADCP_IndividualTrigger	Triggers an individual channel independent of the configured trigger source
	PLIB_ADCP_LowPowerStateEnter	Places the Pipelined ADC module in a low power state.
	PLIB_ADCP_LowPowerStateExit	Takes the Pipelined ADC module out of low power state and puts it into an operational mode.
	PLIB_ADCP_LowPowerStateGet	Returns the state of the low power setting.
	PLIB_ADCP_ModuleIsReady	Returns the overall ready status of the module.
	PLIB_ADCP_OsampDigFilterConfig	Configures the Oversampling Digital Filter on the Pipelined ADC converter.
	PLIB_ADCP_OsampDigFilterDataGet	Used to fetch the data result from the Oversampling Digital Filter.
	PLIB_ADCP_OsampDigFilterDataRdy	Used to determine if the Oversampling Digital Filter has data ready.
	PLIB_ADCP_OsampDigFilterDisable	Disables the Oversampling Digital Filter in the Pipelined ADC module
	PLIB_ADCP_OsampDigFilterEnable	Enables the Oversampling Digital Filter in the Pipelined ADC module
	PLIB_ADCP_ResultGet	Returns a Pipelined ADC conversion result.
	PLIB_ADCP_ResultReady	Returns the ADC conversion result ready bits for the module.
	PLIB_ADCP_ResultReadyGrpIntConfigure	Selects input to include in global interrupt.
	PLIB_ADCP_SHModeSelect	Selects the mode for the specified S&H.

Types

	Name	Description
	AN_READY	This union (identical to the AN_SELECT union) is used as the return value by the PLIB_ADCP_Result_Ready() function.

Unions

	Name	Description
	AN_SELECT	This union of structures is provided to simply selection of analog inputs for inclusion in a particular function.

File Name

plib_ADCP.h

Company

Microchip Technology Incorporated

5.6.3 BMX Peripheral Library

5.6.3.1 Introduction

Bus Matrix (BMX) Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Bus Matrix (BMX) modules on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Bus Matrix Overview

The Bus Matrix is essentially a high-speed switch that establishes connections between devices. Devices can be either master (initiator) or slave (target).

The Bus Matrix also performs these other important functions:

- Partition memory - The Bus Matrix provides the ability to partition both RAM and flash memory into kernel and user areas. It also controls the type of access, program or data, for each memory area.
- Enable/disable program Flash memory cache for Direct Memory Access (DMA)
- Enable/disable bus error exceptions - The Bus Matrix provides the ability to disable bus error exceptions for debugging. They are enabled by default.
- Set Bus Arbitration mode - The Bus Matrix provides the ability to select between arbitration modes. The arbitration modes control the order and priority of initiator access to peripherals.

5.6.3.2 Release Notes

MPLAB Harmony Version: v0.70b **BMX Peripheral Library Version :** 0.70b Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.3.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.3.4 Using the Library

This topic describes the basic architecture of the BMX Peripheral Library and provides information and examples on how to use it.

Interface Header File: `peripheral.h`

The interface to the BMX library is defined in the "`plib_bmx.h`" header file, which is included by the "`peripheral.h`" peripheral library header file. Any C language source (.c) file that uses the BMX library must include "`peripheral.h`".

Library File:

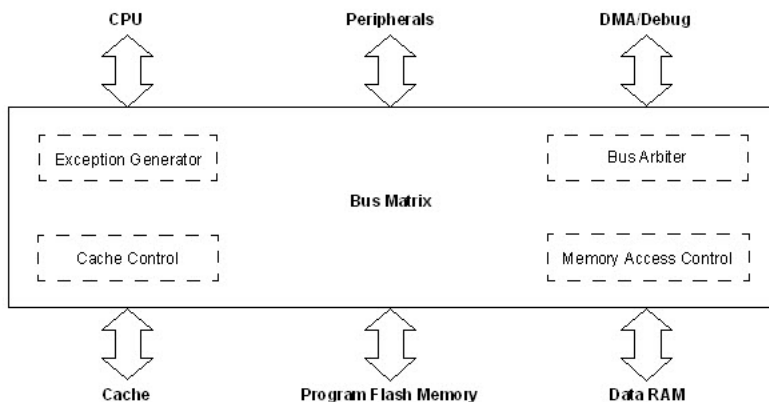
The BMX Peripheral Library is part of the processor-specific peripheral library archive (.a) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

5.6.3.4.1 Hardware Abstraction Model

This library provides a low-level abstraction of the BMX module on Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The following figure shows the Bus Matrix hardware abstraction model.



Hardware Abstraction Model

The BMX Peripheral Library provides interface routines to interact with the blocks shown in the diagram above.

The **Exception Generator** generates a bus error exception for unmapped address accesses from all Bus Matrix initiators:

- Initiator Expansion Interface (IXI) shared bus
- In-Circuit Debug (ICD) unit
- Direct Memory Access (DMA) controller
- CPU data bus
- CPU instruction bus

Each of the above can be individually enabled or disabled with Exception Generator peripheral library routines. All are enabled by default.

The **Cache Control** simply enables/disables program Flash memory (PFM) data cacheability for DMA accesses. The default setting is disabled. When enabled, the PCACHE module must have data caching enabled.

The **Bus Arbitrator** assigns priority levels to bus initiators following one of three priority schemes:

- Fixed, with CPU higher than DMA
- Fixed, with DMA higher than CPU (default)
- Rotating between four priority sequences

The **Memory Access Control** partitions program Flash memory and RAM into kernel and user areas. The RAM can be further partitioned into data and program segments for both kernel and user areas. By default, the full PFM and RAM are mapped to Kernel mode.

5.6.3.4.2 How the Library Works

The Bus Matrix is initialized in the start-up code, before an application is invoked. Most applications will not need to modify the Bus Matrix default settings. The BMX Peripheral Library is available for those applications requiring more complex memory partitioning, including operating systems and applications requiring specialized memory access. The ability to disable exceptions for unmapped memory access may be useful for debugging, but is not recommended for normal use. Changes to the arbitration scheme may improve performance for CPU or DMA intensive applications.

5.6.3.4.2.1 Exception Generator

Any unmapped address access by a Bus Matrix initiator will by default generate an exception. These exceptions may be individually enabled or disabled with the peripheral library.

Example: Enable CPU instruction exception

```
// Enable CPU instruction exception
PLIB_BMX_ISBusExceptionEnable();
```

5.6.3.4.2.2 Cache Control

This peripheral library function enables program Flash memory (data) cacheability for DMA accesses. Data caching must be enabled in the pcache. Hits are still read from the cache, but misses do not update the cache. This function is disabled by default.

Example: Enable PFM Cacheability for DMA access

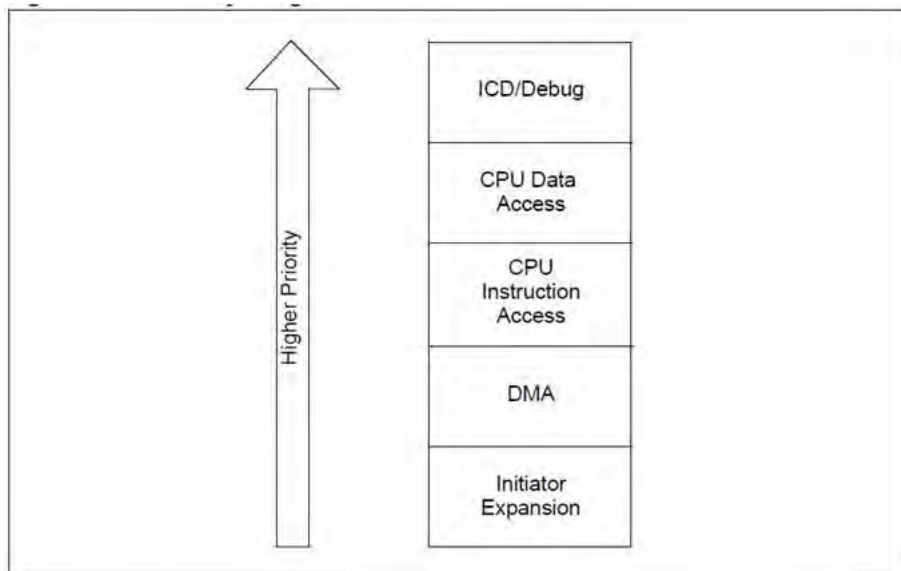
```
PLIB_BMX_EnablePfmCacheDma();
```

5.6.3.4.2.3 Bus Arbiter

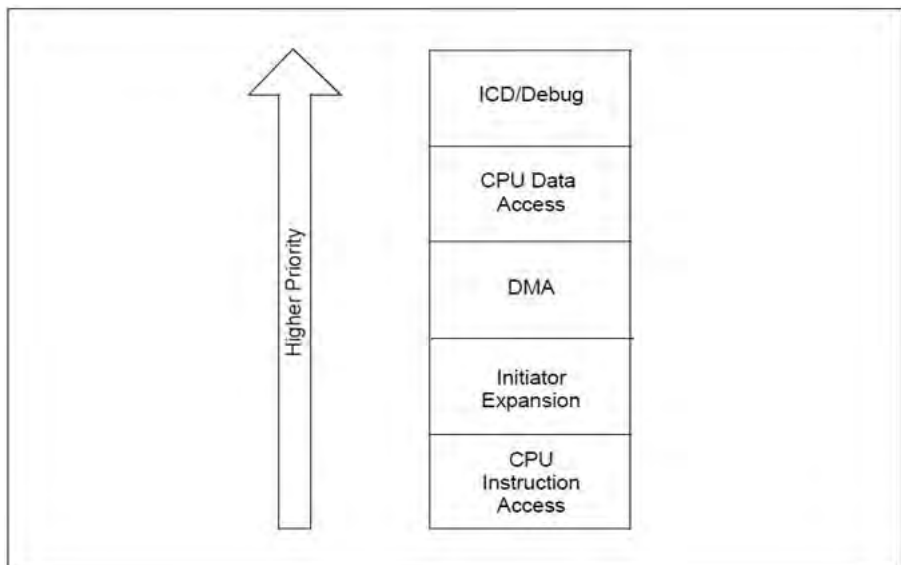
Since there can be more than one initiator attempting to access the same target, an arbitration scheme must be used to control access to the target. The arbitration modes assign priority levels to all the initiators. The initiator with the higher priority level will always win target access over a lower priority initiator. The BMX Bus Arbitrator allows the programmer to select from one of three arbitration modes.

PLIB_BMX_ARB_MODE_INST

Arbitration mode 0 is a fixed priority scheme, with the CPU given higher priority than DMA. This mode can starve the DMA, so choose this mode when DMA is not being used.

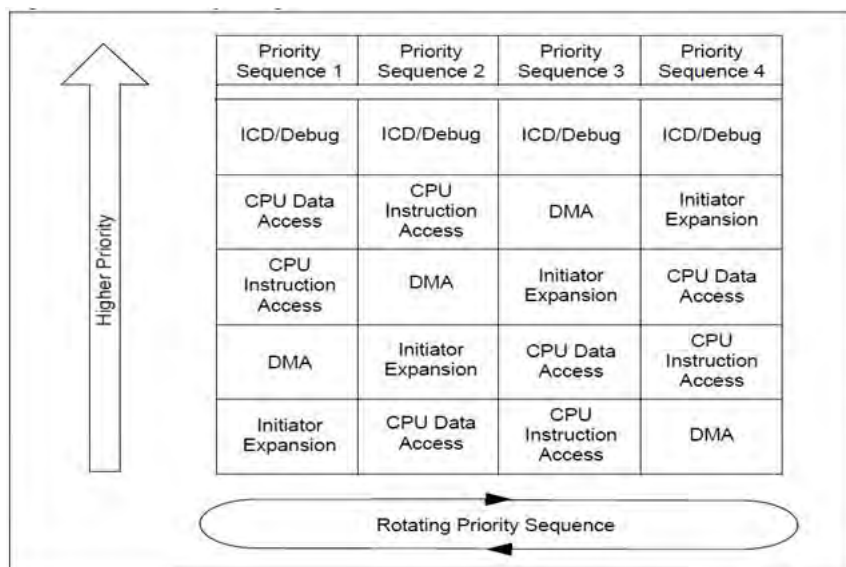
**PLIB_BMX_ARB_MODE_DMA**

Arbitration Mode 1 is a fixed priority scheme like Mode 0, except that the CPU is always the lowest priority. Mode 1 is the default mode.

**PLIB_BMX_ARB_MODE_ROT**

Mode 2 arbitration supports rotating priority assignments to all initiators. Instead of a fixed priority assignment, each initiator is assigned the highest priority in a rotating fashion. In this mode, the rotating priority is applied with the following exceptions:

1. CPU data is always selected over CPU instruction.
2. ICD is always the highest priority.
3. When the CPU is processing an exception (EXL = 1) or an error (ERL = 1), the arbitrator temporarily reverts to Mode 0.



Note: Priority Sequence 2 is not selected in the rotation priority scheme if there is a pending CPU data access. In this case, once the data access is complete, Sequence 2 is selected.

Example: Set Bus Arbitration Mode to Rotating Priority

```
PLIB_BMX_SetArbitrationMode(PLIB_BMX_ARB_ROT);
```

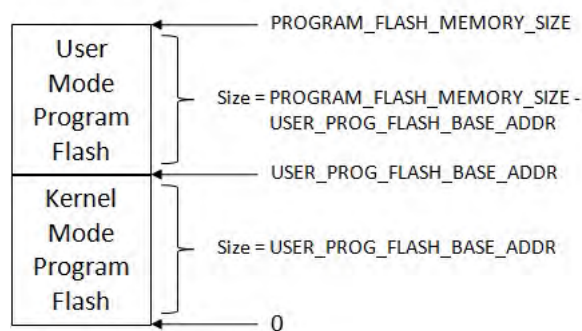
5.6.3.4.2.4 Memory Access Control

The Bus Matrix allows the programmer to partition program Flash memory into user and kernel partitions. It also allows the programmer to partition data RAM into user and kernel partitions, each of which can be sub-divided into program and data partitions. At reset, the entire program Flash memory is one kernel partition and the entire data RAM is one kernel data partition.

Program Flash Memory Partitioning

Partitioning of the program Flash memory is accomplished by setting the offset of the user partition within the program Flash memory using the `PLIB_BMX_ProgramFlashPartition()` function. The size of the program Flash memory varies among devices, but can be retrieved with the `PLIB_BMX_ProgramFlashMemorySizeGet()` function. The offset must be a multiple of the program Flash memory block size. The program Flash memory block size may vary among devices. If programmed with a value that is not a multiple of the block size, the value will automatically be truncated to a block size boundary. At reset, the entire program Flash memory is mapped to Kernel mode.

Allocation of Program Flash Memory



Example: Create a User Mode Partition of 12K in Program Flash Memory

```

size_t pfmSize;
size_t userSize = (6 * PLIB_PCACHE_PFM_BLOCK_SIZE);
size_t userOffset;

// Get size of PFM
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet();
userOffset = pfmSize - userSize;
if (userOffset > 0)
{
    PLIB_BMX_ProgramFlashPartitionSet(userOffset);
}

```

Data RAM Memory Partitioning

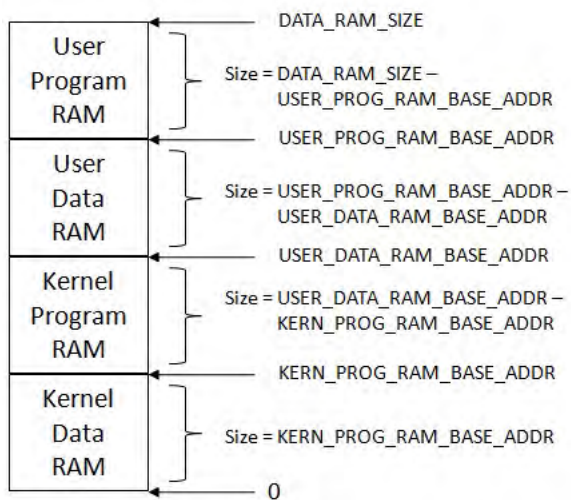
The Data RAM Can be partitioned into four sections:

- Kernel Data
- Kernel Program
- User Data
- User Program

Similar to the program Flash memory partitioning, the data RAM partitions are created by setting the base addresses of the various partitions. The BMX Peripheral Library provides a function to set all of the partitions at once. The size of the data RAM varies among devices, but may be retrieved with the [PLIB_BMX_DataRAMSizeGet\(\)](#) function. The partitions must be a multiple of the Data RAM memory block size. The data RAM memory block size may vary among devices. If programmed with a value that is not a multiple of the block size, the partition will automatically be truncated to a block size boundary. At reset, the entire data RAM is mapped to Kernel mode.

Note: At reset, the entire data RAM is mapped to Kernel mode and all of the offsets are set to zero. To partition the data RAM, all of the offsets must be programmed. If any of the offsets are set to zero, the entire data RAM is allocated to kernel data.

Allocation of Data RAM



Example: RAM Partitioned as Kernel Data, Kernel Program, User Data and User Program

```

//Total Data RAM = 32KB.
size_t totalRamSize;
size_t kernProgOffset;
size_t userDataOffset;
size_t userProgOffset;

```

```

size_t kernDataRamSize = (3 * PLIB_PCACHE_DRM_BLOCK_SIZE); //Kernel Data RAM = 6KB
size_t kernProgRamSize = (2 * PLIB_PCACHE_DRM_BLOCK_SIZE); //Kernel Program RAM = 4KB
size_t userDataRamSize = (6 * PLIB_PCACHE_DRM_BLOCK_SIZE); //User Data RAM = 12KB
size_t userProgRamSize = (5 * PLIB_PCACHE_DRM_BLOCK_SIZE); //User Program RAM = 10KB

//Get Size of Data RAM
totalRamSize = PLIB_BMX_DataRAMSizeGet();

//Verify our partition sizes fit our RAM
if ((kernDataRamSize + kernProgRamSize + userDataRamSize + userProgRamSize) != totalRamSize)
{
    printf("RAM Partitioning Error\n");
    return -1;
}

kernProgOffset = kernDataRamSize;
userDataOffset = kernDataRamSize + kernProgRamSize;
userProgOffset = userDataOffset + userDataRamSize;

PLIB_BMX_DataRAMPartitionSet(kernProgOffset, userDataOffset, userProgOffset);

```

Example: Determine the size of Data RAM partitions

```

size_t kernProgOffset;
size_t userDataOffset;
size_t userProgOffset;
size_t kernDataPart;
size_t kernProgPart;
size_t userDataPart;
size_t userProgPart;
size_t totalRamSize;
totalRamSize = PLIB_BMX_DataRAMSizeGet();
kernProgOffset = PLIB_BMX_DataRAMKernelProgramOffsetGet();
userDataOffset = PLIB_BMX_DataRAMUserDataOffsetGet();
userProgOffset = PLIB_BMX_DataRAMUserProgramOffsetGet();

kernDataPart = kernProgOffset;
kernProgPart = userDataOffset - kernProgOffset;
userDataPart = userProgOffset - userDataOffset;
userProgPart = totalRamSize - userProgOffset;

```

5.6.3.4.3 Configuring the Library

The library is configured for the supported BMX module when the processor is chosen in the MPLAB IDE.

5.6.3.5 Library Interface

Data Types and Constants

	Name	Description
	<u>BMX_MODULE_ID</u>	Lists the possible Module IDs for the bus matrix.
	<u>PLIB_BMX_ARB_MODE</u>	Lists the possible arbitration modes for the bus matrix.
	<u>PLIB_BMX_DATA_RAM_WAIT_STATES</u>	Defines the number of data RAM wait states for address setup.
	<u>PLIB_BMX_EXCEPTION_SRC</u>	Defines which events trigger a bus exception.
	<u>PLIB_BMX_DRM_BLOCK_SIZE</u>	Defines the minimum partition block size in data RAM memory.
	<u>PLIB_BMX_PFM_BLOCK_SIZE</u>	Defines the minimum partition block size in program Flash memory.

Arbiter Routines

	Name	Description
≡	PLIB_BMX_ArbitrationModeGet	Returns the bus matrix arbitration mode.
≡	PLIB_BMX_ArbitrationModeSet	Sets the bus matrix arbitration mode.

Bus Exception Routines

	Name	Description
≡	PLIB_BMX_BusExceptionDataDisable	Disables the data bus exception.
≡	PLIB_BMX_BusExceptionDataEnable	Enables the Data bus exception.
≡	PLIB_BMX_BusExceptionDMADisable	Disables the DMA bus exception.
≡	PLIB_BMX_BusExceptionDMAEnable	Enables the DMA bus exception.
≡	PLIB_BMX_BusExceptionICDDisable	Disables the ICD bus exception.
≡	PLIB_BMX_BusExceptionICDEnable	Enables the ICD bus exception.
≡	PLIB_BMX_BusExceptionInstructionDisable	Disables the instruction bus exception.
≡	PLIB_BMX_BusExceptionInstructionEnable	Enables the instruction bus exception.
≡	PLIB_BMX_BusExceptionIXIDisable	Disables the IXI bus exception.
≡	PLIB_BMX_BusExceptionIXIEnable	Enables the IXI bus exception.

Feature Existence Routines

	Name	Description
≡	PLIB_BMX_ExistsArbitrationMode	Identifies that the ArbitrationMode feature exists on the BMX module.
≡	PLIB_BMX_ExistsBusExceptionData	Identifies that the BusExceptionData feature exists on the BMX module.
≡	PLIB_BMX_ExistsBusExceptionDMA	Identifies that the BusExceptionDMA feature exists on the BMX module.
≡	PLIB_BMX_ExistsBusExceptionICD	Identifies that the BusExceptionICD feature exists on the BMX module.
≡	PLIB_BMX_ExistsBusExceptionInstruction	Identifies that the BusExceptionInstruction feature exists on the BMX module.
≡	PLIB_BMX_ExistsBusExceptionIXI	Identifies that the BusExceptionIXI feature exists on the BMX module.
≡	PLIB_BMX_ExistsDataRAMPartition	Identifies that the DataRAMPartition feature exists on the BMX module.
≡	PLIB_BMX_ExistsDataRAMSize	Identifies that the DataRAMSize feature exists on the BMX module.
≡	PLIB_BMX_ExistsDataRamWaitState	Identifies that the DataRamWaitState feature exists on the BMX module.
≡	PLIB_BMX_ExistsProgramFlashBootSize	Identifies that the ProgramFlashBootSize feature exists on the BMX module.
≡	PLIB_BMX_ExistsProgramFlashMemoryCacheDma	Identifies that the ProgramFlashMemoryCacheDma feature exists on the BMX module.
≡	PLIB_BMX_ExistsProgramFlashMemorySize	Identifies that the ProgramFlashMemorySize feature exists on the BMX module.
≡	PLIB_BMX_ExistsProgramFlashPartition	Identifies that the ProgramFlashPartition feature exists on the BMX module.

Memory Access Control Routines

	Name	Description
≡	PLIB_BMX_DataRAMKernelProgramOffsetGet	Gets the offset (from start of RAM) of the kernel program partition.
≡	PLIB_BMX_DataRAMPartitionSet	Sets the size of the kernel and user partitions in data RAM memory.
≡	PLIB_BMX_DataRAMSizeGet	Gets the size of data RAM memory.
≡	PLIB_BMX_DataRAMUserDataOffsetGet	Gets the offset (from start of RAM) of the user data partition.
≡	PLIB_BMX_DataRAMUserProgramOffsetGet	Gets the offset (from start of RAM) of the user program partition.
≡	PLIB_BMX_DataRamWaitStateGet	Returns the number of data RAM Wait states.
≡	PLIB_BMX_DataRamWaitStateSet	Sets the number of data RAM Wait states.
≡	PLIB_BMX_ProgramFlashBootSizeGet	Gets the size of boot program Flash memory.
≡	PLIB_BMX_ProgramFlashMemorySizeGet	Gets the size of program Flash memory.
≡	PLIB_BMX_ProgramFlashPartitionGet	Gets the size of the kernel partition in program Flash memory.
≡	PLIB_BMX_ProgramFlashPartitionSet	Sets the size of the kernel and user partitions in program Flash memory.

Program Flash Cache Routines

	Name	Description
≡	PLIB_BMX_ProgramFlashMemoryCacheDmaDisable	Disables the bus matrix program Flash cacheability for DMA.
≡	PLIB_BMX_ProgramFlashMemoryCacheDmaEnable	Enables the bus matrix program Flash cacheability for DMA.

Description

The BMX peripheral library consists of a set of interface routines, data types and constants, and macros provided by the [plib_bmx.h](#) header file. This section describes those items.

5.6.3.5.1 Bus Exception Routines**5.6.3.5.1.1 PLIB_BMX_BusExceptionDataDisable Function****C**

```
void PLIB_BMX_BusExceptionDataDisable(
    BMX_MODULE_ID index
);
```

Description

This function disables the data bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionData feature. This feature may not be available on all devices. Please

refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionData](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionDataDisable(BMX_ID_0);
```

5.6.3.5.1.2 PLIB_BMX_BusExceptionDataEnable Function**C**

```
void PLIB_BMX_BusExceptionDataEnable(  
    BMX_MODULE_ID index  
);
```

Description

This function enables the Data bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionData](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionDataEnable(BMX_ID_0);
```

5.6.3.5.1.3 PLIB_BMX_BusExceptionDMADisable Function**C**

```
void PLIB_BMX_BusExceptionDMADisable(  
    BMX_MODULE_ID index  
);
```

Description

This function disables the DMA bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionDMA feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionDMA](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionDMADisable(BMX_ID_0);
```

5.6.3.5.1.4 PLIB_BMX_BusExceptionDMAEnable Function**C**

```
void PLIB_BMX_BusExceptionDMAEnable(  
    BMX_MODULE_ID index  
);
```

Description

This function enables the DMA bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionDMA feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionDMA](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionDMAEnable(BMX_ID_0);
```

5.6.3.5.1.5 PLIB_BMX_BusExceptionICDDisable Function**C**

```
void PLIB_BMX_BusExceptionICDDisable(  
    BMX_MODULE_ID index  
);
```

Description

This function disables the ICD bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionICD feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionICD](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionICDDisable(BMX_ID_0);
```

5.6.3.5.1.6 PLIB_BMX_BusExceptionICDEnable Function**C**

```
void PLIB_BMX_BusExceptionICDEnable(  
    BMX_MODULE_ID index  
);
```

Description

This function enables the ICD bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionICD feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionICD](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionICDEnable(BMX_ID_0);
```

5.6.3.5.1.7 PLIB_BMX_BusExceptionInstructionDisable Function**C**

```
void PLIB_BMX_BusExceptionInstructionDisable(  
    BMX_MODULE_ID index  
);
```

Description

This function disables the instruction bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionInstruction feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionInstruction](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionInstructionDisable(BMX_ID_0);
```

5.6.3.5.1.8 PLIB_BMX_BusExceptionInstructionEnable Function**C**

```
void PLIB_BMX_BusExceptionInstructionEnable(  
    BMX_MODULE_ID index  
);
```

Description

This function enables the instruction bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionInstruction feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionInstruction](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionInstructionEnable(BMX_ID_0);
```

5.6.3.5.1.9 PLIB_BMX_BusExceptionIXIDisable Function**C**

```
void PLIB_BMX_BusExceptionIXIDisable(  
    BMX_MODULE_ID index  
);
```

Description

This function disables the IXI bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionIXI feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionIXI](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionIXIDisable(BMX_ID_0);
```

5.6.3.5.1.10 PLIB_BMX_BusExceptionIXIEnable Function**C**

```
void PLIB_BMX_BusExceptionIXIEnable(  
    BMX_MODULE_ID index  
);
```

Description

This function enables the IXI bus exception.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Returns

None.

Remarks

This function implements an operation of the BusExceptionIXI feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsBusExceptionIXI](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_BMX_BusExceptionIXIEnable(BMX_ID_0);
```

5.6.3.5.2 Program Flash Cache Routines**5.6.3.5.2.1 PLIB_BMX_ProgramFlashMemoryCacheDmaDisable Function****C**

```
void PLIB_BMX_ProgramFlashMemoryCacheDmaDisable(  
    BMX_MODULE_ID index
```

```
);
```

Description

This function disables the program Flash cacheability for DMA.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Returns

None.

Remarks

This function implements an operation of the ProgramFlashMemoryCacheDma feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsProgramFlashMemoryCacheDma](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_BMX_ProgramFlashMemoryCacheDmaDisable(BMX_ID_0);
```

5.6.3.5.2.2 PLIB_BMX_ProgramFlashMemoryCacheDmaEnable Function

C

```
void PLIB_BMX_ProgramFlashMemoryCacheDmaEnable(  
    BMX_MODULE_ID index  
);
```

Description

This function enables the program Flash cacheability for DMA.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Returns

None.

Remarks

This function implements an operation of the ProgramFlashMemoryCacheDma feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsProgramFlashMemoryCacheDma](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_BMX_ProgramFlashMemoryCacheDmaEnable(BMX_ID_0);
```

5.6.3.5.3 Arbiter Routines

5.6.3.5.3.1 PLIB_BMX_ArbitrationModeGet Function

C

```
PLIB_BMX_ARB_MODE PLIB_BMX_ArbitrationModeGet(  
    BMX_MODULE_ID index  
);
```

Description

This function returns the bus matrix arbitration mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

[PLIB_BMX_ARB_MODE](#) enumerator value representing the arbitration mode.

Remarks

This function implements an operation of the ArbitrationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsArbitrationMode](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_BMX_ARB_MODE mode;  
mode = PLIB_BMX_ArbitrationModeGet(BMX_ID_0);
```

5.6.3.5.3.2 PLIB_BMX_ArbitrationModeSet Function

C

```
void PLIB_BMX_ArbitrationModeSet(  
    BMX_MODULE_ID index,  
    PLIB_BMX_ARB_MODE mode  
);
```

Description

This function sets the bus matrix arbitration mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Identifies the desired arbitration mode

Returns

None.

Remarks

This function implements an operation of the ArbitrationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsArbitrationMode](#) in your application to to

automatically determine whether this feature is available.

Example

```
PLIB_BMX_ArbitrationModeSet(BMX_ID_0, PLIB_BMX_ARB_MODE_DMA);
```

5.6.3.5.4 Memory Access Control Routines

5.6.3.5.4.1 PLIB_BMX_DataRAMKernelProgramOffsetGet Function

C

```
size_t PLIB_BMX_DataRAMKernelProgramOffsetGet(  
    BMX_MODULE_ID index  
);
```

Description

This function returns the offset of start address of the kernel program RAM partition. This represents the size of the kernel data RAM partition.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Offset of kernel program partition from base of RAM.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsDataRAMPartition](#) in your application to automatically determine whether this feature is available.

Example

```
size_t kernProgOffset;  
kernProgOffset = PLIB_BMX_DataRAMKernelProgramOffsetGet(BMX_ID_0);
```

5.6.3.5.4.2 PLIB_BMX_DataRAMPartitionSet Function

C

```
void PLIB_BMX_DataRAMPartitionSet(  
    BMX_MODULE_ID index,  
    size_t kernProgOffset,  
    size_t userDataOffset,  
    size_t userProgOffset  
);
```

Description

This function sets the size of the kernel and user partitions in the data RAM memory (DRM). By default, the entire data RAM is mapped to Kernel mode and all of the offsets are zero. To partition the data RAM, all of the offsets must be set to a minimum of one DRM block size. If any of the offsets are set to zero, the entire data RAM is allocated to kernel data. The partitions must be a multiple of [PLIB_BMX_DRM_BLOCK_SIZE](#). If programmed with a value that is not a multiple of [PLIB_BMX_DRM_BLOCK_SIZE](#), the value will be automatically truncated to [PLIB_BMX_DRM_BLOCK_SIZE](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
kernProgOffset	Size of the offset of the Kernel Program partition
userDataOffset	Size of the offset of the User Data partition
userProgOffset	Size of the offset of the User Program partition

Returns

None.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsDataRAMPartition](#) in your application to automatically determine whether this feature is available.

Example

```
//Total Data RAM = 32 KB.
size_t totalRamSize;
size_t kernDataRamSize = (3 * PLIB_BMX_DRM_BLOCK_SIZE);
size_t kernProgRamSize = (2 * PLIB_BMX_DRM_BLOCK_SIZE);
size_t userDataRamSize = (6 * PLIB_BMX_DRM_BLOCK_SIZE);
size_t userProgRamSize = (5 * PLIB_BMX_DRM_BLOCK_SIZE);

//Get Size of Data RAM
totalRamSize = PLIB_BMX_DataRAMSizeGet(BMX_ID_0);

//Verify our partition sizes fit our RAM
if ((kernDataRamSize + kernProgRamSize + userDataRamSize +
    userProgRamSize) != totalRamSize)
{
    printf("RAM Partitioning Error\n");
}

size_t kernProgOffset;
size_t userDataOffset;
size_t userProgOffset;

kernProgOffset = kernDataRamSize;
userDataOffset = kernDataRamSize + kernProgRamSize;
userProgOffset = userDataOffset + userDataRamSize;

PLIB_BMX_DataRAMPartitionSet(BMX_ID_0, kernProgOffset,
    userDataOffset, userProgOffset);
```

5.6.3.5.4.3 PLIB_BMX_DataRAMSizeGet Function**C**

```
size_t PLIB_BMX_DataRAMSizeGet(
    BMX_MODULE_ID index
);
```

Description

This function returns the size of the data RAM memory.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Size of data RAM memory.

Remarks

This function implements an operation of the DataRAMSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsDataRAMSize](#) in your application to automatically determine whether this feature is available.

Example

```
size_t drmSize;
drnSize = PLIB_BMX_DataRAMSizeGet(BMX_ID_0);
```

5.6.3.5.4.4 PLIB_BMX_DataRAMUserDataOffsetGet Function**C**

```
size_t PLIB_BMX_DataRAMUserDataOffsetGet(
    BMX_MODULE_ID index
);
```

Description

This function returns the offset of start address of the user data RAM partition. Subtracting the kernel program offset from the user data offset gives the size of the kernel program RAM partition.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Offset of user data partition from base of RAM.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsDataRAMPartition](#) in your application to automatically determine whether this feature is available.

Example

```
size_t userDataOffset;
userDataOffset = PLIB_BMX_DataRAMUserDataOffsetGet(BMX_ID_0);
```

5.6.3.5.4.5 PLIB_BMX_DataRAMUserProgramOffsetGet Function**C**

```
size_t PLIB_BMX_DataRAMUserProgramOffsetGet(
    BMX_MODULE_ID index
);
```

```
);
```

Description

This function returns the offset of start address of the user program RAM partition. Subtracting the user data offset from the user program offset gives the size of the user data RAM partition.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Offset of user data partition from base of RAM.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsDataRAMPartition](#) in your application to automatically determine whether this feature is available.

Example

```
size_t userProgOffset;  
userProgOffset = PLIB_BMX_DataRAMUserProgramOffsetGet(BMX_ID_0);
```

5.6.3.5.4.6 PLIB_BMX_DataRamWaitStateGet Function**C**

```
PLIB_BMX_DATA_RAM_WAIT_STATES PLIB_BMX_DataRamWaitStateGet(  
    BMX_MODULE_ID index  
);
```

Description

This function returns the number of data RAM Wait states.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

[PLIB_BMX_DATA_RAM_WAIT_STATES](#) enumeration representing the number of wait states.

Remarks

This function implements an operation of the DataRamWaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsDataRamWaitState](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_BMX_DATA_RAM_WAIT_STATES wait;  
wait = PLIB_BMX_DataRamWaitStateGet(BMX_ID_0);
```

5.6.3.5.4.7 PLIB_BMX_DataRamWaitStateSet Function

C

```
void PLIB_BMX_DataRamWaitStateSet(  
    BMX_MODULE_ID index,  
    PLIB_BMX_DATA_RAM_WAIT_STATES wait  
);
```

Description

This function sets the number of data RAM Wait states.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
PLIB_BMX_DATA_RAM_WAIT_STATES	Enumeration representing the number of Wait states

Returns

None.

Remarks

This function implements an operation of the DataRamWaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsDataRamWaitState](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_BMX_DataRamWaitStateSet(BMX_ID_0, PLIB_BMX_DATA_RAM_WAIT_ONE);
```

5.6.3.5.4.8 PLIB_BMX_ProgramFlashBootSizeGet Function

C

```
size_t PLIB_BMX_ProgramFlashBootSizeGet(  
    BMX_MODULE_ID index  
);
```

Description

This function returns the size of the boot program Flash memory.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Size of boot program Flash memory.

Remarks

This function implements an operation of the ProgramFlashBootSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsProgramFlashBootSize](#) in your application to to automatically determine whether this feature is available.

Example

```
size_t bootSize;  
bootSize = PLIB_BMX_ProgramFlashBootSizeGet(BMX_ID_0);
```

5.6.3.5.4.9 PLIB_BMX_ProgramFlashMemorySizeGet Function**C**

```
size_t PLIB_BMX_ProgramFlashMemorySizeGet(  
    BMX_MODULE_ID index  
);
```

Description

This function returns the size of the program Flash memory.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Size of program Flash memory.

Remarks

This function implements an operation of the ProgramFlashMemorySize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsProgramFlashMemorySize](#) in your application to automatically determine whether this feature is available.

Example

```
size_t pfmSize;  
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet(BMX_ID_0);
```

5.6.3.5.4.10 PLIB_BMX_ProgramFlashPartitionGet Function**C**

```
size_t PLIB_BMX_ProgramFlashPartitionGet(  
    BMX_MODULE_ID index  
);
```

Description

This function gets the size of the kernel partition in the program Flash memory. The remaining Flash is set to user mode, and may be accessed by user programs. On reset, the entire Flash is mapped to Kernel mode, and this function will return zero.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Size of the kernel partition in program Flash memory.

Remarks

This function implements an operation of the ProgramFlashPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsProgramFlashPartition](#) in your application to automatically determine whether this feature is available.

Example

```
size_t pfmSize;
size_t userPfmSize;
size_t kernPfmSize;

// Get size of PFM
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet(BMX_ID_0);
kernPfmSize = PLIB_BMX_ProgramFlashPartitionGet(BMX_ID_0);
userPfmSize = pfmSize - kernPfmSize;
```

5.6.3.5.4.11 PLIB_BMX_ProgramFlashPartitionSet Function

C

```
void PLIB_BMX_ProgramFlashPartitionSet(
    BMX_MODULE_ID index,
    size_t user_size
);
```

Description

This function sets the size of the kernel and user partitions in program Flash memory (PFM). Kernel programs may access both partitions, but user programs may not access the kernel partition (including peripheral registers). By default, the entire Flash is mapped to Kernel mode. If called with a non-zero value, a user partition of size `user_size` is created, and the remaining PFM remains in Kernel mode. The user partition must be a multiple of [PLIB_BMX_PFM_BLOCK_SIZE](#). If programmed with a value that is not a multiple of [PLIB_BMX_PFM_BLOCK_SIZE](#), the value will be truncated to a [PLIB_BMX_PFM_BLOCK_SIZE](#) boundary.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
userSize	Size of the user partition in PFM

Returns

None.

Remarks

This function implements an operation of the ProgramFlashPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_BMX_ExistsProgramFlashPartition](#) in your application to automatically determine whether this feature is available.

Example

```
size_t pfm_size;
size_t userSize = (6 * PLIB_BMX_PFM_BLOCK_SIZE);
size_t userOffset;

// Get size of PFM
size_t pfmSize;
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet(BMX_ID_0);
userOffset = pfmSize - userSize;
```

```
if (userOffset > 0)
{
    PLIB_BMX_ProgramFlashPartitionSet(BMX_ID_0, userOffset);
}
```

5.6.3.5.5 Data Types and Constants

5.6.3.5.5.1 BMX_MODULE_ID Enumeration

C

```
typedef enum {
    BMX_ID_0,
    BMX_NUMBER_OF_MODULES
} BMX_MODULE_ID;
```

Description

Module ID

This enumeration lists the possible Module IDs for the bus matrix.

Remarks

Refer to the specific device data sheet to obtain the correct number of modules defined for the desired device.

5.6.3.5.5.2 PLIB_BMX_ARB_MODE Enumeration

C

```
typedef enum {
    PLIB_BMX_ARB_MODE_INST,
    PLIB_BMX_ARB_MODE_DMA,
    PLIB_BMX_ARB_MODE_ROT
} PLIB_BMX_ARB_MODE;
```

Description

Arbitration Mode

This enumeration lists the possible arbitration modes for the bus matrix.

Members

Members	Description
PLIB_BMX_ARB_MODE_INST	Arbitration Mode 0
PLIB_BMX_ARB_MODE_DMA	Arbitration Mode 1
PLIB_BMX_ARB_MODE_ROT	Arbitration Mode 2

5.6.3.5.5.3 PLIB_BMX_DATA_RAM_WAIT_STATES Enumeration

C

```
typedef enum {
    PLIB_BMX_DATA_RAM_WAIT_ZERO,
    PLIB_BMX_DATA_RAM_WAIT_ONE
} PLIB_BMX_DATA_RAM_WAIT_STATES;
```

Description

Wait States

This definition specifies the number of data RAM wait states for address setup.

Members

Members	Description
PLIB_BMX_DATA_RAM_WAIT_ZERO	Zero wait states for address setup
PLIB_BMX_DATA_RAM_WAIT_ONE	One wait state for address setup

5.6.3.5.5.4 PLIB_BMX_EXCEPTION_SRC Enumeration**C**

```
typedef enum {
    PLIB_BMX_ERR_IXI,
    PLIB_BMX_ERR_ICD,
    PLIB_BMX_ERR_DMA,
    PLIB_BMX_ERR_DATA,
    PLIB_BMX_ERR_INST
} PLIB_BMX_EXCEPTION_SRC;
```

Description

Exception Bits

This definition specifies which events trigger a bus exception.

Members

Members	Description
PLIB_BMX_ERR_IXI	IXI Shared Bus
PLIB_BMX_ERR_ICD	In-Circuit Debugger
PLIB_BMX_ERR_DMA	DMA Controller
PLIB_BMX_ERR_DATA	CPU Data Bus
PLIB_BMX_ERR_INST	CPU Instruction Bus

5.6.3.5.5.5 PLIB_BMX_DRM_BLOCK_SIZE Macro**C**

```
#define PLIB_BMX_DRM_BLOCK_SIZE 2048
```

Description

Program Flash Partition Block Size

This definition specifies the minimum partition block size in data RAM memory.

5.6.3.5.5.6 PLIB_BMX_PFM_BLOCK_SIZE Macro**C**

```
#define PLIB_BMX_PFM_BLOCK_SIZE 2048
```

Description

Program Flash Partition Block Size

This definition specifies the minimum partition block size in program Flash memory.

5.6.3.5.6 Feature Existence Routines

5.6.3.5.6.1 PLIB_BMX_ExistsArbitrationMode Function

C

```
bool PLIB_BMX_ExistsArbitrationMode(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the ArbitrationMode feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_ArbitrationModeSet](#)
- [PLIB_BMX_ArbitrationModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ArbitrationMode feature is supported on the device
- false = The ArbitrationMode feature is not supported on the device

Remarks

None.

5.6.3.5.6.2 PLIB_BMX_ExistsBusExceptionData Function

C

```
bool PLIB_BMX_ExistsBusExceptionData(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the BusExceptionData feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_BusExceptionDataEnable](#)
- [PLIB_BMX_BusExceptionDataDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The BusExceptionData feature is supported on the device
- false = The BusExceptionData feature is not supported on the device

Remarks

None.

5.6.3.5.6.3 PLIB_BMX_ExistsBusExceptionDMA Function**C**

```
bool PLIB_BMX_ExistsBusExceptionDMA(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the BusExceptionDMA feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_BusExceptionDMAEnable](#)
- [PLIB_BMX_BusExceptionDMADisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The BusExceptionDMA feature is supported on the device
- false = The BusExceptionDMA feature is not supported on the device

Remarks

None.

5.6.3.5.6.4 PLIB_BMX_ExistsBusExceptionICD Function**C**

```
bool PLIB_BMX_ExistsBusExceptionICD(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the BusExceptionICD feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_BusExceptionICDEnable](#)
- [PLIB_BMX_BusExceptionICDDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The BusExceptionICD feature is supported on the device
- false = TheWhen BusExceptionICD feature is not supported on the device

Remarks

None.

5.6.3.5.6.5 PLIB_BMX_ExistsBusExceptionInstruction Function**C**

```
bool PLIB_BMX_ExistsBusExceptionInstruction(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the BusExceptionInstruction feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_BusExceptionInstructionEnable](#)
- [PLIB_BMX_BusExceptionInstructionDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The BusExceptionInstruction feature is supported on the device
- false = The BusExceptionInstruction feature is not supported on the device

Remarks

None.

5.6.3.5.6.6 PLIB_BMX_ExistsBusExceptionIXI Function**C**

```
bool PLIB_BMX_ExistsBusExceptionIXI(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the BusExceptionIXI feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_BusExceptionIXIEnable](#)
- [PLIB_BMX_BusExceptionIXIDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The BusExceptionIXI feature is supported on the device
- false = The BusExceptionIXI feature is not supported on the device

Remarks

None.

5.6.3.5.6.7 PLIB_BMX_ExistsDataRAMPartition Function**C**

```
bool PLIB_BMX_ExistsDataRAMPartition(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the DataRAMPartition feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_DataRAMPartitionSet](#)
- [PLIB_BMX_DataRAMKernelProgramOffsetGet](#)
- [PLIB_BMX_DataRAMUserDataOffsetGet](#)
- [PLIB_BMX_DataRAMUserProgramOffsetGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The DataRAMPartition feature is supported on the device
- false = The DataRAMPartition feature is not supported on the device

Remarks

None.

5.6.3.5.6.8 PLIB_BMX_ExistsDataRAMSize Function**C**

```
bool PLIB_BMX_ExistsDataRAMSize(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the DataRAMSize feature is available on the BMX module. When this interface returns true, this function is supported on the device:

- [PLIB_BMX_DataRAMSizeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The DataRAMSize feature is supported on the device
- false = The DataRAMSize feature is not supported on the device

Remarks

None.

5.6.3.5.6.9 PLIB_BMX_ExistsDataRamWaitState Function**C**

```
bool PLIB_BMX_ExistsDataRamWaitState(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the DataRamWaitState feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_DataRamWaitStateSet](#)
- [PLIB_BMX_DataRamWaitStateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The DataRamWaitState feature is supported on the device
- false = The DataRamWaitState feature is not supported on the device

Remarks

None.

5.6.3.5.6.10 PLIB_BMX_ExistsProgramFlashBootSize Function**C**

```
bool PLIB_BMX_ExistsProgramFlashBootSize(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the ProgramFlashBootSize feature is available on the BMX module. When this interface returns true, this function is supported on the device:

- [PLIB_BMX_ProgramFlashBootSizeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ProgramFlashBootSize feature is supported on the device
- false = The ProgramFlashBootSize feature is not supported on the device

Remarks

None.

5.6.3.5.6.11 PLIB_BMX_ExistsProgramFlashMemoryCacheDma Function**C**

```
bool PLIB_BMX_ExistsProgramFlashMemoryCacheDma(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the ProgramFlashMemoryCacheDma feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_ProgramFlashMemoryCacheDmaEnable](#)
- [PLIB_BMX_ProgramFlashMemoryCacheDmaDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ProgramFlashMemoryCacheDma feature is supported on the device
- false = The ProgramFlashMemoryCacheDma feature is not supported on the device

Remarks

None.

5.6.3.5.6.12 PLIB_BMX_ExistsProgramFlashMemorySize Function**C**

```
bool PLIB_BMX_ExistsProgramFlashMemorySize(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the ProgramFlashMemorySize feature is available on the BMX module. When this interface returns true, this function is supported on the device:

- [PLIB_BMX_ProgramFlashMemorySizeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ProgramFlashMemorySize feature is supported on the device
- false = The ProgramFlashMemorySize feature is not supported on the device

Remarks

None.

5.6.3.5.6.13 PLIB_BMX_ExistsProgramFlashPartition Function

C

```
bool PLIB_BMX_ExistsProgramFlashPartition(  
    BMX_MODULE_ID index  
);
```

Description

This interface identifies that the ProgramFlashPartition feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- [PLIB_BMX_ProgramFlashPartitionGet](#)
- [PLIB_BMX_ProgramFlashPartitionSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ProgramFlashPartition feature is supported on the device
- false = The ProgramFlashPartition feature is not supported on the device

Remarks

None.

5.6.3.6 Files

Files

Name	Description
plib_bmx.h	Defines the Bus Matrix (BMX) Peripheral Library interface

Description

5.6.3.6.1 plib_bmx.h

Bus Matrix Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the bus matrix (BMX) peripheral library (PLIB) for Microchip microcontrollers. The definitions in this file are for the bus matrix controller module.

Enumerations

	Name	Description
	<u>PLIB_BMX_ARB_MODE</u>	Lists the possible arbitration modes for the bus matrix.
	<u>PLIB_BMX_DATA_RAM_WAIT_STATES</u>	Defines the number of data RAM wait states for address setup.
	<u>PLIB_BMX_EXCEPTION_SRC</u>	Defines which events trigger a bus exception.

Functions

	Name	Description
	<u>PLIB_BMX_ArbitrationModeGet</u>	Returns the bus matrix arbitration mode.
	<u>PLIB_BMX_ArbitrationModeSet</u>	Sets the bus matrix arbitration mode.
	<u>PLIB_BMX_BusExceptionDataDisable</u>	Disables the data bus exception.
	<u>PLIB_BMX_BusExceptionDataEnable</u>	Enables the Data bus exception.
	<u>PLIB_BMX_BusExceptionDMADisable</u>	Disables the DMA bus exception.
	<u>PLIB_BMX_BusExceptionDMAEnable</u>	Enables the DMA bus exception.
	<u>PLIB_BMX_BusExceptionICDDisable</u>	Disables the ICD bus exception.
	<u>PLIB_BMX_BusExceptionICDEnable</u>	Enables the ICD bus exception.
	<u>PLIB_BMX_BusExceptionInstructionDisable</u>	Disables the instruction bus exception.
	<u>PLIB_BMX_BusExceptionInstructionEnable</u>	Enables the instruction bus exception.
	<u>PLIB_BMX_BusExceptionIXIDisable</u>	Disables the IXI bus exception.
	<u>PLIB_BMX_BusExceptionIXIEnable</u>	Enables the IXI bus exception.
	<u>PLIB_BMX_DataRAMKernelProgramOffsetGet</u>	Gets the offset (from start of RAM) of the kernel program partition.
	<u>PLIB_BMX_DataRAMPartitionSet</u>	Sets the size of the kernel and user partitions in data RAM memory.
	<u>PLIB_BMX_DataRAMSizeGet</u>	Gets the size of data RAM memory.
	<u>PLIB_BMX_DataRAMUserDataOffsetGet</u>	Gets the offset (from start of RAM) of the user data partition.
	<u>PLIB_BMX_DataRAMUserProgramOffsetGet</u>	Gets the offset (from start of RAM) of the user program partition.
	<u>PLIB_BMX_DataRamWaitStateGet</u>	Returns the number of data RAM Wait states.
	<u>PLIB_BMX_DataRamWaitStateSet</u>	Sets the number of data RAM Wait states.
	<u>PLIB_BMX_ExistsArbitrationMode</u>	Identifies that the ArbitrationMode feature exists on the BMX module.
	<u>PLIB_BMX_ExistsBusExceptionData</u>	Identifies that the BusExceptionData feature exists on the BMX module.
	<u>PLIB_BMX_ExistsBusExceptionDMA</u>	Identifies that the BusExceptionDMA feature exists on the BMX module.
	<u>PLIB_BMX_ExistsBusExceptionICD</u>	Identifies that the BusExceptionICD feature exists on the BMX module.
	<u>PLIB_BMX_ExistsBusExceptionInstruction</u>	Identifies that the BusExceptionInstruction feature exists on the BMX module.

	PLIB_BMX_ExistsBusExceptionIXI	Identifies that the BusExceptionIXI feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMPartition	Identifies that the DataRAMPartition feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMSize	Identifies that the DataRAMSize feature exists on the BMX module.
	PLIB_BMX_ExistsDataRamWaitState	Identifies that the DataRamWaitState feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashBootSize	Identifies that the ProgramFlashBootSize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashMemoryCacheDma	Identifies that the ProgramFlashMemoryCacheDma feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashMemorySize	Identifies that the ProgramFlashMemorySize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashPartition	Identifies that the ProgramFlashPartition feature exists on the BMX module.
	PLIB_BMX_ProgramFlashBootSizeGet	Gets the size of boot program Flash memory.
	PLIB_BMX_ProgramFlashMemoryCacheDmaDisable	Disables the bus matrix program Flash cacheability for DMA.
	PLIB_BMX_ProgramFlashMemoryCacheDmaEnable	Enables the bus matrix program Flash cacheability for DMA.
	PLIB_BMX_ProgramFlashMemorySizeGet	Gets the size of program Flash memory.
	PLIB_BMX_ProgramFlashPartitionGet	Gets the size of the kernel partition in program Flash memory.
	PLIB_BMX_ProgramFlashPartitionSet	Sets the size of the kernel and user partitions in program Flash memory.

Macros

	Name	Description
	PLIB_BMX_DRM_BLOCK_SIZE	Defines the minimum partition block size in data RAM memory.
	PLIB_BMX_PFM_BLOCK_SIZE	Defines the minimum partition block size in program Flash memory.

File Name

plib_bmx.h

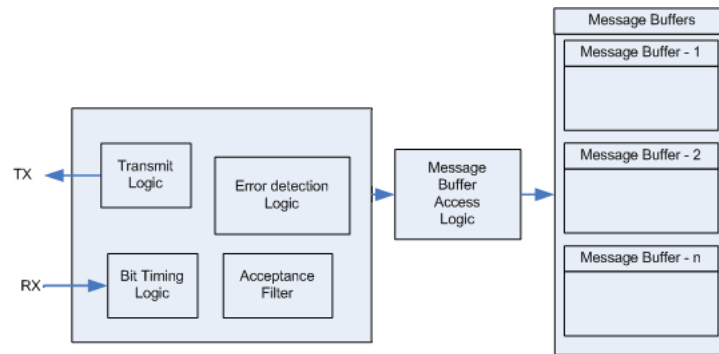
Company

Microchip Technology Inc.

5.6.4 CAN Peripheral Library

5.6.4.1 Introduction

CAN Peripheral Library
for
Microchip Microcontrollers



The Controller Area Network (CAN) is used primarily in industrial and automotive applications. It is an asynchronous serial data communication protocol that provides reliable communication in an electrically noisy environment.

This library provides a low-level abstraction of the CAN module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software.

5.6.4.2 Release Notes

MPLAB Harmony Version: v0.70b **CAN Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.4.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

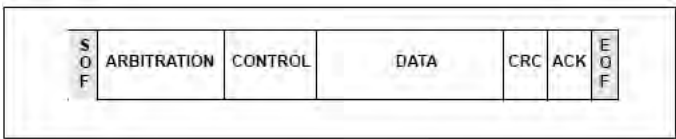
5.6.4.4 How the Library Works

Note: Not all modes are available on all the devices. Please refer to the specific device data sheet to determine the set of modes that are supported for your device.

5.6.4.4.1 About CAN Protocol

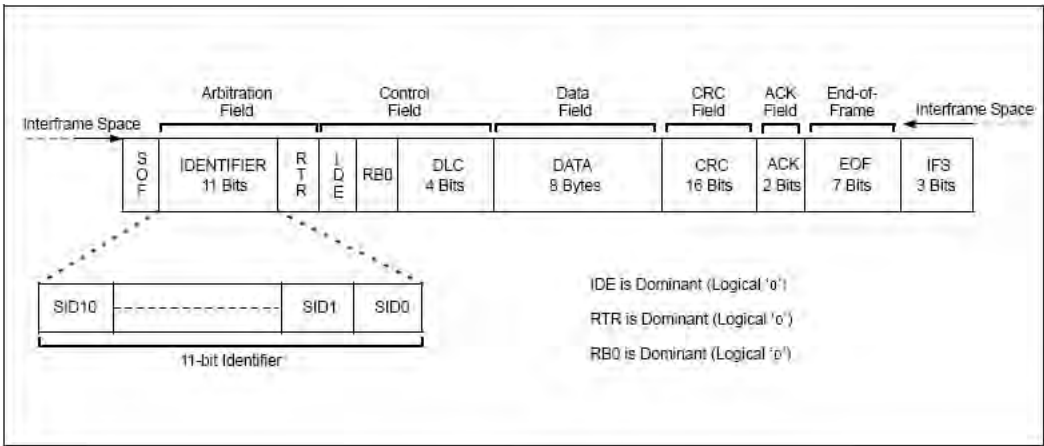
The following section briefly discusses the CAN Protocol briefly. For complete details on the CAN protocol, refer to the CAN 2.0 specification available from Bosch.

The CAN bus protocol uses asynchronous communication. Information is passed from transmitters to receivers in data frames, which are composed of byte fields that define the contents of the data frame. This is shown in the figure below.



Each frame begins with a Start-of-Frame (SOF) bit and terminates with an End-of-Frame (EOF) bit field. The SOF is followed by arbitration and control fields, which identify the message type, format, length and priority. This information allows each node on the CAN bus to respond appropriately to the message. The data field conveys the message content and is of variable length, ranging from 0 to 8 bytes. Error protection is provided by the Cyclic Redundancy Check (CRC) and acknowledgement (ACK) fields.

5.6.4.4.1.1 Standard ID Message Format



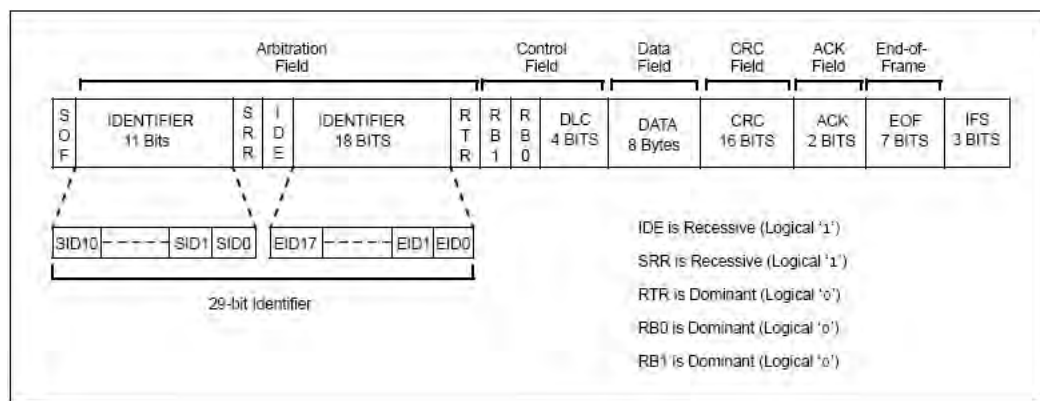
The Standard ID message begins with a Start-of-Frame bit followed by a 12-bit arbitration field. The arbitration field contains an 11-bit identifier and the Remote Transmit Request (RTR) bit. The identifier defines the type of information contained in the message and is used by each receiving node to determine if the message is of interest. The RTR bit distinguishes a data frame from a remote frame. For a standard data frame, the RTR bit is clear.

Following the arbitration field is a 6-bit control field, which provides more information about the contents of the message. The first bit in the control field is an Identifier Extension (IDE) bit, which distinguishes the message as either a standard or extended data frame. A standard data frame is indicated by a dominant state (logic level '0') during transmission of the IDE bit. The second bit in the control field is a reserved (RBO) bit, which is in the dominant state (logic level '0'). The last 4 bits in the control field represent the Data Length Code (DLC), which specifies the number of data bytes present in the message.

The data field follows the control field. This field carries the message data – the actual payload of the data frame. This field is of variable length, ranging from 0 to 8 bytes. The number of bytes is user-selectable.

The data field is followed by the Cyclic Redundancy Check (CRC) field, which is a 15-bit CRC sequence with one delimiter bit. The acknowledgement (ACK) field is sent as a recessive bit (logic level '1') and is overwritten as a dominant bit by any receiver that has received the data correctly. The message is acknowledged by the receiver regardless of the result of the acceptance filter comparison. The last field is the End-of-Frame field, which consists of 7 recessive bits that indicate the end of message.

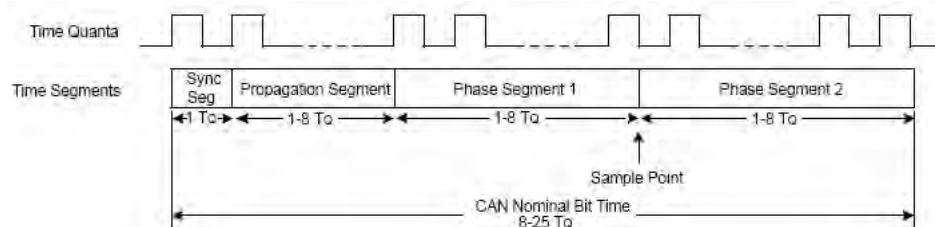
5.6.4.4.1.2 Extended ID Message Format



The Extended ID Message frame begins with an SFF bit followed by a 31-bit arbitration field. The arbitration field for the extended data frame contains 29 identifier bits in two fields separated by a Substitute Remote Request (SRR) bit and an IDE bit. The SRR bit determines if the message is a remote frame. SRR = 1 for extended data frames. The IDE bit indicates the data frame type. For the extended data frame, IDE = 1.

The extended data frame Control field consists of 7 bits. The first bit is the RTR. For the extended data frame, RTR = 0. The next two bits, RB1 and RB0, are reserved bits that are in the dominant state (logic level '0'). The last 4 bits in the Control field are the Data Length Code, which specifies the number of data bytes present in the message. The remaining fields in an extended data frame are identical to a standard data frame.

5.6.4.4.1.3 Can Bit Time Quanta



The nominal bit rate is the number of bits per second transmitted on the CAN bus.

Nominal bit time = $1 \div \text{Nominal Bit Rate}$.

There are four time segments in a bit time to compensate for any phase shifts due to oscillator drifts or propagation delays. These time segments do not overlap each other and are represented in terms of Time Quantum (TQ). One TQ is a fixed unit of time derived from the oscillator clock. The total number of time quanta in a nominal bit time must be programmed between 8 and 25 TQ.

Each bit transmission time consists of four time segments:

Synchronization Segment – This time segment synchronizes the different nodes connected on the CAN bus. A bit edge is expected to be within this segment. Based on CAN protocol, the Synchronization Segment is assumed to be one Time Quantum.

Propagation Segment – This time segment compensates for any time delay that may occur due to the bus line or due to the various transceivers connected on that bus.

Phase Segment 1 – This time segment compensates for errors that may occur due to phase shift in the edges. The time segment may be lengthened during resynchronization to compensate for the phase shift.

Phase Segment 2 – This time segment compensates for errors that may occur due to phase shift in the edges. The time segment may be shortened during resynchronization to compensate for the phase shift. The Phase Segment 2 time can be configured to be either programmable or specified by the Phase Segment 1 time.

Two types of synchronization are used – Hard Synchronization and Resynchronization. A Hard Synchronization occurs once at the start of a frame. Resynchronization occurs inside a frame.

Hard synchronization takes place on the recessive-to-dominant transition of the start bit. The bit time is restarted from that edge.

Resynchronization takes place when a bit edge does not occur within the Synchronization Segment in a message. One of the Phase Segments is shortened or lengthened by an amount that depends on the phase error in the signal. The maximum amount that can be used is determined by the Synchronization Jump Width parameter.

The length of Phase Segment 1 and Phase Segment 2 can be changed depending on oscillator tolerances of the transmitting and receiving node. Resynchronization compensates for any phase shifts that may occur due to the different oscillators used by the transmitting and receiving nodes.

Bit Lengthening – If the transmitting node in CAN has a slower oscillator than the receiving node, the next falling edge, and therefore, the sample point, can be delayed by lengthening Phase Segment 1 in the bit time.

Bit Shortening – If the transmitting node in CAN has a faster oscillator than the receiving node, then the next falling edge, and therefore, the sample point of the next bit, can be reduced by shortening the Phase Segment 2 in the bit time.

Synchronization Jump Width (SJW) – This determines the synchronization jump width by limiting the amount of lengthening or shortening that can be applied to the Phase Segment 1 and Phase Segment 2 time intervals. This segment should not be longer than Phase Segment 2 time. The width can be 1-4 TQ

5.6.4.4.2 Initialization of CAN

5.6.4.4.2.1 Mode Configuration

The CAN module must be enabled before configuration. Enabling the CAN module gives it control over the CAN pins of the device. The following code example shows how the `PLIB_CAN_Enable` function is used to enable the module.

```
/* This code expects the data direction settings to be done from other modules */  
/* Switch the CAN module ON */  
PLIB_CAN_Enable(CAN_ID_1);
```

The CAN module must be placed in Configuration mode before attempting to configure it. The following features are available in Configuration mode only:

- Configuring the CAN module clock and CAN bus speed
- Configuring the message acceptance filter mask
- Assigning the channel buffer memory
- Configuring channel size and Receive Channel Data-only mode.

The CAN module operating mode can be changed by using the `PLIB_CAN_OperationModeSelect` function. The caller should

wait until the operating mode is set before proceeding. The `PLIB_CAN_OperationModeGet` function can be used to obtain the current operating mode of the CAN module.

```
/* Switch the CAN module to Configuration mode. Wait until the switch is complete */
PLIB_CAN_Enable(CAN_ID_1);
PLIB_CAN_OperationModeSelect(CAN_ID_1, CAN_CONFIGURATION_MODE);
while(PLIB_CAN_OperationModeGet(CAN_ID_1) != CAN_CONFIGURATION_MODE);
```

The CAN module must be placed in Normal mode in order to be able to **send** and receive messages.

The caller should wait until the Normal operating mode is set before proceeding. The `PLIB_CAN_OperationModeSelect` function can be used to obtain the current operating mode of the CAN module.

```
/* Switch the CAN module to Normal mode. Wait until the switch is complete */
PLIB_CAN_OperationModeSelect(CAN_ID_1, CAN_NORMAL_MODE);
while(PLIB_CAN_OperationModeGet(CAN_ID_1) != CAN_NORMAL_MODE);
```

5.6.4.4.2.2 Setting Bus Speed

The bus speed at which the CAN node operates is determined by the CAN module clock and the number of Bit Time Quanta (TQ) assigned to the CAN bit.

The following code snippet shows an example of setting the CAN module clock and CAN bus speed. In this example, the number of Bit TQ is 10.

```
/* Configure the CAN Module Clock. The propagation segment, phase segment 1 and phase segment 2
 * are configured to have 3TQ. The CANSetSpeed function sets the baud. The CPU
 * core frequency is 80000000 Hz and the desired CAN bus speed is 250000 bps.
 */

PLIB_CAN_PhaseSegment2LengthSet(CAN_ID_1, CAN_TIME_SEGMENT_LEN_3TQ);
PLIB_CAN_PhaseSegment1LengthSet(CAN_ID_1, CAN_TIME_SEGMENT_LEN_3TQ);
PLIB_CAN_PropagationTimeSegmentSet(CAN_ID_1, CAN_TIME_SEGMENT_LEN_3TQ);
PLIB_CAN_PhaseSegment2LengthFreelyProgrammableEnable(CAN_ID_1);

PLIB_CAN_BusLine3TimesSamplingEnable(CAN_ID_1);
PLIB_CAN_SyncJumpWidthSet(CAN_ID_1, CAN_TIME_SEGMENT_LEN_2TQ);
```

5.6.4.4.2.3 Assigning Buffer Memory

The CAN module requires RAM memory to store received messages and queue message that need to be transmitted. The total amount of memory required depends on the size of each channel. While each channel has a minimum size of one message buffer (16 bytes), to simplify usage issues, channels should be used in a sequentially ascending order starting with channel 0.

To better understand the buffer memory requirements of the CAN module, consider the following examples:

Example 1: Two transmit channels (each with eight message buffers) and one full receive channel (with 10 message buffers):

- Two transmit channels with eight message buffers: $2 * 8 * 16 = 256$ bytes
- One receive channel with 10 message buffers: $1 * 10 * 16 = 160$ bytes
- Total memory required in bytes = 416 bytes

Example 2: Four transmit channels (two with eight message buffers and other two with 10 message buffers) and two full receive channels (one with 10 message buffers and one with 20 message buffers):

- Two transmit channels with eight message buffers: $2 * 8 * 16 = 256$ bytes
- Two transmit channels with 10 message buffers: $2 * 10 * 16 = 320$ bytes
- One receive channel with 10 message buffers: $1 * 10 * 16 = 160$ bytes
- One receive channel with 20 message buffers: $1 * 20 * 16 = 320$ bytes
- Total memory required in bytes = 1056 bytes

Example 3: One transmit channel (with maximum message buffers) and two receive Channels (one full receive with 10 message buffers and one data-only channel with maximum message buffers):

- One transmit channel with maximum (32) message buffers: $1 * 32 * 16 = 512$ bytes
- One receive channel with 10 message buffers: $1 * 10 * 16 = 160$ bytes
- One data-only receive channel with 32 message buffers: $1 * 32 * 8 = 256$ bytes
- Total memory required in bytes = 928 bytes

The [PLIB_CAN_MemoryBufferAssign](#) function is used to assign a memory buffer to the CAN module, which is shown in the following example.

```
/* Assign the buffer area to the CAN module.
 * In this case assign enough memory for 2
 * channels, each with 8 message buffers.*/

uint8_t MY_CAN_MessageFifoArea[2 * 8 * 16];
PLIB_CAN_MemoryBufferAssign(CAN_ID_1, MY_CAN_MessageFifoArea, (2 * 8 * 16));
```

5.6.4.4.2.4 Channel Configuration

To transmit and receive CAN messages, the user application must configure channels. The [PLIB_CAN_ChannelForTransmitSet](#) function is used to configure a channel for transmit operation. The [PLIB_CAN_ChannelForReceiveSet](#) function is used to configure a channel for receive operation.

Note: To receive CAN messages, at least one message acceptance filter with a receive channel must be configured.

A CAN Transmit channel can be enabled to process a Remote Transmit Request (RTR). In this case, the transmit channel will automatically transmit all queued messages when an RTR message is accepted by the CAN module.

The following code shows an example of configuring a CAN Transmit channel.

```
/* Configure CAN1 Channel 0 for Transmit operation. Allocate 8 message buffer,
 * disable the RTR feature and assign low medium priority for transmissions. */

PLIB_CAN_ChannelForTransmitSet(CAN_ID_1, CAN_CHANNEL0, 8, CAN_TX_RTR_DISABLED,
    CAN_LOW_MEDIUM_PRIORITY);
```

A CAN receive channel can be enabled to store the data payload portion of an accepted message. This mode is called the Data-only mode. The CAN module otherwise stores both the ID and data payload portion of the accepted message.

The following code shows an example of configuring a CAN Receive channel.

```
/* Configure CAN1 Channel 1 for Receive operation. Allocate 8 message buffers,
 * disable the Data-only feature (specify full receive mode). */

PLIB_CAN_ChannelForReceiveSet(CAN_ID_1, CAN_CHANNEL1, 8, CAN_RX_FULL_RECEIVE);
```

5.6.4.4.2.5 Filters and Masks Configuration

The CAN message acceptance filter allows an application to [accept](#) only selected messages from the CAN bus. The CAN module achieves this by comparing the Standard or Extended ID of a message with the configured filter ID. The message is accepted on an ID match. The accepted message is stored in the configured destination channel.

The CAN message acceptance filter mask directs the CAN module to ignore specified bits in the filter comparison. This allows the application to [accept](#) messages within an ID range.

At least one filter and one mask must be configured for the CAN module to be able to receive messages. The destination channel for a filter must be a receive channel unless the channel is a transmit channel with the RTR feature enabled.

The [PLIB_CAN_FilterConfigure](#) function is used to configure a message acceptance filter. The [PLIB_CAN_FilterMaskConfigure](#) function is used to configure a message acceptance filter mask. The filter and mask must be linked together to a channel. This is done using the [PLIB_CAN_FilterToChannelLink](#) function. The CAN filter should be enabled to allow messages to be processed.

This is done using the [PLIB_CAN_FilterEnable](#) function.

To better understand how the filter and masks can be used to implement filtering, consider the following code examples:

Example 1: Configure a CAN filter to [accept](#) a Standard ID message with ID 0x201

```
/* Only one message ID is to be accepted. All bits are to be compared
 * hence the mask value should be 0x7FF. The message type is Standard ID.
 * Configure CAN1, Filter 0, Mask 0 and set the destination channel as
 * Channel 1. */
```

```
PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER0, 0x201, CAN_SID);
PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK0, 0x7FF, CAN_SID,
    CAN_FILTER_MASK_IDE_TYPE);
PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER0, CAN_FILTER_MASK0,
    CAN_CHANNEL1);
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER0);
```

Example 2: Configure a CAN filter to [accept](#) an Extended ID message with ID 0x8004001

```
/* Only one message ID is to be accepted. All bits are to be compared
 * hence the mask value should be 0x8004001. The message type is Extended ID.
 * Configure CAN1, Filter 0, Mask 0 and set the destination channel as
 * Channel 1. */
```

```
PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER0, 0x8004001, CAN_EID);
PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK0, 0x3FFFFFFF, CAN_EID,
    CAN_FILTER_MASK_IDE_TYPE);
PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER0, CAN_FILTER_MASK0,
    CAN_CHANNEL1);
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER0);
```

Example 3: Configure a CAN filter to [accept](#) an Extended ID message within ID range 0x8004000 - 0x8004003

```
/* A message ID range is to be implemented. The 2 least significant bits can be
 * ignored. Therefore, the mask value should be 0x3FFFFFFC. The message type is
 * Extended ID. Configure CAN1, Filter 0, Mask 0 and set the destination channel
 * as Channel 1. */
```

```
PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER0, 0x8004000, CAN_EID);
PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK0, 0x3FFFFFFC,
    CAN_EID, CAN_FILTER_MASK_IDE_TYPE);
PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER0, CAN_FILTER_MASK0,
    CAN_CHANNEL1);
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER0);
```

Note: Refer to About CAN Bus Protocol for more information on Standard and Extended ID addressing.

5.6.4.4.2.6 Enabling Events

The CAN module can generate events that indicate the status of the module. Events can be generated at the channel level and at the module level. Enabling a channel event will cause the CAN module to generate a module level event when the channel event occurs. Enabling a module event will cause the CAN module to interrupt the CPU.

The following code shows an example of using the [PLIB_CAN_ChannelEventEnable](#) and [PLIB_CAN_ModuleEventEnable](#) functions to enable channel and module events.

```
/* Enable the Receive channel not empty and Receive channel
 * full events at the channel level. Enable the Receive event
 * at the module level. */
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL1,
    (CAN_RX_CHANNEL_NOT_EMPTY|CAN_RX_CHANNEL_FULL));
PLIB_CAN_ModuleEventEnable (CAN_ID_1, CAN_RX_EVENT);
```

Note: The caller must use the Interrupt Peripheral Library to enable the CAN CPU interrupt.

5.6.4.4.3 Transmitting a CAN Message

The caller application can transmit a CAN message by queuing the message in a transmit channel, updating the channel, and then flushing it. Flushing a transmit channel will cause the CAN module to transmit all of the CAN messages that are queued in the channel.

The [PLIB_CAN_TransmitBufferGet](#) function should be used to obtain a [CAN_TX_MSG_BUFFER](#) handle type to an empty message buffer. The application should then populate this buffer. The [PLIB_CAN_ChannelUpdate](#) function must be called when the buffer has been populated. Finally, the [PLIB_CAN_TransmitChannelFlush](#) function is called to flush the channel.

The following code shows an example of transmitting an Extended ID message using Channel 0 on the CAN1 module.

```
CAN_TX_MSG_BUFFER * My_Message;

/* Get a pointer to the next buffer in the Transmit channel
 * check if the returned value is null. */
My_Message = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL0);

if(My_Message != NULL)
{
    /* Form a Extended ID CAN message. Start by clearing the buffer.
    Set the IDE bit to 1 to indicate extended ID message. Clear RTR bit
    as this is not a RTR message. */
    My_Message->messageWord[0] = 0;
    My_Message->messageWord[1] = 0;
    My_Message->messageWord[2] = 0;
    My_Message->messageWord[3] = 0;
    My_Message->msgSID.sid = 0x200;
    My_Message->msgEID.eid = 0x4002;
    My_Message->msgEID.ide = 1;
    My_Message->msgEID.sub_remote_req = 0;
    My_Message->msgEID.data_length_code = 1;
    My_Message->data[0] = 0xAA;

    /* This function lets the CAN module
    know that the message processing is done.*/
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL0);

    /* Direct the CAN module to flush the
    Transmit channel. This will send any pending
    message in the Transmit channel. */
    PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL0);
}
```

Note: The [PLIB_CAN_TransmitBufferGet](#) function will return NULL if the transmit channel is full. In this occurs, the [PLIB_CAN_TransmitChannelFlush](#) function should be called to empty the channel.

The following code shows an example of transmitting a Standard ID message using Channel 0 on the CAN1 module.

```
CAN_TX_MSG_BUFFER * myMessage;

/* Get a pointer to the next buffer in the channel
 * check if the returned value is null. */
myMessage = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL0);

if(myMessage != NULL)
{
    /* Form a Standard ID CAN message. Start by clearing the buffer.
    Send message to the CAN2 module. IDE = 0 means Standard ID message.
    Send one byte of data.
    */
    myMessage->messageWord[0] = 0;
    myMessage->messageWord[1] = 0;
```

```

myMessage->messageWord[2] = 0;
myMessage->messageWord[3] = 0;

myMessage->msgSID.sid      = 0x202;
myMessage->msgEID.ide      = 0;
myMessage->msgEID.data_length_code = 1;
myMessage->data[0]         = 0x23;

/* This function lets the CAN module
   know that the message processing is done
   and message is ready to be processed. */
PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL0);

/* Direct the CAN module to flush the
   Transmit channel. This will send any pending
   message in the Transmit channel. */
PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL0);
}

```

5.6.4.4.4 Receiving a CAN Message

The caller application can use either the interrupt or polling technique to detect when a message has been received.

The following code shows how the [PLIB_CAN_ReceivedMessageGet](#) function can be used to read a received message.

```

CAN_RX_MSG_BUFFER * message;

message = (CAN_RX_MSG_BUFFER *)PLIB_CAN_ReceivedMessageGet(CAN_ID_1,
    CAN_CHANNEL1);
if(message != NULL)
{
    //Process the message received

    /* Call the PLIB_CAN_ChannelUpdate function to let
       the CAN module know that the message processing
       is done. */
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL1);
}

```

The [PLIB_CAN_ReceivedMessageGet\(\)](#) function returns NULL if the channel is empty.

5.6.4.4.5 Handling Events

The CAN module can generate events at the channel level and at the module level. An enabled channel event will cause a module event. An enabled module event will cause a CAN CPU interrupt. See [Channel Events](#) and [Module Events](#) more information on each type of event.

5.6.4.4.5.1 Channel Events

Each channel in the module has its own set of events. The type of events are common to all channels. While a specific channel event may be active and can be inspected, it will cause a module event only if the channel event is enabled.

The following code shows an example of enabling and disabling channel events using the [PLIB_CAN_ChannelEventEnable](#) function.

```

/* CAN 1 Channel 1 and 3 are Transmit channels and Channel 2 and 4 are Receive
   channels */
/* Enable Channel 1 empty event and channel not full event. Disable Channel 2 full
   and overflow event */
/* Enable all Transmit events on Channel 2 and disable all RX events on Channel 4 */

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL1,

```

```

    (CAN_TX_CHANNEL_EMPTY | CAN_TX_CHANNEL_NOT_FULL));
    PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL2,
    (CAN_RX_CHANNEL_FULL | CAN_RX_CHANNEL_OVERFLOW));
    PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL3, CAN_TX_CHANNEL_ANY_EVENT);
    PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL4, CAN_RX_CHANNEL_ANY_EVENT);

```

The [PLIB_CAN_ModuleEventGet](#) function can be used to check if a specific event is active in a specific channel.

It is only necessary for the event to be active and not enabled for this to work, as shown in the following code example.

```

// Check if RX Channel 2 of CAN module 1 is not empty or if its full.
CAN_CHANNEL_EVENT channelEvent;

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL2);

if((channelEvent & (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_FULL)) != 0)
{
    // This means that either RX Channel 2 is not empty
    // or the Channel is full.
}

```

If a channel event is enabled and is active, it will set a channel event status flag. This flag indicates that some event in the channel is active.

The following code example shows how the [PLIB_CAN_AllChannelEventsGet](#) function is used to check if any channel event is active.

```

/* Check if RX Channel 2 of CAN module 1 is not empty or if its full */
CAN_CHANNEL_EVENT channelEvent;

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL2);

if((channelEvent & (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_FULL)) != 0)
{
    // This means that either RX Channel 2 is not empty
    // or the Channel is full.
}

```

A receive channel overflow condition is special channel event and is available as a module event as well. The [PLIB_CAN_AllChannelOverflowStatusGet](#) function returns the status of all channels that have an active overflow event. Only channels with enabled overflow events will be reflected.

The following code shows an example of how this function is used.

```

/* Check if Channel 2 or 3 of CAN module 1 have any active events */
CAN_CHANNEL_MASK channelEvents;

channelEvents = PLIB_CAN_AllChannelEventsGet(CAN_ID_1);

if((channelEvents & (CAN_CHANNEL2_MASK | CAN_CHANNEL3_MASK)) != 0)
{
    // Either Channel 2 or 3 has an active event.
    // The PLIB_CAN_ChannelEventGet function can be
    // used to query the channel for more details.
}

```

5.6.4.4.5.2 Module Events

The CAN module has various sources of events, which includes the channel events. The [PLIB_CAN_ModuleEventEnable](#) function can be used to enable and disable module level events. Enabling a module event will cause the CAN module to generate a CPU interrupt, as shown in the following code example.

```

// Enable CAN1 module Transmit Event and Receive Events.
// Disable Receive Overflow event and operation
// mode change event.
PLIB_CAN_ModuleEventEnable(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));

```

```
PLIB_CAN_ModuleEventDisable(CAN_ID_1,
    (CAN_OPERATION_MODE_CHANGE_EVENT | CAN_RX_OVERFLOW_EVENT));
```

The [PLIB_CAN_ModuleEventGet](#) function can be used to check if any module level events are active. This is shown in the code snippet below.

```
// Check if the CAN Module 1 Receive event
// or if Transmit event is active
if((PLIB_CAN_ModuleEventGet(CAN_ID_1) & (CAN_RX_EVENT | CAN_TX_EVENT)) != 0)
{
    // Handle the Receive or Transmit module Event here.
}
```

The CAN module also provides code to indicate the highest priority pending event in the CAN module. The [PLIB_CAN_PendingEventsGet](#) function can be used to read and analyze this code, as shown in the following code example.

```
/* Implement a switch to check and process any active CAN module events */
CAN_EVENT_CODE eventCode;

eventCode = PLIB_CAN_PendingEventsGet(CAN_ID_1);

switch(eventCode)
{
    case CAN_NO_EVENT:
        /* Procedure to handle no CAN events */
        break;
    case CAN_WAKEUP_EVENT:
        /* Procedure to handle device wake-up on CAN bus activity event */
        break;
    default:
        break;
}
```

5.6.4.4.6 Using the Library

This topic describes the basic architecture of the CAN Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_can.h](#)

The interface to the CAN Peripheral library is defined in the "[plib_can.h](#)" header file. Any C language source (.c) file that uses the CAN Peripheral library should include "peripheral.h".

Library File:

The CAN Peripheral library archive (.a) file installed with MPLAB Harmony.

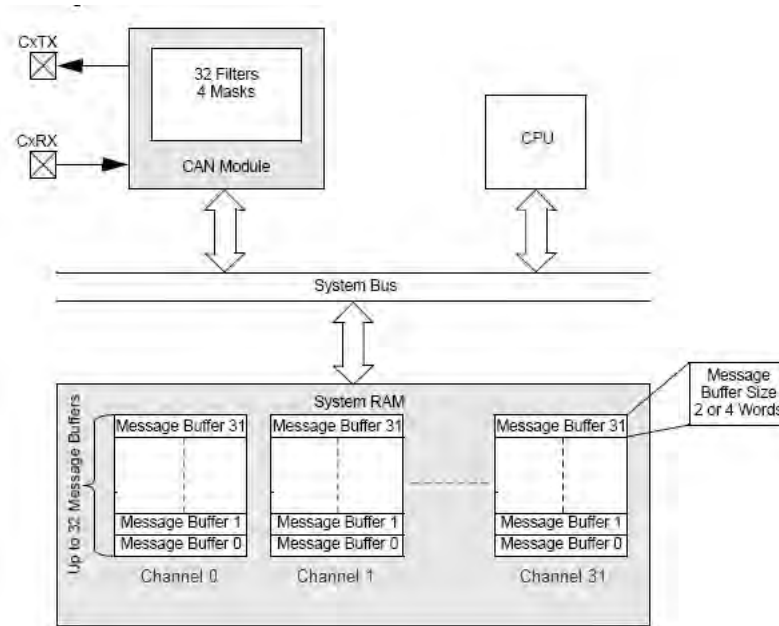
Please refer to the [MPLAB Harmony Overview](#) for how the Peripheral interacts with the framework.

5.6.4.4.6.1 Hardware Abstraction Model

Note: The interface provided is a super set of all of the functionality of the available CAN on the device. Refer to the specific device data sheet or the related family reference manual to determine the set of functions that are supported for each CAN module on the device.

This library provides the low-level abstraction of the CAN module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

CAN Software Abstraction Block Diagram



Software Model:

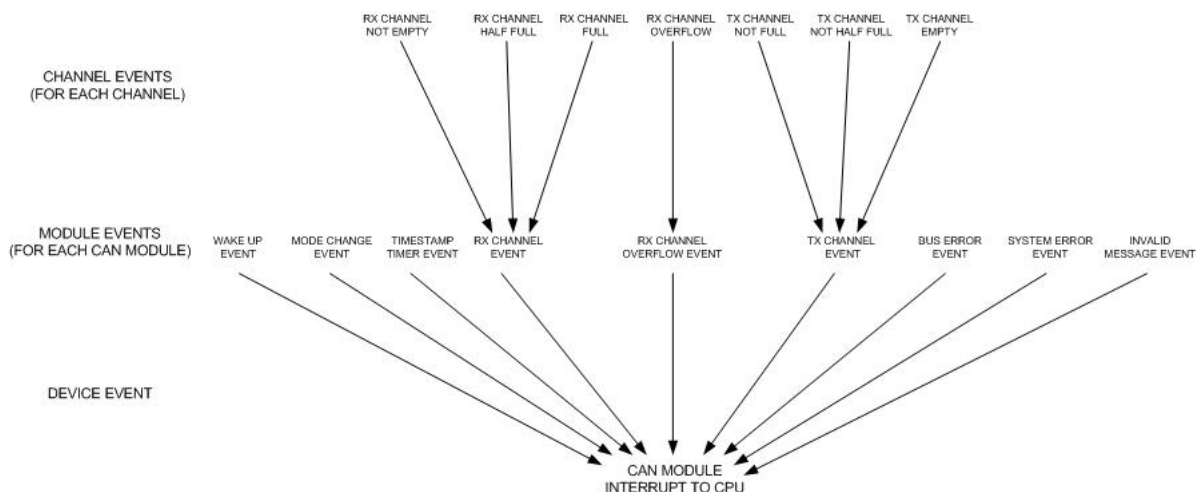
The CAN Peripheral Library provides interface routines to interact with the blocks shown in the diagram above. A channel can be either a transmit channel or a receive channel. The size of a CAN message is typically 16 bytes (8 bytes for data-only receive messages). The size of a channel (the number of messages it can buffer) is configurable. Each channel has a minimum size of one message buffer. The message buffer area for all channels is assigned at configuration time. The CAN Peripheral Library then provides access to these channels and buffers via the library interface.

The user code must enable and configure message acceptance filters to receive messages. These filters compare the ID field of the incoming message with configured value and accepts the messages if IDs match. The message is then stored in a selectable receive channel. At least one message acceptance filter and one filter mask should be enabled for the CAN module to receive messages. A filter mask allows specified filter bits to be ignored during the comparison process. This allows the filter to accept a message with an ID range.

The CAN Peripheral Library allows the CAN module to generate events. Events can be generated at the channel level and at the module level.

Events:

The CAN module is capable of notifying the user application when specific events occur in the module. The following diagram shows these events and how they are organized.



There are two types of events in the CAN module: Channel events and Module events. Channel events are generated by transmit and receive channels. Module events are generated by various sources (including channels) within the CAN module. Each event can be enabled or disabled. Enabling a channel event will cause the CAN module to generate a module event. An enabled module level event will cause the CAN module to generate an interrupt to the CPU.

The application can also poll channel events to check if these are active. This may be done in cases where a polling scheme (rather than an interrupt scheme) needs to be implemented. In this case, the CAN module provides events status on a channel basis. If a channel event is found active, the application can then use library API calls to inspect which channel condition caused the channel event.

5.6.4.4.6.2 Library Overview

The library interface routines are divided into various subsections, each of the sub section addresses one of the blocks or the overall operation of the CAN module.

Library Interface Section	Description
Basic Configuration	APIs that are used for basic control of the module, such as enabling and disabling the module.
Advanced Configuration	Advance configuration, such as selection of the operation mode.
Bus Speed Configuration	Bus speed configuration.
Event Management	APIs to manage the events.
Message Transmit Functions	Transmit APIs.
Message Receive Functions	APIs that are related to data reception.
Error State Tracking	Error tracking.
Information Functions	Information about the module.

5.6.4.4.7 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.4.5 Library Interface

Data Types and Constants

	Name	Description
	<u>CAN_CHANNEL</u>	Identifies the supported CAN Channels.
	<u>CAN_CHANNEL_EVENT</u>	Identifies all Layer 3 interrupts.
	<u>CAN_CHANNEL_MASK</u>	Lists the series of useful masks.
	<u>CAN_DNET_FILTER_SIZE</u>	Specifies the size of the DeviceNet filter.
	<u>CAN_ERROR_STATE</u>	Specifies the CAN module error states.
	<u>CAN_FILTER</u>	CAN event code returned by the CAN module.
	<u>CAN_FILTER_MASK</u>	Identifies the available CAN filter masks.
	<u>CAN_FILTER_MASK_TYPE</u>	Specifies the CAN filter mask type.
	<u>CAN_FUNCTIONAL_MODES</u>	Lists all possible CAN module functional modes.
	<u>CAN_ID_TYPE</u>	Specifies the CAN ID type.
	<u>CAN_MODULE_EVENT</u>	Specifies the CAN module events
	<u>CAN_MODULE_ID</u>	Enumeration: CAN_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.
	<u>CAN_MSG_EID</u>	Defines the structure of the EID word section of the transmit and receive message.
	<u>CAN_OPERATION_MODES</u>	Lists all possible CAN module operational modes.
	<u>CAN_RECEIVE_CHANNEL</u>	Lists all possible CAN module receive channels.
	<u>CAN_RECEIVE_MODES</u>	Lists all possible CAN module receive modes.
	<u>CAN_RX_DATA_MODE</u>	Enables the Data-only Receive mode or Full Receive mode of a CAN receive channel.
	<u>CAN_RX_MSG_BUFFER</u>	Defines the structure of the CAN receive message buffer
	<u>CAN_RX_MSG_SID</u>	Defines the structure of the SID word section of the receive message.
	<u>CAN_TIME_SEGMENT_LENGTH</u>	All possible values for the assignable number of Time Quanta.
	<u>CAN_TX_CHANNEL_STATUS</u>	Identifies possible transmit channel-specific conditions.
	<u>CAN_TX_MSG_BUFFER</u>	Defines the structure of the CAN transmit message buffer.
	<u>CAN_TX_MSG_SID</u>	Defines the structure of the SID word section of the transmit message.
	<u>CAN_TX_RTR</u>	Specifies the status of the CAN Remote Transmit Request (RTR) feature for a CAN transmit channel.
	<u>CAN_TXCHANNEL_PRIORITY</u>	Specifies the priority of a transmit channel.
	<u>CAN_RX_DATA_ONLY_SIZE_BYTES</u>	Used as the size of the CAN data-only receive message buffer in bytes.
	<u>CAN_TX_RX_MESSAGE_SIZE_BYTES</u>	Used as the array size of the CAN transmit and full receive message buffer.

Advanced Configuration

	Name	Description
	<u>PLIB_CAN_ChannelResetsComplete</u>	Returns 'true' if the specified channel reset is complete.
	<u>PLIB_CAN_DeviceNetConfigure</u>	Configures the CAN module DeviceNet(TM) filter feature.
	<u>PLIB_CAN_FunctionalModeGet</u>	Obtains the functional mode of the CAN module.

	PLIB_CAN_MemoryBufferAssign	Assigns buffer memory to the CAN module.
	PLIB_CAN_TimeStampPrescalarSet	Sets the CAN receive message time stamp timer prescalar.
	PLIB_CAN_TimeStampValueGet	Returns the current value of the CAN receive message time stamp timer value.
	PLIB_CAN_TimeStampValueSet	Sets the CAN receive message time stamp timer value.
	PLIB_CAN_FunctionalModeSelect	Selects the functional mode of the CAN module.

Basic Configuration

	Name	Description
	PLIB_CAN_BusActivityWakeupDisable	Disables the wake-up on bus activity receive line filter.
	PLIB_CAN_BusActivityWakeupEnable	Enables the wake-up on bus activity receive line filter.
	PLIB_CAN_Disable	Disables the specified CAN module.
	PLIB_CAN_Enable	Enables the specified CAN module.
	PLIB_CAN_IsActive	Returns 'true' if the CAN module is active.
	PLIB_CAN_OperationModeGet	Obtains the current CAN operating mode.
	PLIB_CAN_OperationModeSelect	Sets the CAN operating mode.
	PLIB_CAN_StopInIdleDisable	Disables the Stop in Idle feature.
	PLIB_CAN_StopInIdleEnable	Enables the CAN module to stop when the processor enters Idle mode.
	PLIB_CAN_TimeStampDisable	Disables the time stamp feature for the CAN module.
	PLIB_CAN_TimeStampEnable	Enables the time stamp feature for the CAN module.

Bus Speed Configuration

	Name	Description
	PLIB_CAN_BusLine3TimesSamplingDisable	Disables the bus line three times sampling.
	PLIB_CAN_BusLine3TimesSamplingEnable	Enables the bus line three times sampling.
	PLIB_CAN_PhaseSegment1LengthSet	Selects the CAN phase buffer segment1 length.
	PLIB_CAN_PhaseSegment2LengthFreelyProgrammableDisable	Disables the CAN phase buffer segment2 length freely programmable mode.
	PLIB_CAN_PhaseSegment2LengthFreelyProgrammableEnable	Makes the CAN phase buffer segment2 length freely programmable.
	PLIB_CAN_PhaseSegment2LengthSet	Selects the CAN phase buffer segment2 length.
	PLIB_CAN_PropagationTimeSegmentSet	Selects the propagation time segment.
	PLIB_CAN_SyncJumpWidthSet	Selects the synchronization jump width.

Channel Configuration

	Name	Description
	PLIB_CAN_ChannelForReceiveSet	Configures a CAN channel for receive operation.
	PLIB_CAN_ChannelForTransmitSet	Configures a CAN channel for transmission.
	PLIB_CAN_ChannelReset	Resets a CAN channel.
	PLIB_CAN_ChannelUpdate	Updates the CAN Channel internal pointers.
	PLIB_CAN_ChannelEventDisable	Enables channel level events.

Error State Tracking

	Name	Description
	PLIB_CAN_ReceiveErrorCountGet	Returns the CAN receive error count.
	PLIB_CAN_ErrorStateGet	Returns the CAN error status word.

Event Management

	Name	Description
≡	PLIB_CAN_ModuleEventClear	Clears the CAN module level events.
≡	PLIB_CAN_ModuleEventDisable	Disables the module level events.
≡	PLIB_CAN_ModuleEventEnable	Enables the module level events.
≡	PLIB_CAN_ModuleEventGet	Returns the status of the CAN module events.
≡	PLIB_CAN_AllChannelEventsGet	Returns a value representing the event status of all CAN channels.
≡	PLIB_CAN_AllChannelOverflowStatusGet	Returns a value representing the receive overflow event status of all CAN channels.
≡	PLIB_CAN_ChannelEventClear	Clears CAN channel events.
≡	PLIB_CAN_ChannelEventEnable	Enables channel level events.
≡	PLIB_CAN_ChannelEventGet	Returns a value representing the event status of a CAN channel.
≡	PLIB_CAN_PendingEventsGet	Returns a value representing the highest priority active event in the module.

Feature Existence Routines

	Name	Description
≡	PLIB_CAN_ExistsActiveStatus	Identifies whether the ActiveStatus feature exists on the CAN module.
≡	PLIB_CAN_ExistsAllChannelEvents	Identifies whether the AllChannelEvents feature exists on the CAN module.
≡	PLIB_CAN_ExistsAllChannelOverflow	Identifies whether the AllChannelOverflow feature exists on the CAN module.
≡	PLIB_CAN_ExistsBusActivityWakeup	Identifies whether the BusActivityWakeup feature exists on the CAN module.
≡	PLIB_CAN_ExistsBusLine3TimesSampling	Identifies whether the BusLine3TimesSampling feature exists on the CAN module.
≡	PLIB_CAN_ExistsChannelEvent	Identifies whether the ChannelEventGet feature exists on the CAN module.
≡	PLIB_CAN_ExistsChannelEventEnable	Identifies whether the ChannelEventEnable feature exists on the CAN module.
≡	PLIB_CAN_ExistsChannelForReceiveSet	Identifies whether the ChannelForReceiveSet feature exists on the CAN module.
≡	PLIB_CAN_ExistsChannelForTransmitSet	Identifies whether the ChannelForTransmitSet feature exists on the CAN module.
≡	PLIB_CAN_ExistsChannelReset	Identifies whether the ChannelReset feature exists on the CAN module.
≡	PLIB_CAN_ExistsChannelUpdate	Identifies whether the ChannelUpdate feature exists on the CAN module.
≡	PLIB_CAN_ExistsDeviceNet	Identifies whether the DeviceNet feature exists on the CAN module.
≡	PLIB_CAN_ExistsEnableControl	Identifies whether the EnableControl feature exists on the CAN module.
≡	PLIB_CAN_ExistsErrorState	Identifies whether the ErrorStateGet feature exists on the CAN module.
≡	PLIB_CAN_ExistsFilterConfigure	Identifies whether the FilterConfigure feature exists on the CAN module.
≡	PLIB_CAN_ExistsFilterEnable	Identifies whether the FilterEnable feature exists on the CAN module.

	<u>PLIB_CAN_ExistsFilterMaskConfigure</u>	Identifies whether the FilterMaskConfigure feature exists on the CAN module.
	<u>PLIB_CAN_ExistsFilterToChannelLink</u>	Identifies whether the FilterToChannelLink feature exists on the CAN module.
	<u>PLIB_CAN_ExistsFunctionalMode</u>	Identifies whether the FunctionalModeSelect feature exists on the CAN module.
	<u>PLIB_CAN_ExistsLatestFilterMatchGet</u>	Identifies whether the LatestFilterMatchGet feature exists on the CAN module.
	<u>PLIB_CAN_ExistsMemoryBufferAssign</u>	Identifies whether the MemoryBufferAssign feature exists on the CAN module.
	<u>PLIB_CAN_ExistsMessageIsReceived</u>	Identifies whether the MessageIsReceived feature exists on the CAN module.
	<u>PLIB_CAN_ExistsModuleEventClear</u>	Identifies whether the ModuleEvents feature exists on the CAN module.
	<u>PLIB_CAN_ExistsModuleEventEnable</u>	Identifies whether the ModuleEventEnable feature exists on the CAN module.
	<u>PLIB_CAN_ExistsModuleInfo</u>	Identifies whether the ModuleInformation feature exists on the CAN module.
	<u>PLIB_CAN_ExistsOperationModeRead</u>	Identifies whether the OperationModeRead feature exists on the CAN module.
	<u>PLIB_CAN_ExistsOperationModeWrite</u>	Identifies whether the OperationModeSet feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPendingEventsGet</u>	Identifies whether the PendingEventsGet feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPendingTransmissionsAbort</u>	Identifies whether the PendingTransmissionsAbort feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPhaseSegment1Length</u>	Identifies whether the PhaseSegment1Length feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPhaseSegment2Length</u>	Identifies whether the PhaseSegment2Length feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPhaseSegment2LengthFreelyProgrammable</u>	Identifies whether the PhaseSegment2LengthFreelyProgrammable feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPropagationTimeSegment</u>	Identifies whether the PropagationTimeSegment feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceiveBufferClear</u>	Identifies whether the ReceiveBufferClear feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceiveBufferIsEmpty</u>	Identifies whether the ReceiveBufferIsEmpty feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceivedMessageGet</u>	Identifies whether the ReceivedMessageGet feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceiveErrorCount</u>	Identifies whether the ReceiveErrorCount feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceiveModeSelect</u>	Identifies whether the ReceiveModeSelect feature exists on the CAN module.
	<u>PLIB_CAN_ExistsRemoteTransmitReq</u>	Identifies whether the RemoteTransmitReq feature exists on the CAN module.
	<u>PLIB_CAN_ExistsRxFIFOFourLeftNotify</u>	Identifies whether the RxFIFOFourLeftNotify feature exists on the CAN module.
	<u>PLIB_CAN_ExistsRxFIFOOneLeftNotify</u>	Identifies whether the RxFIFOOneLeftNotify feature exists on the CAN module.

	PLIB_CAN_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CAN module.
	PLIB_CAN_ExistsSyncJumpWidthSet	Identifies whether the SyncJumpWidth feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampEnable	Identifies whether the TimeStampEnable feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampPrescaler	Identifies whether the TimeStampPrescaler feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampValue	Identifies whether the TimeStampValue feature exists on the CAN module.
	PLIB_CAN_ExistsTransmissionIsAborted	Identifies whether the TransmissionAbortStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitBufferGet	Identifies whether the TransmitBufferGet feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelFlush	Identifies whether the TransmitChannelFlush feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelStatus	Identifies whether the TransmitChannelStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitErrorCountGet	Identifies whether the TransmitErrorCountGet feature exists on the CAN module.

Information Functions

	Name	Description
	PLIB_CAN_TotalChannelsGet	Returns the total number of CAN channels per CAN module.
	PLIB_CAN_TotalFiltersGet	Returns the total number of CAN Filters per CAN module.
	PLIB_CAN_TotalMasksGet	Returns the total number of CAN masks per CAN module.

Message Filtering Functions

	Name	Description
	PLIB_CAN_FilterConfigure	Configures a CAN message acceptance filter.
	PLIB_CAN_FilterDisable	Disables a CAN message acceptance filter.
	PLIB_CAN_FilterEnable	Enables a CAN message acceptance filter.
	PLIB_CAN_FilterIsDisabled	Returns 'true' if the specified filter is disabled.
	PLIB_CAN_FilterMaskConfigure	Configures a CAN filter mask.
	PLIB_CAN_FilterToChannelLink	Links a filter to a channel.
	PLIB_CAN_LatestFilterMatchGet	Returns the index of the filter that accepted the latest message.

Message Receive Functions

	Name	Description
	PLIB_CAN_ReceivedMessageGet	Returns a pointer to a message to be read from the CAN channel.
	PLIB_CAN_RxFIFOFourLeftNotify	Configures the module to generate an interrupt when the FIFO has four receive buffers left.
	PLIB_CAN_RxFIFOOneLeftNotify	Configures the module to generate an interrupt when the FIFO has only one receive buffer left.
	PLIB_CAN_ReceiveModeSelect	Selects the receive mode of the CAN module.
	PLIB_CAN_ReceiveModeGet	Obtains the receive mode of the CAN module.
	PLIB_CAN_MessageIsReceived	Returns whether a message was received and the receive buffer is not empty.
	PLIB_CAN_ReceiveBufferIsEmpty	Returns whether the receive buffer of the specified channel is empty.
	PLIB_CAN_ReceiveBufferClear	Clears the received message flag.

	PLIB_CAN_RemoteTransmitReqlsReceived	Returns whether the message received is a remote transmit request.

Message Transmit Functions

	Name	Description
	PLIB_CAN_PendingTransmissionsAbort	Aborts any pending transmit operations.
	PLIB_CAN_TransmissionIsAborted	Returns 'true' if the transmit abort operation is complete.
	PLIB_CAN_TransmitBufferGet	Returns a pointer to an empty transmit buffer.
	PLIB_CAN_TransmitChannelFlush	Causes all messages in the specified transmit channel to be transmitted.
	PLIB_CAN_TransmitChannelStatusGet	Returns the condition of the transmit channel.
	PLIB_CAN_TransmitErrorCountGet	Returns the CAN transmit error count

Description

This section describes the APIs of the CAN Peripheral. Refer to each section for a detailed description.

5.6.4.5.1 Basic Configuration

5.6.4.5.1.1 PLIB_CAN_BusActivityWakeupDisable Function

C

```
void PLIB_CAN_BusActivityWakeupDisable(
    CAN_MODULE_ID index
);
```

Description

This function disables the Wake up on bus activity receive line filter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsBusActivityWakeup](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_BusActivityWakeupDisable(MY_CAN_ID);
```

5.6.4.5.1.2 PLIB_CAN_BusActivityWakeupEnable Function

C

```
void PLIB_CAN_BusActivityWakeupEnable(
```

```
    CAN_MODULE_ID index
);
```

Description

This function enables the wake-up on bus activity receive line filter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsBusActivityWakeup](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_BusActivityWakeupEnable(MY_CAN_ID);
```

5.6.4.5.1.3 PLIB_CAN_Disable Function**C**

```
void PLIB_CAN_Disable(
    CAN_MODULE_ID index
);
```

Description

This function disables the specified CAN module.

Preconditions

None

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

If the user application clears this bit, it may take a number of cycles before the CAN module completes the current transaction and responds to this request. The user application should check this using the [PLIB_CAN_IsActive](#) function to verify that the request has been honored.

This function works only if the CAN module is in Configuration mode. Use the [PLIB_CAN_OperationModeGet](#) function to get the current mode and the [PLIB_CAN_OperationModeSelect](#) function to change the mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_Disable(MY_CAN_ID);
```

5.6.4.5.1.4 PLIB_CAN_Enable Function

C

```
void PLIB_CAN_Enable(
    CAN_MODULE_ID index
);
```

Description

This function enables the specified CAN module.

Preconditions

The module should be appropriately configured before being enabled.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

Some of the CAN module settings can be done only when the module is off. Therefore, always call the PLIB_CAN_Enable Function after all other configurations are complete.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

//Do all the other configurations before enabling.

PLIB_CAN_Enable(MY_CAN_ID);
```

5.6.4.5.1.5 PLIB_CAN_IsActive Function

C

```
bool PLIB_CAN_IsActive(
    CAN_MODULE_ID index
);
```

Description

This function returns 'true' if the CAN module is active. This function is typically called after the CAN module is disabled using the [PLIB_CAN_Disable](#) function. The CAN module disable request does not complete immediately when requested and depends on

the CAN bus status. This function should be called to check whether the module disable is completed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

- true = The module is still active
- false = The module is completely disabled

Remarks

The CAN module disable operation should not be confused with placing the CAN module in the Disable mode using the [PLIB_CAN_OperationModeSelect](#) function. The Disable mode is one of the operating modes of the CAN module. The CAN module is still enabled while in Disable mode. The module disable operation switches the CAN module off.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsActiveStatus](#) in your application to determine whether this feature is available.

Example

```
// Disable CAN module 1. Wait until the module is completely disabled.  
PLIB_CAN_Disable(CAN_ID_1);  
  
while(PLIB_CAN_IsActive(CAN_ID_1) == true);
```

5.6.4.5.1.6 PLIB_CAN_OperationModeGet Function

C

```
CAN_OPERATION_MODES PLIB_CAN_OperationModeGet(  
    CAN_MODULE_ID index  
);
```

Description

This function obtains the current CAN operating mode. This function is typically called after the [PLIB_CAN_OperationModeSelect](#) function to check if the requested operation mode change is complete.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

The current CAN_OP_MODE type of operation mode of the CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsOperationModeRead](#) in your application to determine whether this feature is available.

Example

```
// Check and wait until the CAN module is in Disable Mode.
```

```
while(PLIB_CAN_OperationModeGet(CAN_ID_1) != CAN_DISABLE_MODE);
```

5.6.4.5.1.7 PLIB_CAN_OperationModeSelect Function

C

```
void PLIB_CAN_OperationModeSelect(  
    CAN_MODULE_ID index,  
    CAN_OPERATION_MODES opMode  
);
```

Description

This function sets the CAN operating mode. The CAN module requires itself to be in certain modes to gain access to module functionality. For example, the module should be in Normal mode to [send](#) and receive messages. Note that after this function is called, it should be checked to determine whether the mode was set by using the [PLIB_CAN_OperationModeGet](#) function.

Preconditions

None.

Parameters

Parameters	Description
module	Identifies the desired CAN module
opMode	Desired CAN_OP_MODE type of the mode to be set

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsOperationModeWrite](#) in your application to determine whether this feature is available.

Example

```
// Set the CAN_ID_1 operating mode to Configuration mode.  
  
PLIB_CAN_OperationModeSelect(CAN_ID_1, CAN_CONFIGURATION_MODE);
```

5.6.4.5.1.8 PLIB_CAN_StopInIdleDisable Function

C

```
void PLIB_CAN_StopInIdleDisable(  
    CAN_MODULE_ID index  
);
```

Description

This function disables the CAN module from stopping operation when the system enters Idle mode. The module continues operation when the system enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_StopInIdleDisable(MY_CAN_ID);
```

5.6.4.5.1.9 PLIB_CAN_StopInIdleEnable Function**C**

```
void PLIB_CAN_StopInIdleEnable(
    CAN_MODULE_ID index
);
```

Description

This function configures the CAN module to stop operation when the system enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_StopInIdleEnable(MY_CAN_ID);
```

5.6.4.5.1.10 PLIB_CAN_TimeStampDisable Function**C**

```
void PLIB_CAN_TimeStampDisable(
    CAN_MODULE_ID index
);
```

Description

This function disables time stamp capture feature for the CAN module. This conserves power by stopping the CAN timer.

Preconditions

None

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTimeStampEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_TimeStampDisable(MY_CAN_ID);
```

5.6.4.5.1.11 PLIB_CAN_TimeStampEnable Function**C**

```
void PLIB_CAN_TimeStampEnable(
    CAN_MODULE_ID index
);
```

Description

This function enables the time stamp capture feature for the CAN module. The CAN timer value will be stored on valid message reception and is stored with the message.

Preconditions

None

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTimeStampEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_TimeStampEnable(MY_CAN_ID);
```

5.6.4.5.2 Advanced Configuration

5.6.4.5.2.1 PLIB_CAN_ChannelResetIsComplete Function

C

```
bool PLIB_CAN_ChannelResetIsComplete(  
    CAN_MODULE_ID index,  
    CAN_CHANNEL channel  
);
```

Description

This function returns 'true' if the specified channel reset is complete. This function should preferably be called after calling the [PLIB_CAN_ChannelReset](#) function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the CAN Channel to be inspected for reset

Returns

- true = Channel reset is complete
- false = Channel reset is in progress

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsChannelReset](#) in your application to determine whether this feature is available.

Example

```
// Reset channel 4 of CAN_ID_1 module  
// and wait until the reset is complete  
PLIB_CAN_ChannelReset(CAN_ID_1, CAN_CHANNEL4);  
  
while(PLIB_CAN_ChannelResetIsComplete(CAN_ID_1, CAN_CHANNEL4) != true);
```

5.6.4.5.2.2 PLIB_CAN_DeviceNetConfigure Function

C

```
void PLIB_CAN_DeviceNetConfigure(  
    CAN_MODULE_ID index,  
    CAN_DNET_FILTER_SIZE dncnt  
);
```

Description

This function configures the CAN module DeviceNet filter feature. DeviceNet filtering allows a portion of the data payload to be used as a filter criteria. This function allows the size of this filter to be configured in bits. The filter can also be disabled.

Preconditions

The CAN module should preferably be in Configuration mode (using the [PLIB_CAN_OperationModeSelect](#) function).

Parameters

Parameters	Description
index	Identifies the desired CAN module
dncnt	Specifies the CAN_DNET_FILTER_SIZE size of the DeviceNet filter in bits

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsDeviceNet](#) in your application to determine whether this feature is available.

Example

```
// Disable the CAN_ID_1 DeviceNet filter feature.  
PLIB_CAN_DeviceNetConfigure(CAN_ID_1, CAN_DNET_FILTER_DISABLE);  
  
// Set the CAN_ID_2 Device Net Filter Size to 10 bits.  
PLIB_CAN_DeviceNetConfigure(CAN_ID_2, CAN_DNET_FILTER_SIZE_10_BIT);
```

5.6.4.5.2.3 PLIB_CAN_FunctionalModeGet Function**C**

```
CAN_FUNCTIONAL_MODES PLIB_CAN_FunctionalModeGet(  
    CAN_MODULE_ID index  
);
```

Description

This function obtains the functional mode of the CAN module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsFunctionalMode](#) in your application to determine whether this feature is available.

Example

```
CAN_FUNCTIONAL_MODES funcMode;  
  
funcMode = PLIB_CAN_FunctionalModeGet(CAN_ID_1);
```

5.6.4.5.2.4 PLIB_CAN_MemoryBufferAssign Function**C**

```
void PLIB_CAN_MemoryBufferAssign(  
    CAN_MODULE_ID index,  
    void * buffer,  
    int sizeInBytes  
);
```

Description

This function assigns buffer memory to the CAN module. The CAN module uses this buffer memory to store messages to be

transmitted and received. The size of the memory buffer should be enough to accommodate the required number of message buffers and channels.

Preconditions

The module should be in Configuration mode (using the [PLIB_CAN_OperationModeSelect](#) function).

Parameters

Parameters	Description
index	Identifies the desired CAN module
buffer	Pointer to the buffer memory area
sizeInBytes	Size of the buffer memory in bytes

Returns

None.

Remarks

This API is useful only if the CAN module uses device RAM for message buffers and message FIFO location is software configurable. This should not be used if the device message buffer is part of SFR.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsMemoryBufferAssign](#) in your application to determine whether this feature is available.

Example

```
// Define and assign a CAN 1 memory buffer for 2 Transmit channels and 1 Receive  
// channel, each with 4 message buffers
```

```
uint8_t canMemoryBuffer [3 * 4 * CAN_TX_RX_MESSAGE_SIZE_BYTES];
```

```
PLIB_CAN_MemoryBufferAssign(CAN_ID_1, canMemoryBuffer, (3 * 4 *  
    CAN_TX_RX_MESSAGE_SIZE_BYTES));
```

5.6.4.5.2.5 PLIB_CAN_TimeStampPrescalarSet Function

C

```
void PLIB_CAN_TimeStampPrescalarSet(  
    CAN_MODULE_ID index,  
    unsigned prescalar  
);
```

Description

This function sets the CAN receive message time stamp timer prescalar. This prescalar determines the rate at which the time stamp timer is incremented. For example, if the prescalar value is 0xFF, the time stamp timer is incremented by 1 every 255 system clock periods.

Preconditions

None.

Parameters

Parameters	Description
module	Identifies the desired CAN module
prescalar	Prescalar for the CAN receive message time stamp timer. This value should be between 0x0 and 0xFFFF.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTimeStampPrescalar](#) in your application to determine whether this feature is available.

Example

```
// Set the CAN_ID_1 Receive Message Time Stamp Timer prescalar to increment
//the Timestamp Timer every 1024 system clock periods.

PLIB_CAN_TimeStampPrescalarSet(CAN_ID_1, 1024);
```

5.6.4.5.2.6 PLIB_CAN_TimeStampValueGet Function**C**

```
unsigned PLIB_CAN_TimeStampValueGet(
    CAN_MODULE_ID index
);
```

Description

This function gets the current value of the CAN receive message time stamp timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

The current value of the CAN receive message time stamp timer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTimeStampValue](#) in your application to determine whether this feature is available.

Example

```
unsigned int timeStampTimerVal;

// Get the current time stamp timer value.
timeStampTimerVal = PLIB_CAN_TimeStampValueGet(CAN_ID_1);
```

5.6.4.5.2.7 PLIB_CAN_TimeStampValueSet Function**C**

```
void PLIB_CAN_TimeStampValueSet(
    CAN_MODULE_ID index,
    unsigned value
);
```

Description

This function sets the CAN receive message time stamp timer value. The timer will then start counting up from this value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
value	Start value of the receive message yime stamp timer. This value should be between 0x0 and 0xFFFF.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTimeStampValue](#) in your application to determine whether this feature is available.

Example

```
// Set start value of CAN_ID_1 Receive Message Time Stamp Timer to 0x100  
PLIB_CAN_TimeStampValueSet(CAN_ID_1, 0x0100);
```

5.6.4.5.2.8 PLIB_CAN_FunctionalModeSelect Function**C**

```
void PLIB_CAN_FunctionalModeSelect(  
    CAN_MODULE_ID index,  
    CAN_FUNCTIONAL_MODES funcMode  
);
```

Description

This function selects the functional mode of the CAN module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
funcMode	Select the functional mode of the CAN module. One of the possible values from CAN_FUNCTIONAL_MODES

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsFunctionalMode](#) in your application to determine whether this feature is available.

Example

```
PLIB_CAN_FunctionalModeSelect(CAN_ID_1, CAN_LEGACY_MODE0);
```

5.6.4.5.3 Bus Speed Configuration**5.6.4.5.3.1 PLIB_CAN_BusLine3TimesSamplingDisable Function****C**

```
void PLIB_CAN_BusLine3TimesSamplingDisable(  

```

```
    CAN_MODULE_ID index
);
```

Description

The bus line will be sampled only once at the sampling point.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This function works only if the CAN module is in Configuration mode. Use the [PLIB_CAN_OperationModeGet](#) function to get the current mode and the [PLIB_CAN_OperationModeSelect](#) function to change the mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsBusLine3TimesSampling](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE != PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
    PLIB_CAN_BusLine3TimesSamplingDisable(MY_CAN_ID);
}
```

5.6.4.5.3.2 PLIB_CAN_BusLine3TimesSamplingEnable Function**C**

```
void PLIB_CAN_BusLine3TimesSamplingEnable(
    CAN_MODULE_ID index
);
```

Description

The bus line will be sampled three times at the sampling point. If this is disabled, the bus line will be sampled only once.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This function works only if the CAN module is in Configuration mode. Use the [PLIB_CAN_OperationModeGet](#) function to get the current mode and the [PLIB_CAN_OperationModeSelect](#) function to change the mode.

This API may not function if the baud rate prescaler value is more than a certain limit. Refer to the specific device data sheet to determine the maximum prescaler that allows three times sampling.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsBusLine3TimesSampling](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE != PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
    PLIB_CAN_BusLine3TimesSamplingEnable(MY_CAN_ID);
}
```

5.6.4.5.3.3 PLIB_CAN_PhaseSegment1LengthSet Function

C

```
void PLIB_CAN_PhaseSegment1LengthSet(
    CAN_MODULE_ID index,
    CAN_TIME_SEGMENT_LENGTH length
);
```

Description

This function selects the CAN phase buffer segment1 length.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
length	Desired segment length

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This function works only if the CAN module is in configuration mode. Use the [PLIB_CAN_OperationModeGet](#) function to get the current mode and the [PLIB_CAN_OperationModeSelect](#) function to change the mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsPhaseSegment1Length](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE != PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
    PLIB_CAN_PhaseSegment1LengthSet(MY_CAN_ID, CAN_TIME_SEGMENT_LEN_8TQ);
}
```

5.6.4.5.3.4 PLIB_CAN_PhaseSegment2LengthFreelyProgrammableDisable Function

C

```
void PLIB_CAN_PhaseSegment2LengthFreelyProgrammableDisable(  
    CAN_MODULE_ID index  
);
```

Description

This function disables the CAN phase buffer segment2 length freely programmable mode. The segment2 length will be the maximum of the segment1 length or information processing time, whichever is greater.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This function works only if the CAN module is in Configuration mode. Use the [PLIB_CAN_OperationModeGet](#) function to get the current mode and the [PLIB_CAN_OperationModeSelect](#) function to change the mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsPhaseSegment2LengthFreelyProgrammable](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1  
  
if (CAN_CONFIGURATION_MODE != PLIB_CAN_OperationModeGet(MY_CAN_ID))  
{  
    PLIB_CAN_PhaseSegment2LengthFreelyProgrammableDisable(MY_CAN_ID);  
}
```

5.6.4.5.3.5 PLIB_CAN_PhaseSegment2LengthFreelyProgrammableEnable Function

C

```
void PLIB_CAN_PhaseSegment2LengthFreelyProgrammableEnable(  
    CAN_MODULE_ID index  
);
```

Description

This function makes the CAN phase buffer segment2 length freely programmable. If this mode is not set, the segment2 length will be the maximum of the segment1 length or information processing time, whichever is greater.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This function works only if the CAN module is in Configuration mode. Use the [PLIB_CAN_OperationModeGet](#) function to get the current mode and the [PLIB_CAN_OperationModeSelect](#) function to change the mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsPhaseSegment2LengthFreelyProgrammable](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE != PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
    PLIB_CAN_PhaseSegment2LengthFreelyProgrammableEnable(MY_CAN_ID);
}
```

5.6.4.5.3.6 PLIB_CAN_PhaseSegment2LengthSet Function**C**

```
void PLIB_CAN_PhaseSegment2LengthSet(
    CAN_MODULE_ID index,
    CAN_TIME_SEGMENT_LENGTH length
);
```

Description

This function selects the CAN phase buffer segment2 length.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
length	Desired segment length, select from CAN_TIME_SEGMENT_LENGTH enum

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This function works only if the CAN module is in Configuration mode. Use the [PLIB_CAN_OperationModeGet](#) function to get the current mode and the [PLIB_CAN_OperationModeSelect](#) function to change the mode.

This function will not work if the freely programmable mode is disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsPhaseSegment2Length](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE != PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
    PLIB_CAN_PhaseSegment2LengthFreelyProgrammableEnable(MY_CAN_ID);

    PLIB_CAN_PhaseSegment2LengthSet(MY_CAN_ID,
        CAN_TIME_SEGMENT_LEN_8TQ);
}
```

5.6.4.5.3.7 PLIB_CAN_PropagationTimeSegmentSet Function**C**

```
void PLIB_CAN_PropagationTimeSegmentSet(
    CAN_MODULE_ID index,
    CAN_TIME_SEGMENT_LENGTH length
);
```

Description

This function selects the propagation time segment.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module.

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This function works only if the CAN module is in configuration mode. Use [PLIB_CAN_OperationModeGet](#) function to get the current mode and [PLIB_CAN_OperationModeSelect](#) function to change the mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsPropagationTimeSegment](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE != PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
    PLIB_CAN_PropagationTimeSegmentSet(MY_CAN_ID, CAN_TIME_SEGMENT_LEN_8TQ);
}
```

5.6.4.5.3.8 PLIB_CAN_SyncJumpWidthSet Function**C**

```
void PLIB_CAN_SyncJumpWidthSet(
    CAN_MODULE_ID index,
    CAN_TIME_SEGMENT_LENGTH length
);
```

```
);
```

Description

This function selects the synchronization jump width.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

None.

Remarks

The MY_CAN_ID macro in the example above is used as a placeholder for the actual CAN bus ID (as defined by the processor-specific [CAN_MODULE_ID](#) enum).

This function works only if the CAN module is in Configuration mode. Use the [PLIB_CAN_OperationModeGet](#) function to get the current mode and the [PLIB_CAN_OperationModeSelect](#) function to change the mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsSyncJumpWidthSet](#) in your application to determine whether this feature is available.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE != PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
    PLIB_CAN_SyncJumpWidthSet(MY_CAN_ID,
        CAN_TIME_SEGMENT_LEN_8TQ);
}
```

5.6.4.5.4 Channel Configuration

5.6.4.5.4.1 PLIB_CAN_ChannelForReceiveSet Function

C

```
void PLIB_CAN_ChannelForReceiveSet(
    CAN_MODULE_ID index,
    CAN_CHANNEL channel,
    uint32_t channelSize,
    CAN_RX_DATA_MODE dataOnly
);
```

Description

This function configures a CAN channel for receive operation. A receive channel can be either a full CAN message receive channel, which receives an entire CAN message (Arbitration field + Data field) or a data-only message channel, which receives only the data payload section of the message. A receive channel can buffer upto 32 messages. The number of messages to buffer (i.e., the size of the channel) is set by the channelSize parameter.

Preconditions

The CAN module should be in Configuration mode. This is achieved by using the [PLIB_CAN_OperationModeSelect](#) function.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Receive channel
channelSize	Specifies the number of received messages that the channel can buffer before it overflows. This should be a value between 1 and 32.

5.6.4.5.4.2 PLIB_CAN_ChannelForTransmitSet Function**C**

```
void PLIB_CAN_ChannelForTransmitSet(  
    CAN_MODULE_ID index,  
    CAN_CHANNEL channel,  
    uint8_t channelSize,  
    CAN_TX_RTR rtren,  
    CAN_TXCHANNEL_PRIORITY priority  
);
```

Description

This function configures a CAN Channel for transmission. The size of the channel specifies the number of messages that the channel can buffer. The channel is a First In First Out (FIFO) type of buffer. The remote transmit request feature allows the transmit channel to start transmitting when an associated filter has received a message. The transmit channel priority determines the priority as compared to other transmit channels. If two transmit channels have the same priority, the natural channel priority determines which channel transmits first.

Preconditions

The CAN Module should be in Configuration mode. This is achieved by using the [PLIB_CAN_OperationModeSelect](#) function.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel
channelSize	Size of the channel in messages. This value should be between 1 and 32.

5.6.4.5.4.3 PLIB_CAN_ChannelReset Function**C**

```
void PLIB_CAN_ChannelReset(  
    CAN_MODULE_ID index,  
    CAN_CHANNEL channel  
);
```

Description

This function resets a CAN channel. Resetting a CAN channel causes it to discard any unread received messages or any messages that have not been transmitted yet. The reset operation will wait if a message is being currently transmitted or is being received. The [PLIB_CAN_ChannelResetIsComplete](#) function can be used to check if the channel reset is complete.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

channel	Identifies the CAN channel to be reset

Returns

None.

Remarks

Attempting to read from a channel that is reset will return messages that may have already been read or may not have been read at all.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsChannelReset](#) in your application to determine whether this feature is available.

Example

```
// Reset channel 4 of CAN_ID_1 module  
// and wait until the reset is complete  
  
PLIB_CAN_ChannelReset(CAN_ID_1, CAN_CHANNEL4);  
  
while(PLIB_CAN_ChannelResetIsComplete(CAN_ID_1, CAN_CHANNEL4) != true);
```

5.6.4.5.4.4 PLIB_CAN_ChannelUpdate Function**C**

```
void PLIB_CAN_ChannelUpdate(  
    CAN_MODULE_ID index,  
    CAN_CHANNEL channel  
);
```

Description

This function updates the CAN Channel internal pointers. This function should be called when a message has been read or processed completely from a CAN receive channel (using the [PLIB_CAN_ReceivedMessageGet](#) function). It should be called after a new message has been written to a CAN transmit channel (using the [PLIB_CAN_TransmitBufferGet](#) function) and before the [PLIB_CAN_TransmitChannelFlush](#) function.

Trying to read a CAN receive channel that has not been updated will cause the [PLIB_CAN_ReceivedMessageGet](#) function to return the same message. Writing to a CAN transmit channel that has not been updated will cause the last written message to be overwritten.

Preconditions

This function is effective only when the CAN module is not in Configuration mode or Disable mode.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the CAN channel to be updated

Returns

None.

Remarks

The PLIB_CAN_ChannelUpdate function should be called on a CAN receive channel only after the message obtained using the [PLIB_CAN_ReceivedMessageGet](#) function has been read or processed completely. The CAN module peripheral library does not provide access to older messages once the PLIB_CAN_ChannelUpdate function has been called.

When using the PLIB_CAN_TransmitMessageBuffer function to write to a CAN transmit channel the PLIB_CAN_ChannelUpdate function should be called only when a valid message has been written to the channel. Calling the update function without writing

to the CAN channel will cause an incorrect message to be output on the CAN channel when the transmit channel is flushed.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsChannelUpdate](#) in your application to determine whether this feature is available.

Example

```
// CAN_ID_1 Channel 1 is a Receive channel and Channel 2 is a Transmit  
// channel. Read and update Channel 1. Write a message to Channel 2 and then  
// update and flush the channel.  
  
CAN_TX_MSG_BUFFER * TransmitMessage;  
CAN_RX_MSG_BUFFER * rxMessage;  
  
rxMessage = PLIB_CAN_ReceivedMessageGet(CAN_ID_1, CAN_CHANNEL1);  
  
if(rxMessage != NULL)  
{  
    // Process the received message.  
  
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL1);  
}  
  
TransmitMessage = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL2);  
if(TransmitMessage != NULL)  
{  
    // Write a message to buffer  
  
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL2);  
    PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL2);  
}
```

5.6.4.5.4.5 PLIB_CAN_ChannelEventDisable Function

C

```
void PLIB_CAN_ChannelEventDisable(  
    CAN_MODULE_ID index,  
    CAN_CHANNEL channel,  
    CAN_CHANNEL_EVENT flags  
);
```

Description

This function disables channel level events. Any enabled channel event will cause a CAN module event.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel
flags	Identifies the CAN channel event(s) to be affected. Several events can be controlled by logically ORed combination of events.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsChannelEventEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_CAN_ChannelEventDisable(CAN_ID_1, CAN_CHANNEL1, CAN_TX_CHANNEL_EMPTY);
```

5.6.4.5.5 Event Management

5.6.4.5.5.1 PLIB_CAN_ModuleEventClear Function

C

```
void PLIB_CAN_ModuleEventClear(  
    CAN_MODULE_ID index,  
    CAN_MODULE_EVENT flags  
);
```

Description

This function clears module level events. If the event condition is persistent, clearing the event will have no effect. The event condition itself should be cleared. The CAN_SYSTEM_ERROR_EVENT can only be cleared by disabling the CAN module using the [PLIB_CAN_Enable](#) function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
flags	Identifies the CAN module level events to be affected. Several events can be cleared by specifying a logically ORed combination of events.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsModuleEventClear](#) in your application to determine whether this feature is available.

Example

```
// Clear CAN_ID_1 Transmit Event and Receive Events.  
  
PLIB_CAN_ModuleEventClear(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
```

5.6.4.5.5.2 PLIB_CAN_ModuleEventDisable Function

C

```
void PLIB_CAN_ModuleEventDisable(  
    CAN_MODULE_ID index,  
    CAN_MODULE_EVENT flags  
);
```

Description

This function disables module level events. Any enabled module event will cause the CAN module to generate a CPU interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
flags	Identifies the CAN module level events to be affected. Several events can be controlled by logically ORed combination of events.

Returns

None.

Remarks

An event can be active regardless of it being enabled or disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsModuleEventEnable](#) in your application to determine whether this feature is available.

Example

```
// Enable CAN_ID_1 Transmit Event and Receive Events.  
// Disable Receive Overflow event and operation  
// mode change event.  
  
PLIB_CAN_ModuleEventDisable(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));  
PLIB_CAN_ModuleEventDisable(CAN_ID_1, (CAN_OPERATION_MODE_CHANGE_EVENT |  
    CAN_RX_OVERFLOW_EVENT));
```

5.6.4.5.5.3 PLIB_CAN_ModuleEventEnable Function**C**

```
void PLIB_CAN_ModuleEventEnable(  
    CAN_MODULE_ID index,  
    CAN_MODULE_EVENT flags  
);
```

Description

This function enables module level events. Any enabled module event will cause the CAN module to generate a CPU interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
flags	Identifies the CAN module level events to be affected. Several events can be controlled by logically ORed combination of events.

Returns

None.

Remarks

An event can be active regardless of it being enabled or disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsModuleEventEnable](#) in your application to determine whether this feature is available.

Example

```
// Enable CAN_ID_1 Transmit Event and Receive Events.  
// Disable Receive Overflow event and operation
```

```
// mode change event.
```

```
PLIB_CAN_ModuleEventEnable(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
PLIB_CAN_ModuleEventEnable(CAN_ID_1, (CAN_OPERATION_MODE_CHANGE_EVENT |
    CAN_RX_OVERFLOW_EVENT));
```

5.6.4.5.5.4 PLIB_CAN_ModuleEventGet Function

C

```
CAN_MODULE_EVENT PLIB_CAN_ModuleEventGet(
    CAN_MODULE_ID index
);
```

Description

This function returns the status of the CAN module events. The routine returns a status word. This word should be logically ANDed with the desired [CAN_MODULE_EVENT](#) event mask. A non-zero result of such an operation would mean that the events specified in the event mask are active. An event mask can contain one event or can be a logical OR combination of multiple events.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

A status word representing the status of the CAN module events.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsModuleEventClear](#) in your application to determine whether this feature is available.

Example

```
// Check if the CAN_ID_1 Module Receive event
// or if Transmit event is active

if((PLIB_CAN_ModuleEventGet(CAN_ID_1) & (CAN_RX_EVENT | CAN_TX_EVENT)) != 0)
{
    // Handle the Receive or Transmit module Event here.
}

// Check if the CAN_ID_2 System Error Event
// is active

if((PLIB_CAN_ModuleEventGet(CAN_ID_2) & CAN_SYSTEM_ERROR_EVENT) != 0)
{
    // CAN_ID_2 System error event is active.
}
```

5.6.4.5.5.5 PLIB_CAN_AllChannelEventsGet Function

C

```
CAN_CHANNEL_MASK PLIB_CAN_AllChannelEventsGet(
    CAN_MODULE_ID index
);
```

Description

This function returns a value representing the event status of all of the CAN channels. The return value can be logically ANDed with a [CAN_CHANNEL_MASK](#) type to check whether the channel has any active events. Only an enabled channel event will cause the bit to be updated.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

Returns a value that can be logically ANDed with a [CAN_CHANNEL_MASK](#) mask value to check if any event on a channel is active.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsAllChannelEvents](#) in your application to determine whether this feature is available.

Example

```
// Check if Channel 2 or 3 of CAN_ID_1 module
// have any active events

CAN_CHANNEL_MASK channelEvents;

channelEvents = PLIB_CAN_AllChannelEventsGet(CAN_ID_1);

if((channelEvents & (CAN_CHANNEL2_MASK | CAN_CHANNEL3_MASK)) != 0)
{
    // Either Channel 2 or 3 has an active event.
    // The PLIB_CAN_ChannelEventGet function can be
    // used to query the channel for more details.
}

// Check if Channel 31 of CAN_ID_2 module
// has an any active events

channelEvents = PLIB_CAN_AllChannelEventsGet(CAN_ID_2);

if((channelEvents & CAN_CHANNEL31_MASK) != 0)
{
    // Channel 31 of CAN_ID_2 module
    // has an active event.
}
```

5.6.4.5.5.6 PLIB_CAN_AllChannelOverflowStatusGet Function**C**

```
CAN_CHANNEL_MASK PLIB_CAN_AllChannelOverflowStatusGet(
    CAN_MODULE_ID index
);
```

Description

This function returns a value representing the Receive overflow event status of all the CAN Channels. The return value can be logically ANDed with a [CAN_CHANNEL_MASK](#) type to check whether a channel has any active receive overflow events. The return value will reflect the channel status only if the receive overflow event of the channel is enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

Returns a value that can be logically ANDed with a [CAN_CHANNEL_MASK](#) mask value to check if a receive channel overflow event is active.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsAllChannelOverflow](#) in your application to determine whether this feature is available.

Example

```
// Check if Receive Channel 2 or 3 of CAN_ID_1 module
// have overflowed

CAN_CHANNEL_MASK channelOverflowEvent;

channelOverflowEvent = PLIB_CAN_AllChannelOverflowStatusGet(CAN_ID_1);

if((channelOverflowEvent & (CAN_CHANNEL2_MASK | CAN_CHANNEL3_MASK)) != 0)
{
    // Either Receive Channel 2 or 3 has overflowed.
    // The PLIB_CAN_ChannelEventGet function can be
    // used to query the channel for more details.
}

// Check if Receive Channel 31 of CAN_ID_2 module
// has overflowed

channelOverflowEvent = PLIB_CAN_AllChannelOverflowStatusGet(CAN_ID_2);

if((channelOverflowEvent & CAN_CHANNEL31_MASK) != 0)
{
    // Channel 31 of CAN_ID_2 module
    // has overflowed.
}
```

5.6.4.5.5.7 PLIB_CAN_ChannelEventClear Function**C**

```
void PLIB_CAN_ChannelEventClear(
    CAN_MODULE_ID index,
    CAN_CHANNEL channel,
    CAN_CHANNEL_EVENT events
);
```

Description

This function clears channel events. The events to be cleared are specified as mask. Note that only the receive overflow event is clearable. Attempting to clear other events will have no effect since these events are persistent and clear only when the event condition is cleared.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel
events	Mask specifying the events to be cleared

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsChannelEvent](#) in your application to determine whether this feature is available.

Example

```
// Clear CAN_ID_2 Receive Channel 3 overflow event  
PLIB_CAN_ChannelEventClear(CAN_ID_2, CAN_CHANNEL3, CAN_RX_CHANNEL_OVERFLOW);
```

5.6.4.5.8 PLIB_CAN_ChannelEventEnable Function**C**

```
void PLIB_CAN_ChannelEventEnable(  
    CAN_MODULE_ID index,  
    CAN_CHANNEL channel,  
    CAN_CHANNEL_EVENT flags  
);
```

Description

This function enables channel level events. Any enabled channel event will cause a CAN module event. An event can be active regardless of it being enabled or disabled. Enabling a transmit type of event for a receive channel will have no any effect.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel
flags	Identifies the CAN channel event(s) to be affected. Several events can be controlled by logically ORed combination of events.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsChannelEventEnable](#) in your application to determine whether this feature is available.

Example

```
// CAN_ID_1 Channel 1 and 3 are Transmit channels and Channel 2 and 4 are  
// Receive channels. Enable Channel 1 empty event and channel not full  
// event. Disable Channel 2 full and overflow event.  
// Enable all Transmit events on Channel 2 and disable all Receive events on  
// on Channel 4.
```

```

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL1, (CAN_TX_CHANNEL_EMPTY |
    CAN_TX_CHANNEL_NOT_FULL));

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL2, (CAN_RX_CHANNEL_FULL |
    CAN_RX_CHANNEL_OVERFLOW));

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL3, CAN_TX_CHANNEL_ANY_EVENT);
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL4, CAN_RX_CHANNEL_ANY_EVENT);

```

5.6.4.5.9 PLIB_CAN_ChannelEventGet Function

C

```

CAN_CHANNEL_EVENT PLIB_CAN_ChannelEventGet(
    CAN_MODULE_ID index,
    CAN_CHANNEL channel
);

```

Description

This function returns a value representing the event status of a CAN channel. The return value can be logically ANDed with [CAN_CHANNEL_EVENT](#) type to check for a specific event(s). Channels events are affected regardless of whether the event itself is enabled or disabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel

Returns

Returns a value that can be logically ANDed with a [CAN_CHANNEL_EVENT](#) type to check if specific CAN channel events are active.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsChannelEvent](#) in your application to determine whether this feature is available.

Example

```

// Check if Receive Channel 2 of CAN_ID_1 module
// is not empty or if its full.

CAN_CHANNEL_EVENT channelEvent;

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL2);

if((channelEvent & (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_FULL)) != 0)
{
    // This means that either Receive Channel 2 is not empty
    // or the Channel is full.

}

// Check if Transmit Channel 3 of CAN_ID_2 module
// has any active events

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_2, CAN_CHANNEL3);

if((channelEvent & CAN_TX_CHANNEL_ANY_EVENT) != 0)

```

```
{
    // This means that some event is active
}

// Check if Transmit Channel 6 of CAN_ID_2 module is not full
channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_2, CAN_CHANNEL6);
if((channelEvent & CAN_TX_CHANNEL_NOT_FULL) != 0)
{
    // This means the Channel is not full.
}
}
```

5.6.4.5.5.10 PLIB_CAN_PendingEventsGet Function

C

```
CAN_EVENT_CODE PLIB_CAN_PendingEventsGet(
    CAN_MODULE_ID index
);
```

Description

This function returns a value representing the highest priority active event in the module. The return value is a CAN_EVENT_CODE type.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

Returns a CAN_EVENT_CODE type representing the highest priority active event in the module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsPendingEventsGet](#) in your application to determine whether this feature is available.

Example

```
// Implement a switch to check and process
// any active CAN module events.

CAN_EVENT_CODE eventCode;

eventCode = PLIB_CAN_PendingEventsGet(CAN_ID_1);

switch(eventCode)
{
    case CAN_NO_EVENT:
        // Procedure to handle no CAN events
        break;

    case CAN_WAKEUP_EVENT:
        // Procedure to handle device wake-up
        // on CAN bus activity event
        break;

    default:
        break;
}
```



```
}
```

5.6.4.5.6 Message Transmit Functions

5.6.4.5.6.1 PLIB_CAN_PendingTransmissionsAbort Function

C

```
void PLIB_CAN_PendingTransmissionsAbort(  
    CAN_MODULE_ID index,  
    CAN_CHANNEL channel  
);
```

Description

This function aborts any pending transmit operations. Any messages that are yet to be transmitted will not be transmitted. The messages will still be present in the respective channel. Any message that is in the process of being transmitted will be transmitted completely. Either one channel or all channels can be specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel. By specifying CAN_ALL_CHANNELS, transmission on all transmit channels will be aborted.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsPendingTransmissionsAbort](#) in your application to determine whether this feature is available.

Example

```
// Abort any pending transmissions on CAN_ID_1 Channel 4 and  
// Channel 5.  
PLIB_CAN_PendingTransmissionsAbort(CAN_ID_1, CAN_CHANNEL4);  
PLIB_CAN_PendingTransmissionsAbort(CAN_ID_1, CAN_CHANNEL5);
```

5.6.4.5.6.2 PLIB_CAN_TransmissionIsAborted Function

C

```
bool PLIB_CAN_TransmissionIsAborted(  
    CAN_MODULE_ID index,  
    CAN_CHANNEL channel  
);
```

Description

This function returns 'true' if the transmit abort operation is complete. Either individual channels can be specified or all channels can be specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel. By specifying CAN_ALL_CHANNELS the status of transmit abort on all transmit channels will be returned.

Returns

- true = If channel = CAN_ALL_CHANNELS, Transmit Abort is complete = If channel = CAN_CHANNELx, Transmit Abort was successful
- false = If channel = CAN_ALL_CHANNELS, Transmit Abort is in progress = If channel = CAN_CHANNELx, Transmit Abort was not successful

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTransmissionIsAborted](#) in your application to determine whether this feature is available.

Example

```
// Abort any pending transmissions on CAN_ID_1 Channel 4 and
// check if the current message transmission was aborted.

PLIB_CAN_PendingTransmissionsAbort(CAN_ID_1, CAN_CHANNEL4);

if(PLIB_CAN_TransmissionIsAborted(CAN_ID_1, CAN_CHANNEL4) == false)
{
    // The message was not aborted.
}
else
{
    // The message was aborted.
}
```

5.6.4.5.6.3 PLIB_CAN_TransmitBufferGet Function**C**

```
CAN_TX_MSG_BUFFER * PLIB_CAN_TransmitBufferGet(
    CAN_MODULE_ID index,
    CAN_CHANNEL channel
);
```

Description

This function returns a pointer to an empty transmit buffer. The routine will return a NULL pointer if there aren't any empty transmit buffers. In such a case, the application should flush the channel and wait until the transmit channel has at least one empty buffer. In order to function correctly, it is essential that the [PLIB_CAN_ChannelUpdate](#) function is called in the proper sequence for the PLIB_CAN_TransmitBufferGet function to return a pointer to an empty buffer.

Preconditions

[PLIB_CAN_MemoryBufferAssign](#) must be called if the 'transmit buffer' should be in the device RAM. It is not required if the 'transmit buffer' is in SFR space.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel

Returns

Returns a [CAN_TX_MSG_BUFFER](#) type pointer to an empty message buffer in the Transmit channel. Returns NULL if the channel is full and there aren't any empty message buffers.

Remarks

Calling the `PLIB_CAN_TransmitBufferGet` function on a channel that has not been updated after a message was written to the channel, will cause the function to return a pointer to the written message in case of transmit buffer FIFO. Therefore, a non-transmitted message could get overwritten.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTransmitBufferGet](#) in your application to determine whether this feature is available.

Example

```
// Transmit a message through CAN_ID_1 Channel 4

CAN_TX_MSG_BUFFER * msgBuffer;

msgBuffer = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL4);

if(msgBuffer != NULL)
{
    // Load the message here
}
else
{
    // No space in the channel
    // wait until a message
    // buffer is free.

    while((PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL4) &
        CAN_TX_CHANNEL_NOT_FULL) == 0);
}
```

5.6.4.5.6.4 PLIB_CAN_TransmitChannelFlush Function

C

```
void PLIB_CAN_TransmitChannelFlush(
    CAN_MODULE_ID index,
    CAN_CHANNEL channel
);
```

Description

This function causes all messages in the specified transmit channel to be transmitted. All messages in the channel at the time of the flush operation will be transmitted. The transmit channel flush operation should preferably be called immediately after the [PLIB_CAN_ChannelUpdate](#) function. This will ensure that messages are transmitted promptly.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTransmitChannelFlush](#) in your application to determine whether this feature is available.

Example

```
// Flush CAN_ID_1 Transmit Channel 4.

PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL4);
```

5.6.4.5.6.5 PLIB_CAN_TransmitChannelStatusGet Function**C**

```
CAN_TX_CHANNEL_STATUS PLIB_CAN_TransmitChannelStatusGet(
    CAN_MODULE_ID index,
    CAN_CHANNEL channel
);
```

Description

This function returns the condition of the transmit channel. The return value can be logically ANDed with [CAN_TX_CHANNEL_STATUS](#) type values.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel

Returns

Returns a status word that can be logically ANDed with the [CAN_TX_CHANNEL_STATUS](#) type to check whether a condition exists.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTransmitChannelStatus](#) in your application to determine whether this feature is available.

Example

```
// Check if CAN_ID_1 Transmit Channel 4
// is still transmitting

CAN_TX_CHANNEL_STATUS status;

status = PLIB_CAN_TransmitChannelStatusGet(CAN_ID_1, CAN_CHANNEL4);

if((status & CAN_TX_CHANNEL_TRANSMITTING) != 0)
{
    // This means that channel is still
    // transmitting.
}

// Check if CAN_ID_2 Transmit Channel 5 has lost arbitration
// or other Transmit errors.

status = PLIB_CAN_TransmitChannelStatusGet(CAN_ID_2, CAN_CHANNEL5);

if((status & (CAN_TX_CHANNEL_ARBITRATION_LOST | CAN_TX_CHANNEL_ERROR)) != 0)
```

```
{  
    // This means that the Transmit channel has either  
    // lost arbitration or a Transmit error has occurred.  
}
```

5.6.4.5.6 PLIB_CAN_TransmitErrorCountGet Function

C

```
int PLIB_CAN_TransmitErrorCountGet(  
    CAN_MODULE_ID index  
);
```

Description

This function returns the CAN transmit error count. Refer to CAN 2.0B specification for more details on the CAN transmit error count and its significance.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

Returns the CAN transmit error count.

Remarks

There are multiple bus conditions, which could cause the transmit error count to increase. Please refer to the CAN 2.0B specification for details.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsTransmitErrorCountGet](#) in your application to determine whether this feature is available.

Example

```
// Check if the CAN_ID_1 Transmit error count is more than 200  
int errorCount;  
  
errorCount = PLIB_CAN_TransmitErrorCountGet(CAN_ID_1);  
if(errorCount > 200)  
{  
    // This error count is high.  
    // Do some diagnostics.  
}
```

5.6.4.5.7 Message Receive Functions

5.6.4.5.7.1 PLIB_CAN_ReceivedMessageGet Function

C

```
CAN_RX_MSG_BUFFER * PLIB_CAN_ReceivedMessageGet(  
    CAN_MODULE_ID index,  
    CAN_CHANNEL channel  
);
```

Description

This function returns a [CAN_RX_MSG_BUFFER](#) pointer to a message to be read from the CAN channel. The routine will return

NULL if there are no new messages to be read. The [PLIB_CAN_ChannelUpdate](#) routine should be called after the received message has been processed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module.
channel	Identifies the desired CAN Receive channel.

Returns

Returns NULL if there are no messages in the channel; otherwise returns a [CAN_RX_MSG_BUFFER](#) type pointer to a received CAN message. If the receive channel is a full CAN message receive channel, the caller should use the msgSID, msgEID and data members of the [CAN_RX_MSG_BUFFER](#) data structure to access the received CAN message. If the receive channel is a data-only channel, the message will only contain 8 payload data bytes (even if the message was placed on the bus with less than 8 bytes). The caller in this case should use the dataOnlyMsgData member of the [CAN_RX_MSG_BUFFER](#) data structure to read the data contained in the received CAN message.

Remarks

The CAN receive channel is configured as a full CAN message receive channel or a data-Only channel while configuring the channel using the [PLIB_CAN_ChannelForReceiveSet](#) function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsReceivedMessageGet](#) in your application to determine whether this feature is available.

Example

```
// Read a message from the CAN_ID_1 Channel 8
// which is configured as full CAN message
// receive channel.

CAN_RX_MSG_BUFFER * receivedMsg;

receivedMsg = (CAN_RX_MSG_BUFFER *)PLIB_CAN_ReceivedMessageGet(CAN_ID_1,
    CAN_CHANNEL8);

if(receivedMsg != NULL)
{
    // rxMsg is pointing to
    // a CAN message. Process
    // the message and then update
    // the CAN channel.

    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL8);
}

// Read a message from the CAN_ID_2 Channel 9
// which is configured as data only message
// receive channel. Access the message
// as bytes;

CAN_RX_MSG_BUFFER * rxMsg;

rxMsg = PLIB_CAN_ReceivedMessageGet(CAN_ID_2, CAN_CHANNEL9);

if(rxMsg != NULL)
{
    // rxMsg is pointing to
    // a CAN message. Process
```

```
// the message and then update
// the CAN channel.

// rxMsg->dataOnlyMsgData[0] is the first byte of message
// data payload. rxMsg->dataOnlyMsgData[0] is the second
// byte of message data payload and so on.
```

```
    PLIB_CAN_ChannelUpdate(CAN_ID_2, CAN_CHANNEL9);
}
```

5.6.4.5.7.2 PLIB_CAN_RxFIFOFourLeftNotify Function

C

```
void PLIB_CAN_RxFIFOFourLeftNotify(
    CAN_MODULE_ID index
);
```

Description

This function configures the module to generate an interrupt when the FIFO has four receive buffers left. If this API is not called, the module will interrupt if four receive buffers are empty.

Preconditions

The functional mode of the module should be configured for "CAN_ENHANCED_FIFO_MODE2" using [PLIB_CAN_FunctionalModeSelect](#) function.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

The total number of CAN Masks per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsRxFIFOFourLeftNotify](#) in your application to determine whether this feature is available.

Example

```
PLIB_CAN_RxFIFOFourLeftNotify(CAN_ID_1);
```

5.6.4.5.7.3 PLIB_CAN_RxFIFOOneLeftNotify Function

C

```
void PLIB_CAN_RxFIFOOneLeftNotify(
    CAN_MODULE_ID index
);
```

Description

This function configures the module to generate an interrupt when the FIFO has only one receive buffer left. If this API is not called, the module will interrupt if four receive buffers are empty.

Preconditions

The functional mode of the module should be configured for "CAN_ENHANCED_FIFO_MODE2" using [PLIB_CAN_FunctionalModeSelect](#) function.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

The total number of CAN Masks per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsRxFIFOOneLeftNotify](#) in your application to determine whether this feature is available.

Example

```
PLIB_CAN_RxFIFOOneLeftNotify(CAN_ID_1);
```

5.6.4.5.7.4 PLIB_CAN_ReceiveModeSelect Function**C**

```
void PLIB_CAN_ReceiveModeSelect(  
    CAN_MODULE_ID index,  
    CAN_RECEIVE_CHANNEL rcvChannel,  
    CAN_RECEIVE_MODES recMode  
);
```

Description

This function selects the receive mode of the CAN module.

Preconditions

The functional mode should support the 'receive mode' selected.

Parameters

Parameters	Description
index	Identifies the desired CAN module
rcvChannel	Receive channel number, one of the possible values from CAN_RECEIVE_CHANNEL
recMode	Receive mode, one of the possible values from CAN_RECEIVE_MODES

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsReceiveModeSelect](#) in your application to determine whether this feature is available.

Example

```
PLIB_CAN_FunctionalModeSelect(CAN_ID_1, CAN_LEGACY_MODE0);  
  
PLIB_CAN_ReceiveModeSelect(CAN_ID_1, CAN_RECEIVE_CHANNEL0, CAN_RECEIVE_ALL_MSG_MODE);
```

5.6.4.5.7.5 PLIB_CAN_ReceiveModeGet Function**C**

```
CAN_RECEIVE_MODES PLIB_CAN_ReceiveModeGet(  
    CAN_MODULE_ID index,  
    CAN_RECEIVE_CHANNEL rcvChannel  
);
```


Description

This function obtains the receive mode of the CAN module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
rcvChannel	Receive channel number, one of the possible values from CAN_RECEIVE_CHANNEL

Returns

Receive mode, one of the possible values from [CAN_RECEIVE_MODES](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsReceiveModeSelect](#) in your application to determine whether this feature is available.

Example

```
CAN_RECEIVE_MODES recMode;

recMode = PLIB_CAN_ReceiveModeGet(CAN_ID_1, CAN_RECEIVE_CHANNEL0);
```

5.6.4.5.7.6 PLIB_CAN_MessageIsReceived Function**C**

```
bool PLIB_CAN_MessageIsReceived(
    CAN_MODULE_ID index,
    CAN_RECEIVE_CHANNEL rcvChannel
);
```

Description

This function returns 'true' if a message was received and the receive buffer is not empty.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
rcvChannel	Receive channel number, one of the possible values from CAN_RECEIVE_CHANNEL

Returns

- true = A message was received for the specified channel
- false = The receive buffer is not empty for the specified channel

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsMessageIsReceived](#) in your application to determine whether this feature is available.

Example

```
if (PLIB_CAN_MessageIsReceived(CAN_ID_1, CAN_RECEIVE_CHANNEL0))
```

```
{  
    //Read the message  
  
    //Acknowledge read; otherwise, it will cause the interrupt to  
    //generated again  
    PLIB_CAN_ReceiveBufferClear(CAN_ID_1, CAN_RECEIVE_CHANNEL0);  
}
```

5.6.4.5.7.7 PLIB_CAN_ReceiveBufferIsEmpty Function

C

```
bool PLIB_CAN_ReceiveBufferIsEmpty(  
    CAN_MODULE_ID index,  
    CAN_RECEIVE_CHANNEL rcvChannel  
);
```

Description

This function returns 'true' if the receive buffer of the specified channel is empty.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
rcvChannel	Receive channel number, one of the possible values from CAN_RECEIVE_CHANNEL

Returns

- true = The receive buffer is not empty for the specified channel
- false = A message was received for the specified channel

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsReceiveBufferIsEmpty](#) in your application to determine whether this feature is available.

Example

```
if (PLIB_CAN_ReceiveBufferIsEmpty(CAN_ID_1, CAN_RECEIVE_CHANNEL0))  
{  
    //Buffer is empty  
}
```

5.6.4.5.7.8 PLIB_CAN_ReceiveBufferClear Function

C

```
void PLIB_CAN_ReceiveBufferClear(  
    CAN_MODULE_ID index,  
    CAN_RECEIVE_CHANNEL rcvChannel  
);
```

Description

This function clears the received message flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
rcvChannel	Receive channel number, one of the possible values from CAN_RECEIVE_CHANNEL

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsReceiveBufferClear](#) in your application to determine whether this feature is available.

This API should be called once the receive message buffer is read. This function must be called once the receive message buffer has been read or the interrupt request will still be active and the interrupt may occur again immediately.

Example

```
if (PLIB_CAN_MessageIsReceived(CAN_ID_1, CAN_RECEIVE_CHANNEL0))
{
    //Read the message

    //Acknowledge read, otherwise it will cause the interrupt to
    //be generated again
    PLIB_CAN_ReceiveBufferClear(CAN_ID_1, CAN_RECEIVE_CHANNEL0);
}
```

5.6.4.5.7.9 PLIB_CAN_RemoteTransmitReqIsReceived Function**C**

```
bool PLIB_CAN_RemoteTransmitReqIsReceived(
    CAN_MODULE_ID index,
    CAN_RECEIVE_CHANNEL rcvChannel
);
```

Description

This function returns 'true' if the message received is a remote transmit request.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
rcvChannel	Receive channel number, one of the possible values from CAN_RECEIVE_CHANNEL

Returns

- true = Received message is a remote transmission request
- false = Received message is not a remote transmission request

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsRemoteTransmitReq](#) in your application to determine whether this feature is available.

Example

```
if (PLIB_CAN_RemoteTransmitReqIsReceived(CAN_ID_1, CAN_RECEIVE_CHANNEL0))
{
    //Request for remote transmit
}
```

5.6.4.5.8 Message Filtering Functions

5.6.4.5.8.1 PLIB_CAN_FilterConfigure Function

C

```
void PLIB_CAN_FilterConfigure(
    CAN_MODULE_ID index,
    CAN_FILTER filter,
    uint32_t id,
    CAN_ID_TYPE filterType
);
```

Description

This function configures a CAN message acceptance filter. The ID field of the incoming message must match the filter bits for the CAN module to **accept** the message. A filter can be a EID type filter, which filters EID messages or a SID filter, which in turn filters SID messages. The filter mask bits (configured using the [PLIB_CAN_FilterMaskConfigure](#) function) additionally allow specified message ID bits to be ignored in the filtering process.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN receive filter
id	A value in the range 0x0 to 0x7FF for SID filter type or 0x0 to 0x1FFFFFFF for EID filter type.

5.6.4.5.8.2 PLIB_CAN_FilterDisable Function

C

```
void PLIB_CAN_FilterDisable(
    CAN_MODULE_ID index,
    CAN_FILTER filter
);
```

Description

This function disables a CAN message acceptance filter. At least one filter must be enabled for the CAN module to receive messages. A receive channel associated with a filter will not receive messages unless the filter is enabled. After a filter is disabled, the [PLIB_CAN_FilterIsDisabled](#) function should be called to verify that the filter is disabled. A filter must be disabled before it can be configured.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN Message Acceptance Filter

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsFilterEnable](#) in your application to determine whether this feature is available.

Example

```
// Enable filter 6 of CAN_ID_2
PLIB_CAN_FilterDisable (CAN_ID_2, CAN_FILTER6);

while(PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER4));
```

5.6.4.5.8.3 PLIB_CAN_FilterEnable Function**C**

```
void PLIB_CAN_FilterEnable(
    CAN_MODULE_ID index,
    CAN_FILTER filter
);
```

Description

This function enables a CAN message acceptance filter. At least one filter must be enabled for the CAN module to receive messages. A receive channel associated with a filter will not receive messages unless the filter is enabled. After a filter is disabled, the [PLIB_CAN_FilterIsDisabled](#) function should be called to verify that the filter is disabled. A filter must be disabled before it can be configured.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN message acceptance filter

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsFilterEnable](#) in your application to determine whether this feature is available.

Example

```
// Enable filter 6 of CAN_ID_2
PLIB_CAN_FilterEnable (CAN_ID_2, CAN_FILTER6);
```

```
while(!PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER4));
```

5.6.4.5.8.4 PLIB_CAN_FilterIsDisabled Function

C

```
bool PLIB_CAN_FilterIsDisabled(  
    CAN_MODULE_ID index,  
    CAN_FILTER filter  
);
```

Description

Returns 'true' if the specified filter is disabled. This function should be called to check if the filter is disabled before calling the [PLIB_CAN_FilterConfigure](#) function and [PLIB_CAN_FilterToChannelLink](#) function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN filter

Returns

- true = The filter is disabled
- false = The filter is enabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsFilterEnable](#) in your application to determine whether this feature is available.

Example

```
// Disable CAN_ID_1 filter 3 and wait until the  
// filter is disabled.  
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER3);  
  
while(PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER3) == false);
```

5.6.4.5.8.5 PLIB_CAN_FilterMaskConfigure Function

C

```
void PLIB_CAN_FilterMaskConfigure(  
    CAN_MODULE_ID index,  
    CAN_FILTER_MASK mask,  
    uint32_t maskbits,  
    CAN_ID_TYPE idType,  
    CAN_FILTER_MASK_TYPE mide  
);
```

Description

This function configures a CAN filter mask. The mask bits determine which message ID bits are ignored and compared during the filtering process.

Preconditions

The CAN module should be in Configuration mode. This is achieved by using the [PLIB_CAN_OperationModeSelect](#) function.

Parameters

Parameters	Description
index	Identifies the desired CAN module
mask	Identifies the desired CAN receive filter mask.
maskbits	A value in the range 0x0 to 0x7FF for SID range or 0x0 to 0x1FFFFFFF for the EID range. Each set bit will mean that the corresponding bit in the filter will be compared to the corresponding bit in the message ID. A clear mask bit means the corresponding bit in the incoming message ID field will be ignored.

5.6.4.5.8.6 PLIB_CAN_FilterToChannelLink Function**C**

```
void PLIB_CAN_FilterToChannelLink(
    CAN_MODULE_ID index,
    CAN_FILTER filter,
    CAN_FILTER_MASK mask,
    CAN_CHANNEL channel
);
```

Description

This function links a filter to a channel. A filter is typically linked to a receive channel. This allows the channel to receive messages accepted by the filter. A filter can also be linked to a transmit channel if the transmit channel is configured for remote request transmit. In this case, a message accepted by the filter will automatically cause the linked transmit channel to transmit CAN messages that are buffered in the channel. Note that a filter should be enabled using the [PLIB_CAN_FilterEnable](#) function after the filter has been linked to the desired channel.

Preconditions

Filter should have been disabled using the [PLIB_CAN_FilterDisable](#) function.

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN Filter
mask	Identifies the mask to be used with this filter
channel	Identifies the channel to be linked to this filter. If a transmit channel is linked, the transmit channel should have its remote transmit request feature enabled.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsFilterToChannelLink](#) in your application to determine whether this feature is available.

Example

```
// Configure CAN_ID_1 filter 3 to accept extended ID messages
// with EID 0x1D400 and link the filter to CAN_ID_1 Channel 5.
// Note the sequence in which the steps are performed.
// Disable the filter and check if its disabled.
// Configure the filter. Link it to the Channel and then
// enable the filter.

PLIB_CAN_FilterDisable(CAN_ID_1, CAN_FILTER3);

while(PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER3) == false);
```

```
PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER3, 0x1D400, CAN_EID);

PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER3, CAN_FILTER_MASK0,
    CAN_CHANNEL5);

PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER3);
```

5.6.4.5.8.7 PLIB_CAN_LatestFilterMatchGet Function

C

```
CAN_FILTER PLIB_CAN_LatestFilterMatchGet(
    CAN_MODULE_ID index
);
```

Description

This function returns the index of the filter that accepted the latest message.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

Index of the filter that accepted the latest message

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsLatestFilterMatchGet](#) in your application to determine whether this feature is available.

Example

```
// Check if CAN_ID_2 module Receive Buffer event
// is active and if so check which filter
// accepted the message.

CAN_FILTER filter;

if((PLIB_CAN_ModuleEventGet(CAN_ID_1) & CAN_RX_EVENT) != 0)
{
    // Check which filter accepted the message

    filter = PLIB_CAN_LatestFilterMatchGet(CAN_ID_1);
}
```

5.6.4.5.9 Error State Tracking

5.6.4.5.9.1 PLIB_CAN_ReceiveErrorCountGet Function

C

```
int PLIB_CAN_ReceiveErrorCountGet(
    CAN_MODULE_ID index
);
```

Description

This function returns the CAN receive error count. Refer to the CAN 2.0B specification for more details on the CAN receive error

count and its significance.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

Returns the CAN receive error count.

Remarks

There are multiple bus conditions, which could cause the receive error count to increase. Please refer to the CAN 2.0b specification for details.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsReceiveErrorCount](#) in your application to determine whether this feature is available.

Example

```
// Check if CAN_ID_1 Receive error count is more than 200.
int errCount;

errCount = PLIB_CAN_ReceiveErrorCountGet(CAN_ID_1);
if(errCount > 200)
{
    // This error count is high.
    // Do some diagnostics.
}
```

5.6.4.5.9.2 PLIB_CAN_ErrorStateGet Function

C

```
CAN_ERROR_STATE PLIB_CAN_ErrorStateGet(
    CAN_MODULE_ID index
);
```

Description

This function returns the CAN error status word. The return word can be logically ANDed with the desired [CAN_ERROR_STATE](#) member to check if the CAN module is in a specific error state.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

Returns the [CAN_ERROR_STATE](#) word, which can be logically ANDed with the desired [CAN_ERROR_STATE](#) member to check whether the CAN module is in a specific error state.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsErrorState](#) in your application to determine whether this feature is available.

Example

```

// Check if CAN_ID_1 is in the Transmit or Receive warning state.
CAN_ERROR_STATE errorState;

errorState = PLIB_CAN_ErrorStateGet(CAN_ID_1);

if((errorState & CAN_TX_RX_WARNING_STATE) != 0)
{
    // CAN_ID_1 is in either Transmit or Receive warning state.
}
// Check if CAN_ID_2 is in the Receive Bus Passive or Transmit Bus passive
// state.
errorState = PLIB_CAN_ErrorStateGet(CAN_ID_2);

if((errorState & (CAN_TX_BUS_PASSIVE_STATE | CAN_RX_BUS_PASSIVE_STATE)) != 0)
{
    // This means that the CAN module is in Transmit Bus Passive
    // or Receive Bus Passive state.
}

```

5.6.4.5.10 Information Functions**5.6.4.5.10.1 PLIB_CAN_TotalChannelsGet Function****C**

```

char PLIB_CAN_TotalChannelsGet(
    CAN_MODULE_ID index
);

```

Description

This function returns the total number of CAN channels per CAN module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

The total number of CAN channels per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsModuleInfo](#) in your application to determine whether this feature is available.

Example

```

char totalChannels;

totalChannels = PLIB_CAN_TotalChannelsGet(CAN_ID_1);

```

5.6.4.5.10.2 PLIB_CAN_TotalFiltersGet Function**C**

```

char PLIB_CAN_TotalFiltersGet(
    CAN_MODULE_ID index
);

```

Description

This function returns the total number of CAN filters per CAN module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

The total number of CAN Filters per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsModuleInfo](#) in your application to determine whether this feature is available.

Example

```
char totalFilters;  
  
totalFilters = PLIB_CAN_TotalFiltersGet(CAN_ID_1);
```

5.6.4.5.10.3 PLIB_CAN_TotalMasksGet Function**C**

```
char PLIB_CAN_TotalMasksGet(  
    CAN_MODULE_ID index  
);
```

Description

This function returns the total number of CAN masks per CAN module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired CAN module

Returns

The total number of CAN Masks per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CAN_ExistsModuleInfo](#) in your application to determine whether this feature is available.

Example

```
char totalMasks;  
  
totalMasks = PLIB_CAN_TotalMasksGet(CAN_ID_1);
```

5.6.4.5.11 Data Types and Constants

5.6.4.5.11.1 CAN_CHANNEL Enumeration

C

```
typedef enum {  
    CAN_CHANNEL0,  
    CAN_CHANNEL1,  
    CAN_CHANNEL2,  
    CAN_CHANNEL3,  
    CAN_CHANNEL4,  
    CAN_CHANNEL5,  
    CAN_CHANNEL6,  
    CAN_CHANNEL7,  
    CAN_CHANNEL8,  
    CAN_CHANNEL9,  
    CAN_CHANNEL10,  
    CAN_CHANNEL11,  
    CAN_CHANNEL12,  
    CAN_CHANNEL13,  
    CAN_CHANNEL14,  
    CAN_CHANNEL15,  
    CAN_CHANNEL16,  
    CAN_CHANNEL17,  
    CAN_CHANNEL18,  
    CAN_CHANNEL19,  
    CAN_CHANNEL20,  
    CAN_CHANNEL21,  
    CAN_CHANNEL22,  
    CAN_CHANNEL23,  
    CAN_CHANNEL24,  
    CAN_CHANNEL25,  
    CAN_CHANNEL26,  
    CAN_CHANNEL27,  
    CAN_CHANNEL28,  
    CAN_CHANNEL29,  
    CAN_CHANNEL30,  
    CAN_CHANNEL31  
} CAN_CHANNEL;
```

Description

CAN Channel

This enumeration identifies the available CAN channels.

Members

Members	Description
CAN_CHANNEL0	Channel 0 ID
CAN_CHANNEL1	Channel 1 ID
CAN_CHANNEL2	Channel 2 ID
CAN_CHANNEL3	Channel 3 ID
CAN_CHANNEL4	Channel 4 ID
CAN_CHANNEL5	Channel 5 ID
CAN_CHANNEL6	Channel 6 ID
CAN_CHANNEL7	Channel 7 ID
CAN_CHANNEL8	Channel 8 ID
CAN_CHANNEL9	Channel 9 ID
CAN_CHANNEL10	Channel 10 ID
CAN_CHANNEL11	Channel 11 ID

CAN_CHANNEL12	Channel 12 ID
CAN_CHANNEL13	Channel 13 ID
CAN_CHANNEL14	Channel 14 ID
CAN_CHANNEL15	Channel 15 ID
CAN_CHANNEL16	Channel 16 ID
CAN_CHANNEL17	Channel 17 ID
CAN_CHANNEL18	Channel 18 ID
CAN_CHANNEL19	Channel 19 ID
CAN_CHANNEL20	Channel 20 ID
CAN_CHANNEL21	Channel 21 ID
CAN_CHANNEL22	Channel 22 ID
CAN_CHANNEL23	Channel 23 ID
CAN_CHANNEL24	Channel 24 ID
CAN_CHANNEL25	Channel 25 ID
CAN_CHANNEL26	Channel 26 ID
CAN_CHANNEL27	Channel 27 ID
CAN_CHANNEL28	Channel 28 ID
CAN_CHANNEL29	Channel 29 ID
CAN_CHANNEL30	Channel 30 ID
CAN_CHANNEL31	Channel 31 ID

5.6.4.5.11.2 CAN_CHANNEL_EVENT Enumeration

C

```
typedef enum {
    CAN_RX_CHANNEL_NOT_EMPTY,
    CAN_RX_CHANNEL_HALF_FULL,
    CAN_RX_CHANNEL_FULL,
    CAN_RX_CHANNEL_OVERFLOW,
    CAN_RX_CHANNEL_ANY_EVENT,
    CAN_TX_CHANNEL_EMPTY,
    CAN_TX_CHANNEL_HALF_EMPTY,
    CAN_TX_CHANNEL_NOT_FULL,
    CAN_TX_CHANNEL_ANY_EVENT
} CAN_CHANNEL_EVENT;
```

Description

CAN Layer 3 Interrupts

This enumerates all the leading interrupts generated due to CAN transmit and receive events. This enumeration can be used to enable or disable channel events and as a mask to check if a channel event is active.

A single or a combination of the interrupts can be logically ORed to specify the interrupt(s) to be enabled disabled or events to check.

This enumeration is used by [PLIB_CAN_ChannelEventEnable](#), [PLIB_CAN_ChannelEventDisable](#), [PLIB_CAN_ChannelEventClear](#), and [PLIB_CAN_ChannelEventGet](#) functions.

Members

Members	Description
CAN_RX_CHANNEL_NOT_EMPTY	CAN Receive Channel Not Empty Event Mask
CAN_RX_CHANNEL_HALF_FULL	CAN Receive Channel Half Full Event Mask

CAN_RX_CHANNEL_FULL	CAN Receive Channel Full Event Mask
CAN_RX_CHANNEL_OVERFLOW	CAN Receive Channel Overflow Event Mask
CAN_RX_CHANNEL_ANY_EVENT	CAN Receive Channel Any Event Mask
CAN_TX_CHANNEL_EMPTY	CAN Transmit Channel Empty Event Mask
CAN_TX_CHANNEL_HALF_EMPTY	CAN Transmit Channel Half Empty Event Mask
CAN_TX_CHANNEL_NOT_FULL	CAN Transmit Channel Not Full Event Mask
CAN_TX_CHANNEL_ANY_EVENT	CAN Transmit Channel Any Event Mask

5.6.4.5.11.3 CAN_CHANNEL_MASK Enumeration

C

```
typedef enum {
    CAN_CHANNEL0_MASK,
    CAN_CHANNEL1_MASK,
    CAN_CHANNEL2_MASK,
    CAN_CHANNEL3_MASK,
    CAN_CHANNEL4_MASK,
    CAN_CHANNEL5_MASK,
    CAN_CHANNEL6_MASK,
    CAN_CHANNEL7_MASK,
    CAN_CHANNEL8_MASK,
    CAN_CHANNEL9_MASK,
    CAN_CHANNEL10_MASK,
    CAN_CHANNEL11_MASK,
    CAN_CHANNEL12_MASK,
    CAN_CHANNEL13_MASK,
    CAN_CHANNEL14_MASK,
    CAN_CHANNEL15_MASK,
    CAN_CHANNEL16_MASK,
    CAN_CHANNEL17_MASK,
    CAN_CHANNEL18_MASK,
    CAN_CHANNEL19_MASK,
    CAN_CHANNEL20_MASK,
    CAN_CHANNEL21_MASK,
    CAN_CHANNEL22_MASK,
    CAN_CHANNEL23_MASK,
    CAN_CHANNEL24_MASK,
    CAN_CHANNEL25_MASK,
    CAN_CHANNEL26_MASK,
    CAN_CHANNEL27_MASK,
    CAN_CHANNEL28_MASK,
    CAN_CHANNEL29_MASK,
    CAN_CHANNEL30_MASK,
    CAN_CHANNEL31_MASK,
    CAN_ANYCHANNEL_MASK
} CAN_CHANNEL_MASK;
```

Description

CAN Channel Masks

This enumeration identifies a series of useful masks. Each mask represents a CAN channel. These masks are used with the [PLIB_CAN_AllChannelEventsGet](#) and [PLIB_CAN_AllChannelOverflowStatusGet](#) functions. The value returned by these functions can be logically ANDed with any of these masks to check if an event or overflow event for that channel is active.

Members

Members	Description
CAN_CHANNEL0_MASK	Channel 0 Mask
CAN_CHANNEL1_MASK	Channel 1 Mask

CAN_CHANNEL2_MASK	Channel 2 Mask
CAN_CHANNEL3_MASK	Channel 3 Mask
CAN_CHANNEL4_MASK	Channel 4 Mask
CAN_CHANNEL5_MASK	Channel 5 Mask
CAN_CHANNEL6_MASK	Channel 6 Mask
CAN_CHANNEL7_MASK	Channel 7 Mask
CAN_CHANNEL8_MASK	Channel 8 Mask
CAN_CHANNEL9_MASK	Channel 9 Mask
CAN_CHANNEL10_MASK	Channel 10 Mask
CAN_CHANNEL11_MASK	Channel 11 Mask
CAN_CHANNEL12_MASK	Channel 12 Mask
CAN_CHANNEL13_MASK	Channel 13 Mask
CAN_CHANNEL14_MASK	Channel 14 Mask
CAN_CHANNEL15_MASK	Channel 15 Mask
CAN_CHANNEL16_MASK	Channel 16 Mask
CAN_CHANNEL17_MASK	Channel 17 Mask
CAN_CHANNEL18_MASK	Channel 18 Mask
CAN_CHANNEL19_MASK	Channel 19 Mask
CAN_CHANNEL20_MASK	Channel 20 Mask
CAN_CHANNEL21_MASK	Channel 21 Mask
CAN_CHANNEL22_MASK	Channel 22 Mask
CAN_CHANNEL23_MASK	Channel 23 Mask
CAN_CHANNEL24_MASK	Channel 24 Mask
CAN_CHANNEL25_MASK	Channel 25 Mask
CAN_CHANNEL26_MASK	Channel 26 Mask
CAN_CHANNEL27_MASK	Channel 27 Mask
CAN_CHANNEL28_MASK	Channel 28 Mask
CAN_CHANNEL29_MASK	Channel 29 Mask
CAN_CHANNEL30_MASK	Channel 30 Mask
CAN_CHANNEL31_MASK	Channel 31 Mask
CAN_ANYCHANNEL_MASK	Channel any channel Mask

5.6.4.5.11.4 CAN_DNET_FILTER_SIZE Enumeration

C

```
typedef enum {
    CAN_DNET_FILTER_SIZE_18_BIT,
    CAN_DNET_FILTER_SIZE_17_BIT,
    CAN_DNET_FILTER_SIZE_16_BIT,
    CAN_DNET_FILTER_SIZE_15_BIT,
    CAN_DNET_FILTER_SIZE_14_BIT,
    CAN_DNET_FILTER_SIZE_13_BIT,
    CAN_DNET_FILTER_SIZE_12_BIT,
    CAN_DNET_FILTER_SIZE_11_BIT,
    CAN_DNET_FILTER_SIZE_10_BIT,
    CAN_DNET_FILTER_SIZE_9_BIT,
    CAN_DNET_FILTER_SIZE_8_BIT,
    CAN_DNET_FILTER_SIZE_7_BIT,
    CAN_DNET_FILTER_SIZE_6_BIT,
    CAN_DNET_FILTER_SIZE_5_BIT,
    CAN_DNET_FILTER_SIZE_4_BIT,
```

```

    CAN_DNET_FILTER_SIZE_3_BIT,
    CAN_DNET_FILTER_SIZE_2_BIT,
    CAN_DNET_FILTER_SIZE_1_BIT,
    CAN_DNET_FILTER_DISABLE
} CAN_DNET_FILTER_SIZE;

```

Description

CAN DeviceNet filter bit numbers.

This enumeration identifies the size of the DeviceNet filter in bits. If the size of the DeviceNet filter is "n", the "n" most significant bits of the data payload are compared with the EID field of enabled filters.

Members

Members	Description
CAN_DNET_FILTER_SIZE_18_BIT	Compare up to data byte 2 bit 6 with filter mask bit 17
CAN_DNET_FILTER_SIZE_17_BIT	Compare up to data byte 2 bit 7 with filter mask bit 16
CAN_DNET_FILTER_SIZE_16_BIT	Compare up to data byte 1 bit 0 with filter mask bit 15
CAN_DNET_FILTER_SIZE_15_BIT	Compare up to data byte 1 bit 1 with filter mask bit 14
CAN_DNET_FILTER_SIZE_14_BIT	Compare up to data byte 1 bit 2 with filter mask bit 13
CAN_DNET_FILTER_SIZE_13_BIT	Compare up to data byte 1 bit 3 with filter mask bit 12
CAN_DNET_FILTER_SIZE_12_BIT	Compare up to data byte 1 bit 4 with filter mask bit 11
CAN_DNET_FILTER_SIZE_11_BIT	Compare up to data byte 1 bit 5 with filter mask bit 10
CAN_DNET_FILTER_SIZE_10_BIT	Compare up to data byte 1 bit 6 with filter mask bit 9
CAN_DNET_FILTER_SIZE_9_BIT	Compare up to data byte 1 bit 7 with filter mask bit 8
CAN_DNET_FILTER_SIZE_8_BIT	Compare up to data byte 0 bit 0 with filter mask bit 7
CAN_DNET_FILTER_SIZE_7_BIT	Compare up to data byte 0 bit 1 with filter mask bit 6
CAN_DNET_FILTER_SIZE_6_BIT	Compare up to data byte 0 bit 2 with filter mask bit 5
CAN_DNET_FILTER_SIZE_5_BIT	Compare up to data byte 0 bit 3 with filter mask bit 4
CAN_DNET_FILTER_SIZE_4_BIT	Compare up to data byte 0 bit 4 with filter mask bit 3
CAN_DNET_FILTER_SIZE_3_BIT	Compare up to data byte 0 bit 5 with filter mask bit 2
CAN_DNET_FILTER_SIZE_2_BIT	Compare up to data byte 0 bit 6 with filter mask bit 1
CAN_DNET_FILTER_SIZE_1_BIT	Compare up to data byte 0 bit 7 with filter mask bit 0
CAN_DNET_FILTER_DISABLE	Do not compare data bytes, Device Net Filtering is disabled

5.6.4.5.11.5 CAN_ERROR_STATE Enumeration

C

```

typedef enum {
    CAN_TX_RX_WARNING_STATE,
    CAN_RX_WARNING_STATE,
    CAN_TX_WARNING_STATE,
    CAN_RX_BUS_PASSIVE_STATE,
    CAN_TX_BUS_PASSIVE_STATE,
    CAN_TX_BUS_OFF_STATE
} CAN_ERROR_STATE;

```

Description

CAN Error States

This enumeration identifies all of the CAN module error states events. The CAN module enters or exits an error state as the transmit and receive error counter values change. Refer to the CAN 2.0B specification for more details on the error states.

This enumeration is used with the [PLIB_CAN_ErrorStateGet](#) function.

Members

Members	Description
CAN_TX_RX_WARNING_STATE	CAN Module is in a Transmit or Receive warning state.
CAN_RX_WARNING_STATE	CAN Module is in a Receive warning state.
CAN_TX_WARNING_STATE	CAN Module is in a Transmit warning state.
CAN_RX_BUS_PASSIVE_STATE	CAN Receive is in a Bus Passive state.
CAN_TX_BUS_PASSIVE_STATE	CAN Transmit is in a Bus Passive state.
CAN_TX_BUS_OFF_STATE	CAN Transmit is in Bus Off state.

5.6.4.5.11.6 CAN_FILTER Enumeration**C**

```
typedef enum {
    CAN_CHANNEL0_EVENT,
    CAN_CHANNEL1_EVENT,
    CAN_CHANNEL2_EVENT,
    CAN_CHANNEL3_EVENT,
    CAN_CHANNEL4_EVENT,
    CAN_CHANNEL5_EVENT,
    CAN_CHANNEL6_EVENT,
    CAN_CHANNEL7_EVENT,
    CAN_CHANNEL8_EVENT,
    CAN_CHANNEL9_EVENT,
    CAN_CHANNEL10_EVENT,
    CAN_CHANNEL11_EVENT,
    CAN_CHANNEL12_EVENT,
    CAN_CHANNEL13_EVENT,
    CAN_CHANNEL14_EVENT,
    CAN_CHANNEL15_EVENT,
    CAN_CHANNEL16_EVENT,
    CAN_CHANNEL17_EVENT,
    CAN_CHANNEL18_EVENT,
    CAN_CHANNEL19_EVENT,
    CAN_CHANNEL20_EVENT,
    CAN_CHANNEL21_EVENT,
    CAN_CHANNEL22_EVENT,
    CAN_CHANNEL23_EVENT,
    CAN_CHANNEL24_EVENT,
    CAN_CHANNEL25_EVENT,
    CAN_CHANNEL26_EVENT,
    CAN_CHANNEL27_EVENT,
    CAN_CHANNEL28_EVENT,
    CAN_CHANNEL29_EVENT,
    CAN_CHANNEL30_EVENT,
    CAN_CHANNEL31_EVENT,
    CAN_NO_EVENT,
    CAN_ERROR_EVENT,
    CAN_WAKEUP_EVENT,
    CAN_RX_CHANNEL_OVERFLOW_EVENT,
    CAN_ADDRESS_ERROR_EVENT,
    CAN_BUS_BANDWIDTH_ERROR,
    CAN_TIMESTAMP_TIMER_EVENT,
    CAN_MODE_CHANGE_EVENT,
    CAN_FILTER0,
    CAN_FILTER1,
    CAN_FILTER2,
    CAN_FILTER3,
    CAN_FILTER4,
    CAN_FILTER5,
    CAN_FILTER6,
    CAN_FILTER7,
```

```

CAN_FILTER8,
CAN_FILTER9,
CAN_FILTER10,
CAN_FILTER11,
CAN_FILTER12,
CAN_FILTER13,
CAN_FILTER14,
CAN_FILTER15,
CAN_FILTER16,
CAN_FILTER17,
CAN_FILTER18,
CAN_FILTER19,
CAN_FILTER20,
CAN_FILTER21,
CAN_FILTER22,
CAN_FILTER23,
CAN_FILTER24,
CAN_FILTER25,
CAN_FILTER26,
CAN_FILTER27,
CAN_FILTER28,
CAN_FILTER29,
CAN_FILTER30,
CAN_FILTER31
} CAN_FILTER;

```

Description

CAN Event Code

This enumeration identifies all of the event codes returned by the [PLIB_CAN_PendingEventsGet](#) function.

Members

Members	Description
CAN_CHANNEL0_EVENT	An event on Channel 0 is active
CAN_CHANNEL1_EVENT	An event on Channel 1 is active
CAN_CHANNEL2_EVENT	An event on Channel 2 is active
CAN_CHANNEL3_EVENT	An event on Channel 3 is active
CAN_CHANNEL4_EVENT	An event on Channel 4 is active
CAN_CHANNEL5_EVENT	An event on Channel 5 is active
CAN_CHANNEL6_EVENT	An event on Channel 6 is active
CAN_CHANNEL7_EVENT	An event on Channel 7 is active
CAN_CHANNEL8_EVENT	An event on Channel 8 is active
CAN_CHANNEL9_EVENT	An event on Channel 9 is active
CAN_CHANNEL10_EVENT	An event on Channel 10 is active
CAN_CHANNEL11_EVENT	An event on Channel 11 is active
CAN_CHANNEL12_EVENT	An event on Channel 12 is active
CAN_CHANNEL13_EVENT	An event on Channel 13 is active
CAN_CHANNEL14_EVENT	An event on Channel 14 is active
CAN_CHANNEL15_EVENT	An event on Channel 15 is active
CAN_CHANNEL16_EVENT	An event on Channel 16 is active
CAN_CHANNEL17_EVENT	An event on Channel 17 is active
CAN_CHANNEL18_EVENT	An event on Channel 18 is active
CAN_CHANNEL19_EVENT	An event on Channel 19 is active
CAN_CHANNEL20_EVENT	An event on Channel 20 is active
CAN_CHANNEL21_EVENT	An event on Channel 21 is active

CAN_CHANNEL22_EVENT	An event on Channel 22 is active
CAN_CHANNEL23_EVENT	An event on Channel 23 is active
CAN_CHANNEL24_EVENT	An event on Channel 24 is active
CAN_CHANNEL25_EVENT	An event on Channel 25 is active
CAN_CHANNEL26_EVENT	An event on Channel 26 is active
CAN_CHANNEL27_EVENT	An event on Channel 27 is active
CAN_CHANNEL28_EVENT	An event on Channel 28 is active
CAN_CHANNEL29_EVENT	An event on Channel 29 is active
CAN_CHANNEL30_EVENT	An event on Channel 30 is active
CAN_CHANNEL31_EVENT	An event on Channel 31 is active
CAN_NO_EVENT	No Event is active
CAN_ERROR_EVENT	CAN Bus Error Event is active
CAN_WAKEUP_EVENT	CAN Bus Wakeup Event is active
CAN_RX_CHANNEL_OVERFLOW_EVENT	CAN Receive Channel Overflow Event is active
CAN_ADDRESS_ERROR_EVENT	CAN Address Error Event is active
CAN_BUS_BANDWIDTH_ERROR	CAN Bus Bandwidth Error is active
CAN_TIMESTAMP_TIMER_EVENT	CAN Timestamp Timer Overflow event is active
CAN_MODE_CHANGE_EVENT	CAN Module Mode Change is active

5.6.4.5.11.7 CAN_FILTER_MASK Enumeration

C

```
typedef enum {
    CAN_FILTER_MASK0,
    CAN_FILTER_MASK1,
    CAN_FILTER_MASK2,
    CAN_FILTER_MASK3,
    CAN_NUMBER_OF_FILTER_MASKS
} CAN_FILTER_MASK;
```

Description

CAN Filter Masks

This enumeration identifies the available filters masks in each CAN module.

Members

Members	Description
CAN_FILTER_MASK0	CAN Filter Mask 0
CAN_FILTER_MASK1	CAN Filter Mask 1
CAN_FILTER_MASK2	CAN Filter Mask 2
CAN_FILTER_MASK3	CAN Filter Mask 3
CAN_NUMBER_OF_FILTER_MASKS	Total number of filter masks in the module

5.6.4.5.11.8 CAN_FILTER_MASK_TYPE Enumeration

C

```
typedef enum {
    CAN_FILTER_MASK_IDE_TYPE,
    CAN_FILTER_MASK_ANY_TYPE
} CAN_FILTER_MASK_TYPE;
```

Description

CAN Filter Mask Type

This enumeration specifies the filter mask type. The filter mask can either process messages with any type of ID (extended or standard) or can only process by ID specified in the filter configuration. For example, if the filter associated with the mask only accepts EID type messages and if the mask is configured to process by ID type, then SID messages will not be accepted. If the mask is configured to process any type of message, both SID and EID messages will be accepted on a filter match.

Members

Members	Description
CAN_FILTER_MASK_IDE_TYPE	Mask processes only Filter type of message
CAN_FILTER_MASK_ANY_TYPE	Mask processes any type (SID or EID) of message

5.6.4.5.11.9 CAN_FUNCIONAL_MODES Enumeration**C**

```
typedef enum {
    CAN_LEGACY_MODE0,
    CAN_ENHANCED_LEGACY_MODE1,
    CAN_ENHANCED_FIFO_MODE2
} CAN_FUNCIONAL_MODES;
```

Description

CAN Functional Modes

This enumerates all operating modes of the CAN module. The application should set the desired mode using the [PLIB_CAN_FunctionalModeSelect](#) function, and should then use the [PLIB_CAN_FunctionalModeGet](#) function to check if the CAN module has entered the requested mode.

Members

Members	Description
CAN_LEGACY_MODE0	CAN Mode 0, Legacy Mode. This mode enabled minimum features. Hides features of Enhanced CAN. This is the default mode of operation on all Reset conditions
CAN_ENHANCED_LEGACY_MODE1	CAN Mode 1, Enhanced Legacy Mode. Enables ECAN features. Enables DeviceNet filtering.
CAN_ENHANCED_FIFO_MODE2	CAN Mode 2, Enhanced FIFO Mode. In this mode, two or more receive buffers are used to form the receive FIFO buffer.

5.6.4.5.11.10 CAN_ID_TYPE Enumeration**C**

```
typedef enum {
    CAN_EID,
    CAN_SID
} CAN_ID_TYPE;
```

Description

CAN ID Type

This enumeration specifies the two CAN ID types: Standard and Extended. The Standard Type ID is 11 bits long and the Extended ID is 29 bits long. This enumeration then specifies the type of the ID specified while configuring filters and filter masks.

Members

Members	Description
CAN_EID	CAN Extended ID
CAN_SID	CAN Standard ID

5.6.4.5.11.11 CAN_MODULE_EVENT Enumeration**C**

```
typedef enum {
    CAN_TX_EVENT,
    CAN_RX_EVENT,
    CAN_TIMESTAMP_TIMER_OVERFLOW_EVENT,
    CAN_OPERATION_MODE_CHANGE_EVENT,
    CAN_RX_OVERFLOW_EVENT,
    CAN_SYSTEM_ERROR_EVENT,
    CAN_BUS_ERROR_EVENT,
    CAN_BUS_ACTIVITY_WAKEUP_EVENT,
    CAN_INVALID_RX_MESSAGE_EVENT
} CAN_MODULE_EVENT;
```

Description

CAN Module Events

This enumeration identifies all of the CAN module events. A combination of listed events can be logically ORed to enable or disable module level events. Similarly, a combination of events can be checked if active.

This enumeration is used with the [PLIB_CAN_ModuleEventEnable](#) function and [PLIB_CAN_ModuleEventGet](#) function.

Members

Members	Description
CAN_TX_EVENT	Transmit channel event. This event will occur if any of the Transmit Channel events are active.
CAN_RX_EVENT	Receive channel event. This event will occur if any of the Receive Channel events are active.
CAN_TIMESTAMP_TIMER_OVERFLOW_EVENT	CAN Timer Stamp Timer Overflow event occurs. This event occurs when the Timestamp Timer has overflowed.
CAN_OPERATION_MODE_CHANGE_EVENT	CAN Operation Mode Change Event. This event occurs when the CAN module has changed it's operating mode successfully.
CAN_RX_OVERFLOW_EVENT	CAN Receive Channel Overflow Event. This event occurs when any of the Receive Channel has overflowed.
CAN_SYSTEM_ERROR_EVENT	CAN System Error Event. This event occurs when CAN module tries to access an invalid Device RAM location.
CAN_BUS_ERROR_EVENT	CAN Bus Error Event. This event occurs when the CAN module cannot access the system bus.
CAN_BUS_ACTIVITY_WAKEUP_EVENT	CAN Bus Activity Wakeup. This event occurs when the device is in Sleep mode and bus activity is detected on the CAN bus.
CAN_INVALID_RX_MESSAGE_EVENT	CAN Bus Invalid Receive Message Event. This event occurs when the CAN module receives an Invalid message.

5.6.4.5.11.12 CAN_MODULE_ID Enumeration**C**

```
typedef enum {
    CAN_ID_1,
    CAN_ID_2,
```

```

    CAN_ID_3,
    CAN_NUMBER_OF_MODULES
} CAN_MODULE_ID;

```

Description

Enumeration: CAN_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers.

Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.

Members

Members	Description
CAN_NUMBER_OF_MODULES	The total number of modules available.

5.6.4.5.11.13 CAN_MSG_EID Structure

C

```

typedef struct {
    unsigned data_length_code : 4;
    unsigned reserved0 : 1;
    unsigned unimplemented1 : 3;
    unsigned reserved1 : 1;
    unsigned remote_request : 1;
    unsigned eid : 18;
    unsigned ide : 1;
    unsigned sub_remote_req : 1;
    unsigned unimplemented2 : 2;
} CAN_MSG_EID;

```

Description

CAN Message EID Word

This data structure represents the EID section of the CAN transmit and receive message. The data structure is an element of the [CAN_TX_MSG_BUFFER](#) and [CAN_RX_MSG_BUFFER](#) data structure.

Members

Members	Description
unsigned data_length_code : 4;	Data Length Control. Specifies the size of the data payload section of the CAN packet. Valid values are 0x0 - 0x8
unsigned reserved0 : 1;	Reserved bit. Should be always 0.
unsigned unimplemented1 : 3;	Unimplemented bit. Should be always 0.
unsigned reserved1 : 1;	Reserved bit. Should be always 0.
unsigned remote_request : 1;	Remote Transmit Request bit. Should be set for RTR messages, clear otherwise.
unsigned eid : 18;	CAN Transmit and Receive Extended ID field. Valid values are in range 0x0 - 0x3FFFF
unsigned ide : 1;	Identifier bit. If 0 means that message is SID. If 1 means that message is EID type.
unsigned sub_remote_req : 1;	Substitute Remote request bit. This bit should always be clear for an EID message. It is ignored for an SID message.
unsigned unimplemented2 : 2;	Unimplemented bit. Should be always 0.

5.6.4.5.11.14 CAN_OPERATION_MODES Enumeration

C

```

typedef enum {

```

```

    CAN_LISTEN_ALL_MESSAGES_MODE,
    CAN_CONFIGURATION_MODE,
    CAN_LISTEN_ONLY_MODE,
    CAN_LOOPBACK_MODE,
    CAN_DISABLE_MODE,
    CAN_NORMAL_MODE
} CAN_OPERATION_MODES;

```

Description

CAN Operation Modes

This enumerates all operating modes of the CAN module. The application should set the desired mode using the [PLIB_CAN_OperationModeSelect](#) function, and should then use the [PLIB_CAN_OperationModeGet](#) function to check if the CAN module has entered the requested mode.

Members

Members	Description
CAN_LISTEN_ALL_MESSAGES_MODE	CAN Listen All Message Mode. The CAN module listens to all messages, regardless of errors
CAN_CONFIGURATION_MODE	CAN Configuration Mode. Various CAN module settings can be configured in this mode
CAN_LISTEN_ONLY_MODE	CAN Listen Only Mode. In this mode, the CAN module will not acknowledge signal and will not participate in error signaling. All messages are captured
CAN_LOOPBACK_MODE	CAN Loopback Mode. In this mode, the CAN module Transmit is internally connected to the Receive line. This mode is useful for debugging operation
CAN_DISABLE_MODE	CAN Disable Mode. The CAN module does not transmit or receive messages in this mode
CAN_NORMAL_MODE	CAN Normal Operation Mode. The CAN module transmits and receives messages in this mode

5.6.4.5.11.15 CAN_RECEIVE_CHANNEL Enumeration

C

```

typedef enum {
    CAN_RECEIVE_CHANNEL0,
    CAN_RECEIVE_CHANNEL1
} CAN_RECEIVE_CHANNEL;

```

Description

CAN Receive Channels

This enumerates all CAN module receive channels(channels that are not allowed to be configured for transmit).

Members

Members	Description
CAN_RECEIVE_CHANNEL0	CAN receive channel 0
CAN_RECEIVE_CHANNEL1	CAN receive channel 1

5.6.4.5.11.16 CAN_RECEIVE_MODES Enumeration

C

```

typedef enum {
    CAN_RECEIVE_ALL_MSG_MODE,
    CAN_RECEIVE_VALID_EID_MODE,
    CAN_RECEIVE_VALID_STD_MODE,
    CAN_RECEIVE_VALID_MODE
}

```

```
} CAN_RECEIVE_MODES;
```

Description

CAN Receive Modes

This enumerates all operating modes of the CAN module. The application should set the desired mode using the [PLIB_CAN_ReceiveModeSelect](#) function, and should then use the [PLIB_CAN_ReceiveModeGet](#) function to check if the CAN module has entered the requested mode.

Members

Members	Description
CAN_RECEIVE_ALL_MSG_MODE	Receive all messages(Including with errors). The message filters will not be considered
CAN_RECEIVE_VALID_EID_MODE	Receive only valid messages(with error will be discarded, filter will be considered) with extended identifier. This is applicable only for functional mode 0
CAN_RECEIVE_VALID_STD_MODE	Receive only valid messages(with error will be discarded, filter will be considered) with standard identifier. This is applicable only for functional mode 0
CAN_RECEIVE_VALID_MODE	Receive only valid messages(with error will be discarded, filter will be considered) with extended identifier/standard identifier Based on the configuration

5.6.4.5.11.17 CAN_RX_DATA_MODE Enumeration

C

```
typedef enum {
    CAN_RX_DATA_ONLY,
    CAN_RX_FULL_RECEIVE
} CAN_RX_DATA_MODE;
```

Description

CAN Receive Channel Data Only Mode

This enumeration specifies the status of the CAN receive channel data-only feature. If the feature is enabled, the CAN module will store only the data payload portion of the received CAN message. If the Full Receive mode is specified, the CAN module stores the entire CAN message (ID field plus data payload). The receive channel can be either in Data-Only mode or Full Receive mode.

Members

Members	Description
CAN_RX_DATA_ONLY	CAN Receive Channel Data Only Mode is enabled
CAN_RX_FULL_RECEIVE	CAN Receive Channel Full Receive Mode is enabled

5.6.4.5.11.18 CAN_RX_MSG_BUFFER Union

C

```
typedef union {
    struct {
        CAN_RX_MSG_SID msgSID;
        CAN_MSG_EID msgEID;
        uint8_t data[8];
    }
    uint8_t dataOnlyMsgData[8];
    uint32_t messageWord[4];
} CAN_RX_MSG_BUFFER;
```

Description

CAN Receive Message Buffer

This data structure represents the CAN receive message buffer. Received messages could be either full-receive messages or data-only messages. A full-receive message contains the message header and data payload section. For a full-receive CAN receive channel, the caller should use the msgSID, msgEID and data members. A data-only message contains only an 8-byte data payload. While using this data structure with a data-only type of CAN receive channel, the caller should use the dataOnlyMsgData member of the structure and should read only 8 bytes of data.

Members

Members	Description
CAN_RX_MSG_SID msgSID;	This is SID portion of the CAN Receive message
CAN_MSG_EID msgEID;	This is EID portion of the CAN Receive message
uint8_t data[8];	This is the data payload section of the received message
uint8_t dataOnlyMsgData[8];	This can be used if the message buffer is to be read from a Data-Only type of CAN Receive Channel
uint32_t messageWord[4];	This is CAN Receive message organized as a set of 32 bit words

5.6.4.5.11.19 CAN_RX_MSG_SID Structure

C

```
typedef struct {
    unsigned sid : 11;
    unsigned filter_hit : 5;
    unsigned msg_timestamp : 16;
} CAN_RX_MSG_SID;
```

Description

CAN Receive Message SID Word

This data structure represents the SID section of the CAN receive message. The data structure is an element of the [CAN_RX_MSG_BUFFER](#) data structure.

Members

Members	Description
unsigned sid : 11;	SID of the Received CAN Message
unsigned filter_hit : 5;	Filter which accepted this message
unsigned msg_timestamp : 16;	Time stamp of the received message. This is valid only if the Timestamping is enabled

5.6.4.5.11.20 CAN_TIME_SEGMENT_LENGTH Enumeration

C

```
typedef enum {
    CAN_TIME_SEGMENT_LEN_8TQ,
    CAN_TIME_SEGMENT_LEN_7TQ,
    CAN_TIME_SEGMENT_LEN_6TQ,
    CAN_TIME_SEGMENT_LEN_5TQ,
    CAN_TIME_SEGMENT_LEN_4TQ,
    CAN_TIME_SEGMENT_LEN_3TQ,
    CAN_TIME_SEGMENT_LEN_2TQ,
    CAN_TIME_SEGMENT_LEN_1TQ
} CAN_TIME_SEGMENT_LENGTH;
```

Description

CAN Segment length

This enumeration defines values that can be assigned while defining the number of Time Quanta in a bit.

Members

Members	Description
CAN_TIME_SEGMENT_LEN_8TQ	Segment length is 8 x TQ
CAN_TIME_SEGMENT_LEN_7TQ	Segment length is 7 x TQ
CAN_TIME_SEGMENT_LEN_6TQ	Segment length is 6 x TQ
CAN_TIME_SEGMENT_LEN_5TQ	Segment length is 5 x TQ
CAN_TIME_SEGMENT_LEN_4TQ	Segment length is 4 x TQ
CAN_TIME_SEGMENT_LEN_3TQ	Segment length is 3 x TQ
CAN_TIME_SEGMENT_LEN_2TQ	Segment length is 2 x TQ
CAN_TIME_SEGMENT_LEN_1TQ	Segment length is 1 x TQ

5.6.4.5.11.21 CAN_TX_CHANNEL_STATUS Enumeration**C**

```
typedef enum {
    CAN_TX_CHANNEL_TRANSMITTING,
    CAN_TX_CHANNEL_ERROR,
    CAN_TX_CHANNEL_ARBITRATION_LOST
} CAN_TX_CHANNEL_STATUS;
```

Description

CAN Transmit Channel Condition

This enumeration identifies the possible transmit channel condition. These masks can be logically ANDed with values returned by the [PLIB_CAN_TransmitChannelStatusGet](#) function to check if a condition is active.

Members

Members	Description
CAN_TX_CHANNEL_TRANSMITTING	CAN Transmit Channel is currently Transmitting.
CAN_TX_CHANNEL_ERROR	CAN Transmit Channel Error has occurred.
CAN_TX_CHANNEL_ARBITRATION_LOST	CAN Transmit Channel lost arbitration.

5.6.4.5.11.22 CAN_TX_MSG_BUFFER Union**C**

```
typedef union {
    struct {
        CAN_TX_MSG_SID msgSID;
        CAN_MSG_EID msgEID;
        unsigned char data[8];
    }
    uint32_t messageWord[4];
} CAN_TX_MSG_BUFFER;
```

Description

CAN Transmit Message Buffer

This data structure represents the CAN transmit message buffer. This should be used for writing data to a CAN transmit channel and transmitting data. A pointer to this type of data structure is returned by the [PLIB_CAN_TransmitBufferGet](#) function.

The data structure allows the transmit message buffer to be accessed as fields of a CAN transmit message and also as an array of four 32-bit words. The latter allows for quick initialization of the message buffer.

Members

Members	Description
CAN_TX_MSG_SID msgSID;	This is SID portion of the CAN Transmit message
CAN_MSG_EID msgEID;	This is EID portion of the CAN Transmit message
unsigned char data[8];	This is the data portion of the CAN Transmit message
uint32_t messageWord[4];	This is CAN Transmit message organized as a set of 32 bit words

5.6.4.5.11.23 CAN_TX_MSG_SID Structure**C**

```
typedef struct {
    unsigned sid : 11;
} CAN_TX_MSG_SID;
```

Description

CAN Transmit Message SID Word

This data structure represents the SID section of the CAN transmit message. The data structure is an element of the [CAN_TX_MSG_BUFFER](#) data structure.

Members

Members	Description
unsigned sid : 11;	CAN Transmit Message Standard ID. This value should be between 0x0 - 0x7FF

5.6.4.5.11.24 CAN_TX_RTR Enumeration**C**

```
typedef enum {
    CAN_TX_RTR_ENABLED,
    CAN_TX_RTR_DISABLED
} CAN_TX_RTR;
```

Description

CAN Transmit Channel Remote Transmit Request (RTR)

This enumeration specifies the status of the CAN RTR feature for a CAN transmit channel. The RTR feature allows a node on the CAN Bus to request a transmission from another node on the bus. The responding node in this case should have a RTR enabled Transmit Channel in order to be able to respond to this request.

Members

Members	Description
CAN_TX_RTR_ENABLED	CAN Transmit Channel RTR Feature is enabled
CAN_TX_RTR_DISABLED	CAN Transmit Channel RTR Feature is disabled

5.6.4.5.11.25 CAN_TXCHANNEL_PRIORITY Enumeration**C**

```
typedef enum {
    CAN_LOWEST_PRIORITY,
    CAN_LOW_MEDIUM_PRIORITY,
    CAN_HIGH_MEDIUM_PRIORITY,
    CAN_HIGHEST_PRIORITY
} CAN_TXCHANNEL_PRIORITY;
```

Description

CAN Transmit Channel Priority

This enumeration identifies the available transmit channel priorities. A transmit channel has its own natural priority order, which determines priority when two or more transmit channels are assigned the same priority level. Channel 1 has higher natural priority than channel 0 and channel 2 has a natural priority than channel 1, and so on.

Members

Members	Description
CAN_LOWEST_PRIORITY	CAN lowest priority
CAN_LOW_MEDIUM_PRIORITY	CAN low medium priority
CAN_HIGH_MEDIUM_PRIORITY	CAN high medium priority
CAN_HIGHEST_PRIORITY	CAN highest priority

5.6.4.5.11.26 CAN_RX_DATA_ONLY_SIZE_BYTES Macro

C

```
#define CAN_RX_DATA_ONLY_SIZE_BYTES 8
```

Description

CAN Data-Only Receive Message Buffer

This constant is used as the size of the CAN data-only receive message buffer in bytes.

5.6.4.5.11.27 CAN_TX_RX_MESSAGE_SIZE_BYTES Macro

C

```
#define CAN_TX_RX_MESSAGE_SIZE_BYTES 16
```

Description

CAN Transmit and Receive Message Buffer size

This constant is used as the array size of the CAN transmit and full receive message buffer.

5.6.4.5.12 Feature Existence Routines**5.6.4.5.12.1 PLIB_CAN_ExistsActiveStatus Function**

C

```
bool PLIB_CAN_ExistsActiveStatus(
    CAN_MODULE_ID index
);
```

Description

This function identifies whether the ActiveStatus feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_IsActive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ActiveStatus feature is supported on the device
- false = The ActiveStatus feature is not supported on the device

Remarks

None.

5.6.4.5.12.2 PLIB_CAN_ExistsAllChannelEvents Function**C**

```
bool PLIB_CAN_ExistsAllChannelEvents(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the AllChannelEvents feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_AllChannelEventsGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The AllChannelEvents feature is supported on the device
- false = The AllChannelEvents feature is not supported on the device

Remarks

None.

5.6.4.5.12.3 PLIB_CAN_ExistsAllChannelOverflow Function**C**

```
bool PLIB_CAN_ExistsAllChannelOverflow(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the AllChannelOverflow feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_AllChannelOverflowStatusGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The AllChannelOverflow feature is supported on the device
- false = The AllChannelOverflow feature is not supported on the device

Remarks

None.

5.6.4.5.12.4 PLIB_CAN_ExistsBusActivityWakeup Function**C**

```
bool PLIB_CAN_ExistsBusActivityWakeup(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the BusActivityWakeup feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_BusActivityWakeupEnable](#)
- [PLIB_CAN_BusActivityWakeupDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The BusActivityWakeup feature is supported on the device
- false = The BusActivityWakeup feature is not supported on the device

Remarks

None.

5.6.4.5.12.5 PLIB_CAN_ExistsBusLine3TimesSampling Function**C**

```
bool PLIB_CAN_ExistsBusLine3TimesSampling(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the BusLine3TimesSampling feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_BusLine3TimesSamplingEnable](#)
- [PLIB_CAN_BusLine3TimesSamplingDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The BusLine3TimesSampling feature is supported on the device
- false = The BusLine3TimesSampling feature is not supported on the device

Remarks

None.

5.6.4.5.12.6 PLIB_CAN_ExistsChannelEvent Function**C**

```
bool PLIB_CAN_ExistsChannelEvent(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelEventGet feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_ChannelEventGet](#)
- [PLIB_CAN_ChannelEventClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ChannelEventGet feature is supported on the device
- false = The ChannelEventGet feature is not supported on the device

Remarks

None.

5.6.4.5.12.7 PLIB_CAN_ExistsChannelEventEnable Function**C**

```
bool PLIB_CAN_ExistsChannelEventEnable(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelEventEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_ChannelEventEnable](#)
- [PLIB_CAN_ChannelEventDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ChannelEventEnable feature is supported on the device
- false = The ChannelEventEnable feature is not supported on the device

Remarks

None.

5.6.4.5.12.8 PLIB_CAN_ExistsChannelForReceiveSet Function

C

```
bool PLIB_CAN_ExistsChannelForReceiveSet(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelForReceiveSet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_ChannelForReceiveSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ChannelForReceiveSet feature is supported on the device
- false = The ChannelForReceiveSet feature is not supported on the device

Remarks

None.

5.6.4.5.12.9 PLIB_CAN_ExistsChannelForTransmitSet Function

C

```
bool PLIB_CAN_ExistsChannelForTransmitSet(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelForTransmitSet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_ChannelForTransmitSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ChannelForTransmitSet feature is supported on the device
- false = The ChannelForTransmitSet feature is not supported on the device

Remarks

None.

5.6.4.5.12.10 PLIB_CAN_ExistsChannelReset Function**C**

```
bool PLIB_CAN_ExistsChannelReset(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelReset feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_ChannelReset](#)
- [PLIB_CAN_ChannelResetIsComplete](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ChannelReset feature is supported on the device
- false = The ChannelReset feature is not supported on the device

Remarks

None.

5.6.4.5.12.11 PLIB_CAN_ExistsChannelUpdate Function**C**

```
bool PLIB_CAN_ExistsChannelUpdate(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelUpdate feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_ChannelUpdate](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ChannelUpdate feature is supported on the device
- false = The ChannelUpdate feature is not supported on the device

Remarks

None.

5.6.4.5.12.12 PLIB_CAN_ExistsDeviceNet Function**C**

```
bool PLIB_CAN_ExistsDeviceNet(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the DeviceNet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_DeviceNetConfigure](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The DeviceNet feature is supported on the device
- false = The DeviceNet feature is not supported on the device

Remarks

None.

5.6.4.5.12.13 PLIB_CAN_ExistsEnableControl Function**C**

```
bool PLIB_CAN_ExistsEnableControl(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_Enable](#)

- [PLIB_CAN_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The EnableControl feature is supported on the device
- false = The EnableControl feature is not supported on the device

Remarks

None.

5.6.4.5.12.14 PLIB_CAN_ExistsErrorState Function**C**

```
bool PLIB_CAN_ExistsErrorState(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ErrorStateGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_ErrorStateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ErrorStateGet feature is supported on the device
- false = The ErrorStateGet feature is not supported on the device

Remarks

None.

5.6.4.5.12.15 PLIB_CAN_ExistsFilterConfigure Function**C**

```
bool PLIB_CAN_ExistsFilterConfigure(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the FilterConfigure feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_FilterConfigure](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The FilterConfigure feature is supported on the device
- false = the FilterConfigure feature is not supported on the device

Remarks

None.

5.6.4.5.12.16 PLIB_CAN_ExistsFilterEnable Function**C**

```
bool PLIB_CAN_ExistsFilterEnable(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the FilterEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_FilterEnable](#)
- [PLIB_CAN_FilterDisable](#)
- [PLIB_CAN_FilterIsDisabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The FilterEnable feature is supported on the device
- false = The FilterEnable feature is not supported on the device

Remarks

None.

5.6.4.5.12.17 PLIB_CAN_ExistsFilterMaskConfigure Function**C**

```
bool PLIB_CAN_ExistsFilterMaskConfigure(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the FilterMaskConfigure feature is available on the CAN module. When this function returns true,

this function is supported on the device:

- [PLIB_CAN_FilterMaskConfigure](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The FilterMaskConfigure feature is supported on the device
- false = The FilterMaskConfigure feature is not supported on the device

Remarks

None.

5.6.4.5.12.18 PLIB_CAN_ExistsFilterToChannelLink Function

C

```
bool PLIB_CAN_ExistsFilterToChannelLink(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the FilterToChannelLink feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_FilterToChannelLink](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The FilterToChannelLink feature is supported on the device
- false = The FilterToChannelLink feature is not supported on the device

Remarks

None.

5.6.4.5.12.19 PLIB_CAN_ExistsFunctionalMode Function

C

```
bool PLIB_CAN_ExistsFunctionalMode(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the FunctionalModeSelect feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_FunctionalModeSelect](#)
- [PLIB_CAN_FunctionalModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The FunctionalModeSelect feature is supported on the device
- false = The FunctionalModeSelect feature is not supported on the device

Remarks

None.

5.6.4.5.12.20 PLIB_CAN_ExistsLatestFilterMatchGet Function

C

```
bool PLIB_CAN_ExistsLatestFilterMatchGet(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the LatestFilterMatchGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_LatestFilterMatchGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The LatestFilterMatchGet feature is supported on the device
- false = The LatestFilterMatchGet feature is not supported on the device

Remarks

None.

5.6.4.5.12.21 PLIB_CAN_ExistsMemoryBufferAssign Function

C

```
bool PLIB_CAN_ExistsMemoryBufferAssign(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the MemoryBufferAssign feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_MemoryBufferAssign](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The MemoryBufferAssign feature is supported on the device
- false = The MemoryBufferAssign feature is not supported on the device

Remarks

None.

5.6.4.5.12.22 PLIB_CAN_ExistsMessageIsReceived Function**C**

```
bool PLIB_CAN_ExistsMessageIsReceived(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the MessageIsReceived feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_MessageIsReceived](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The MessageIsReceived feature is supported on the device
- false = The MessageIsReceived feature is not supported on the device

Remarks

None.

5.6.4.5.12.23 PLIB_CAN_ExistsModuleEventClear Function**C**

```
bool PLIB_CAN_ExistsModuleEventClear(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ModuleEvents feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_ModuleEventClear](#)

- [PLIB_CAN_ModuleEventGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ModuleEvents feature is supported on the device
- false = The ModuleEvents feature is not supported on the device

Remarks

None.

5.6.4.5.12.24 PLIB_CAN_ExistsModuleEventEnable Function**C**

```
bool PLIB_CAN_ExistsModuleEventEnable(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ModuleEventEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_ModuleEventEnable](#)
- [PLIB_CAN_ModuleEventDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ModuleEventEnable feature is supported on the device
- false = The ModuleEventEnable feature is not supported on the device

Remarks

None.

5.6.4.5.12.25 PLIB_CAN_ExistsModuleInfo Function**C**

```
bool PLIB_CAN_ExistsModuleInfo(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ModuleInformation feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_TotalChannelsGet](#)
- [PLIB_CAN_TotalFiltersGet](#)
- [PLIB_CAN_TotalMasksGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ModuleInformation feature is supported on the device
- false = The ModuleInformation feature is not supported on the device

Remarks

None.

5.6.4.5.12.26 PLIB_CAN_ExistsOperationModeRead Function**C**

```
bool PLIB_CAN_ExistsOperationModeRead(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the OperationModeRead feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_OperationModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The OperationModeRead feature is supported on the device
- false = The OperationModeRead feature is not supported on the device

Remarks

None.

5.6.4.5.12.27 PLIB_CAN_ExistsOperationModeWrite Function**C**

```
bool PLIB_CAN_ExistsOperationModeWrite(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the OperationModeSet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_OperationModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The OperationModeSet feature is supported on the device
- false = The OperationModeSet feature is not supported on the device

Remarks

None.

5.6.4.5.12.28 PLIB_CAN_ExistsPendingEventsGet Function**C**

```
bool PLIB_CAN_ExistsPendingEventsGet(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the PendingEventsGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_PendingEventsGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PendingEventsGet feature is supported on the device
- false = The PendingEventsGet feature is not supported on the device

Remarks

None.

5.6.4.5.12.29 PLIB_CAN_ExistsPendingTransmissionsAbort Function**C**

```
bool PLIB_CAN_ExistsPendingTransmissionsAbort(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the PendingTransmissionsAbort feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_PendingTransmissionsAbort](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PendingTransmissionsAbort feature is supported on the device
- false = The PendingTransmissionsAbort feature is not supported on the device

Remarks

None.

5.6.4.5.12.30 PLIB_CAN_ExistsPhaseSegment1Length Function**C**

```
bool PLIB_CAN_ExistsPhaseSegment1Length(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the PhaseSegment1Length feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_PhaseSegment1LengthSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PhaseSegment1Length feature is supported on the device
- false = The PhaseSegment1Length feature is not supported on the device

Remarks

None.

5.6.4.5.12.31 PLIB_CAN_ExistsPhaseSegment2Length Function**C**

```
bool PLIB_CAN_ExistsPhaseSegment2Length(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the PhaseSegment2Length feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_PhaseSegment2LengthSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PhaseSegment2Length feature is supported on the device
- false = The PhaseSegment2Length feature is not supported on the device

Remarks

None.

5.6.4.5.12.32 PLIB_CAN_ExistsPhaseSegment2LengthFreelyProgrammable Function**C**

```
bool PLIB_CAN_ExistsPhaseSegment2LengthFreelyProgrammable(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the PhaseSegment2LengthFreelyProgrammable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_PhaseSegment2LengthFreelyProgrammableEnable](#)
- [PLIB_CAN_PhaseSegment2LengthFreelyProgrammableDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PhaseSegment2LengthFreelyProgrammable feature is supported on the device
- false = The PhaseSegment2LengthFreelyProgrammable feature is not supported on the device

Remarks

None.

5.6.4.5.12.33 PLIB_CAN_ExistsPropagationTimeSegment Function**C**

```
bool PLIB_CAN_ExistsPropagationTimeSegment(  
    CAN_MODULE_ID index
```

```
);
```

Description

This function identifies whether the PropagationTimeSegment feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_PropagationTimeSegmentSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The PropagationTimeSegment feature is supported on the device
- false = The PropagationTimeSegment feature is not supported on the device

Remarks

None.

5.6.4.5.12.34 PLIB_CAN_ExistsReceiveBufferClear Function

C

```
bool PLIB_CAN_ExistsReceiveBufferClear(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveBufferClear feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_ReceiveBufferClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ReceiveBufferClear feature is supported on the device
- false = The ReceiveBufferClear feature is not supported on the device

Remarks

None.

5.6.4.5.12.35 PLIB_CAN_ExistsReceiveBufferIsEmpty Function

C

```
bool PLIB_CAN_ExistsReceiveBufferIsEmpty(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveBufferIsEmpty feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_ReceiveBufferIsEmpty](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ReceiveBufferIsEmpty feature is supported on the device
- false = The ReceiveBufferIsEmpty feature is not supported on the device

Remarks

None.

5.6.4.5.12.36 PLIB_CAN_ExistsReceivedMessageGet Function**C**

```
bool PLIB_CAN_ExistsReceivedMessageGet(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ReceivedMessageGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_ReceivedMessageGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ReceivedMessageGet feature is supported on the device
- false = The ReceivedMessageGet feature is not supported on the device

Remarks

None.

5.6.4.5.12.37 PLIB_CAN_ExistsReceiveErrorCount Function**C**

```
bool PLIB_CAN_ExistsReceiveErrorCount(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveErrorCount feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_ReceiveErrorCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ReceiveErrorCount feature is supported on the device
- false = The ReceiveErrorCount feature is not supported on the device

Remarks

None.

5.6.4.5.12.38 PLIB_CAN_ExistsReceiveModeSelect Function**C**

```
bool PLIB_CAN_ExistsReceiveModeSelect(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveModeSelect feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_ReceiveModeSelect](#)
- [PLIB_CAN_ReceiveModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ReceiveModeSelect feature is supported on the device
- false = The ReceiveModeSelect feature is not supported on the device

Remarks

None.

5.6.4.5.12.39 PLIB_CAN_ExistsRemoteTransmitReq Function**C**

```
bool PLIB_CAN_ExistsRemoteTransmitReq(  
    CAN_MODULE_ID index
```

```
);
```

Description

This function identifies whether the RemoteTransmitReq feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_RemoteTransmitReqlsReceived](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The RemoteTransmitReq feature is supported on the device
- false = The RemoteTransmitReq feature is not supported on the device

Remarks

None.

5.6.4.5.12.40 PLIB_CAN_ExistsRxFIFOFourLeftNotify Function

C

```
bool PLIB_CAN_ExistsRxFIFOFourLeftNotify(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the RxFIFOFourLeftNotify feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_RxFIFOFourLeftNotify](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The RxFIFOFourLeftNotify feature is supported on the device
- false = The RxFIFOFourLeftNotify feature is not supported on the device

Remarks

None.

5.6.4.5.12.41 PLIB_CAN_ExistsRxFIFOOneLeftNotify Function

C

```
bool PLIB_CAN_ExistsRxFIFOOneLeftNotify(  
    CAN_MODULE_ID index  
);
```


Description

This function identifies whether the RxFIFOOneLeftNotify feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_RxFIFOOneLeftNotify](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The RxFIFOOneLeftNotify feature is supported on the device
- false = The RxFIFOOneLeftNotify feature is not supported on the device

Remarks

None.

5.6.4.5.12.42 PLIB_CAN_ExistsStopInIdle Function**C**

```
bool PLIB_CAN_ExistsStopInIdle(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_StopInIdleEnable](#)
- [PLIB_CAN_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The StopInIdle feature is supported on the device
- false = The StopInIdle feature is not supported on the device

Remarks

None.

5.6.4.5.12.43 PLIB_CAN_ExistsSyncJumpWidthSet Function**C**

```
bool PLIB_CAN_ExistsSyncJumpWidthSet(  
    CAN_MODULE_ID index
```

```
);
```

Description

This function identifies whether the SyncJumpWidth feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_SyncJumpWidthSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The SyncJumpWidth feature is supported on the device
- false = The SyncJumpWidth feature is not supported on the device

Remarks

None.

5.6.4.5.12.44 PLIB_CAN_ExistsTimeStampEnable Function

C

```
bool PLIB_CAN_ExistsTimeStampEnable(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the TimeStampEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_TimeStampEnable](#)
- [PLIB_CAN_TimeStampDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The TimeStampEnable feature is supported on the device
- false = The TimeStampEnable feature is not supported on the device

Remarks

None.

5.6.4.5.12.45 PLIB_CAN_ExistsTimeStampPrescalar Function

C

```
bool PLIB_CAN_ExistsTimeStampPrescalar(  

```

```
    CAN_MODULE_ID index
);
```

Description

This function identifies whether the TimeStampPrescalar feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_TimeStampPrescalarSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The TimeStampPrescalar feature is supported on the device
- false = The TimeStampPrescalar feature is not supported on the device

Remarks

None.

5.6.4.5.12.46 PLIB_CAN_ExistsTimeStampValue Function

C

```
bool PLIB_CAN_ExistsTimeStampValue(
    CAN_MODULE_ID index
);
```

Description

This function identifies whether the TimeStampValue feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- [PLIB_CAN_TimeStampValueSet](#)
- [PLIB_CAN_TimeStampValueGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The TimeStampValue feature is supported on the device
- false = The TimeStampValue feature is not supported on the device

Remarks

None.

5.6.4.5.12.47 PLIB_CAN_ExistsTransmissionIsAborted Function

C

```
bool PLIB_CAN_ExistsTransmissionIsAborted(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the TransmissionAbortStatus feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_TransmissionIsAborted](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The TransmissionAbortStatus feature is supported on the device
- false = The TransmissionAbortStatus feature is not supported on the device

Remarks

None.

5.6.4.5.12.48 PLIB_CAN_ExistsTransmitBufferGet Function

C

```
bool PLIB_CAN_ExistsTransmitBufferGet(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitBufferGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_TransmitBufferGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The TransmitBufferGet feature is supported on the device
- false = The TransmitBufferGet feature is not supported on the device

Remarks

None.

5.6.4.5.12.49 PLIB_CAN_ExistsTransmitChannelFlush Function

C

```
bool PLIB_CAN_ExistsTransmitChannelFlush(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitChannelFlush feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_TransmitChannelFlush](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The TransmitChannelFlush feature is supported on the device
- false = The TransmitChannelFlush feature is not supported on the device

Remarks

None.

5.6.4.5.12.50 PLIB_CAN_ExistsTransmitChannelStatus Function

C

```
bool PLIB_CAN_ExistsTransmitChannelStatus(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitChannelStatus feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_TransmitChannelStatusGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The TransmitChannelStatus feature is supported on the device
- false = The TransmitChannelStatus feature is not supported on the device

Remarks

None.

5.6.4.5.12.51 PLIB_CAN_ExistsTransmitErrorCountGet Function

C

```
bool PLIB_CAN_ExistsTransmitErrorCountGet(  
    CAN_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitErrorCountGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- [PLIB_CAN_TransmitErrorCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The TransmitErrorCountGet feature is supported on the device
- false = The TransmitErrorCountGet feature is not supported on the device

Remarks

None.

5.6.4.6 Files

Files

Name	Description
plib_can.h	This file contains the interface definition for the CAN Peripheral Library.

Description

5.6.4.6.1 plib_can.h

CAN Peripheral Library Interface Header

This library provides a low-level abstraction of the Controller Area Network (CAN) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

Functions

	Name	Description
≡	PLIB_CAN_AllChannelEventsGet	Returns a value representing the event status of all CAN channels.
≡	PLIB_CAN_AllChannelOverflowStatusGet	Returns a value representing the receive overflow event status of all CAN channels.

	PLIB_CAN_BusActivityWakeupDisable	Disables the wake-up on bus activity receive line filter.
	PLIB_CAN_BusActivityWakeupEnable	Enables the wake-up on bus activity receive line filter.
	PLIB_CAN_BusLine3TimesSamplingDisable	Disables the bus line three times sampling.
	PLIB_CAN_BusLine3TimesSamplingEnable	Enables the bus line three times sampling.
	PLIB_CAN_ChannelEventClear	Clears CAN channel events.
	PLIB_CAN_ChannelEventDisable	Enables channel level events.
	PLIB_CAN_ChannelEventEnable	Enables channel level events.
	PLIB_CAN_ChannelEventGet	Returns a value representing the event status of a CAN channel.
	PLIB_CAN_ChannelForReceiveSet	Configures a CAN channel for receive operation.
	PLIB_CAN_ChannelForTransmitSet	Configures a CAN channel for transmission.
	PLIB_CAN_ChannelReset	Resets a CAN channel.
	PLIB_CAN_ChannelResetIsComplete	Returns 'true' if the specified channel reset is complete.
	PLIB_CAN_ChannelUpdate	Updates the CAN Channel internal pointers.
	PLIB_CAN_DeviceNetConfigure	Configures the CAN module DeviceNet(TM) filter feature.
	PLIB_CAN_Disable	Disables the specified CAN module.
	PLIB_CAN_Enable	Enables the specified CAN module.
	PLIB_CAN_ErrorStateGet	Returns the CAN error status word.
	PLIB_CAN_ExistsActiveStatus	Identifies whether the ActiveStatus feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelEvents	Identifies whether the AllChannelEvents feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelOverflow	Identifies whether the AllChannelOverflow feature exists on the CAN module.
	PLIB_CAN_ExistsBusActivityWakeup	Identifies whether the BusActivityWakeup feature exists on the CAN module.
	PLIB_CAN_ExistsBusLine3TimesSampling	Identifies whether the BusLine3TimesSampling feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEvent	Identifies whether the ChannelEventGet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEventEnable	Identifies whether the ChannelEventEnable feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForReceiveSet	Identifies whether the ChannelForReceiveSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForTransmitSet	Identifies whether the ChannelForTransmitSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelReset	Identifies whether the ChannelReset feature exists on the CAN module.
	PLIB_CAN_ExistsChannelUpdate	Identifies whether the ChannelUpdate feature exists on the CAN module.
	PLIB_CAN_ExistsDeviceNet	Identifies whether the DeviceNet feature exists on the CAN module.
	PLIB_CAN_ExistsEnableControl	Identifies whether the EnableControl feature exists on the CAN module.
	PLIB_CAN_ExistsErrorState	Identifies whether the ErrorStateGet feature exists on the CAN module.

	<u>PLIB_CAN_ExistsFilterConfigure</u>	Identifies whether the FilterConfigure feature exists on the CAN module.
	<u>PLIB_CAN_ExistsFilterEnable</u>	Identifies whether the FilterEnable feature exists on the CAN module.
	<u>PLIB_CAN_ExistsFilterMaskConfigure</u>	Identifies whether the FilterMaskConfigure feature exists on the CAN module.
	<u>PLIB_CAN_ExistsFilterToChannelLink</u>	Identifies whether the FilterToChannelLink feature exists on the CAN module.
	<u>PLIB_CAN_ExistsFunctionalMode</u>	Identifies whether the FunctionalModeSelect feature exists on the CAN module.
	<u>PLIB_CAN_ExistsLatestFilterMatchGet</u>	Identifies whether the LatestFilterMatchGet feature exists on the CAN module.
	<u>PLIB_CAN_ExistsMemoryBufferAssign</u>	Identifies whether the MemoryBufferAssign feature exists on the CAN module.
	<u>PLIB_CAN_ExistsMessagesReceived</u>	Identifies whether the MessagesReceived feature exists on the CAN module.
	<u>PLIB_CAN_ExistsModuleEventClear</u>	Identifies whether the ModuleEvents feature exists on the CAN module.
	<u>PLIB_CAN_ExistsModuleEventEnable</u>	Identifies whether the ModuleEventEnable feature exists on the CAN module.
	<u>PLIB_CAN_ExistsModuleInfo</u>	Identifies whether the ModuleInformation feature exists on the CAN module.
	<u>PLIB_CAN_ExistsOperationModeRead</u>	Identifies whether the OperationModeRead feature exists on the CAN module.
	<u>PLIB_CAN_ExistsOperationModeWrite</u>	Identifies whether the OperationModeSet feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPendingEventsGet</u>	Identifies whether the PendingEventsGet feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPendingTransmissionsAbort</u>	Identifies whether the PendingTransmissionsAbort feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPhaseSegment1Length</u>	Identifies whether the PhaseSegment1Length feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPhaseSegment2Length</u>	Identifies whether the PhaseSegment2Length feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPhaseSegment2LengthFreelyProgrammable</u>	Identifies whether the PhaseSegment2LengthFreelyProgrammable feature exists on the CAN module.
	<u>PLIB_CAN_ExistsPropagationTimeSegment</u>	Identifies whether the PropagationTimeSegment feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceiveBufferClear</u>	Identifies whether the ReceiveBufferClear feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceiveBufferIsEmpty</u>	Identifies whether the ReceiveBufferIsEmpty feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceivedMessageGet</u>	Identifies whether the ReceivedMessageGet feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceiveErrorCount</u>	Identifies whether the ReceiveErrorCount feature exists on the CAN module.
	<u>PLIB_CAN_ExistsReceiveModeSelect</u>	Identifies whether the ReceiveModeSelect feature exists on the CAN module.
	<u>PLIB_CAN_ExistsRemoteTransmitReq</u>	Identifies whether the RemoteTransmitReq feature exists on the CAN module.

	PLIB_CAN_ExistsRxFIFOFourLeftNotify	Identifies whether the RxFIFOFourLeftNotify feature exists on the CAN module.
	PLIB_CAN_ExistsRxFIFOOneLeftNotify	Identifies whether the RxFIFOOneLeftNotify feature exists on the CAN module.
	PLIB_CAN_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CAN module.
	PLIB_CAN_ExistsSyncJumpWidthSet	Identifies whether the SyncJumpWidth feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampEnable	Identifies whether the TimeStampEnable feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampPrescaler	Identifies whether the TimeStampPrescaler feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampValue	Identifies whether the TimeStampValue feature exists on the CAN module.
	PLIB_CAN_ExistsTransmissionIsAborted	Identifies whether the TransmissionAbortStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitBufferGet	Identifies whether the TransmitBufferGet feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelFlush	Identifies whether the TransmitChannelFlush feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelStatus	Identifies whether the TransmitChannelStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitErrorCountGet	Identifies whether the TransmitErrorCountGet feature exists on the CAN module.
	PLIB_CAN_FilterConfigure	Configures a CAN message acceptance filter.
	PLIB_CAN_FilterDisable	Disables a CAN message acceptance filter.
	PLIB_CAN_FilterEnable	Enables a CAN message acceptance filter.
	PLIB_CAN_FilterIsDisabled	Returns 'true' if the specified filter is disabled.
	PLIB_CAN_FilterMaskConfigure	Configures a CAN filter mask.
	PLIB_CAN_FilterToChannelLink	Links a filter to a channel.
	PLIB_CAN_FunctionalModeGet	Obtains the functional mode of the CAN module.
	PLIB_CAN_FunctionalModeSelect	Selects the functional mode of the CAN module.
	PLIB_CAN_IsActive	Returns 'true' if the CAN module is active.
	PLIB_CAN_LatestFilterMatchGet	Returns the index of the filter that accepted the latest message.
	PLIB_CAN_MemoryBufferAssign	Assigns buffer memory to the CAN module.
	PLIB_CAN_MessageIsReceived	Returns whether a message was received and the receive buffer is not empty.
	PLIB_CAN_ModuleEventClear	Clears the CAN module level events.
	PLIB_CAN_ModuleEventDisable	Disables the module level events.
	PLIB_CAN_ModuleEventEnable	Enables the module level events.
	PLIB_CAN_ModuleEventGet	Returns the status of the CAN module events.
	PLIB_CAN_OperationModeGet	Obtains the current CAN operating mode.
	PLIB_CAN_OperationModeSelect	Sets the CAN operating mode.
	PLIB_CAN_PendingEventsGet	Returns a value representing the highest priority active event in the module.
	PLIB_CAN_PendingTransmissionsAbort	Aborts any pending transmit operations.
	PLIB_CAN_PhaseSegment1LengthSet	Selects the CAN phase buffer segment1 length.

	PLIB_CAN_PhaseSegment2LengthFreelyProgrammableDisable	Disables the CAN phase buffer segment2 length freely programmable mode.
	PLIB_CAN_PhaseSegment2LengthFreelyProgrammableEnable	Makes the CAN phase buffer segment2 length freely programmable.
	PLIB_CAN_PhaseSegment2LengthSet	Selects the CAN phase buffer segment2 length.
	PLIB_CAN_PropagationTimeSegmentSet	Selects the propagation time segment.
	PLIB_CAN_ReceiveBufferClear	Clears the received message flag.
	PLIB_CAN_ReceiveBufferIsEmpty	Returns whether the receive buffer of the specified channel is empty.
	PLIB_CAN_ReceivedMessageGet	Returns a pointer to a message to be read from the CAN channel.
	PLIB_CAN_ReceiveErrorCountGet	Returns the CAN receive error count.
	PLIB_CAN_ReceiveModeGet	Obtains the receive mode of the CAN module.
	PLIB_CAN_ReceiveModeSelect	Selects the receive mode of the CAN module.
	PLIB_CAN_RemoteTransmitReqIsReceived	Returns whether the message received is a remote transmit request.
	PLIB_CAN_RxFIFOFourLeftNotify	Configures the module to generate an interrupt when the FIFO has four receive buffers left.
	PLIB_CAN_RxFIFOOneLeftNotify	Configures the module to generate an interrupt when the FIFO has only one receive buffer left.
	PLIB_CAN_StopInIdleDisable	Disables the Stop in Idle feature.
	PLIB_CAN_StopInIdleEnable	Enables the CAN module to stop when the processor enters Idle mode.
	PLIB_CAN_SyncJumpWidthSet	Selects the synchronization jump width.
	PLIB_CAN_TimeStampDisable	Disables the time stamp feature for the CAN module.
	PLIB_CAN_TimeStampEnable	Enables the time stamp feature for the CAN module.
	PLIB_CAN_TimeStampPrescalerSet	Sets the CAN receive message time stamp timer prescaler.
	PLIB_CAN_TimeStampValueGet	Returns the current value of the CAN receive message time stamp timer value.
	PLIB_CAN_TimeStampValueSet	Sets the CAN receive message time stamp timer value.
	PLIB_CAN_TotalChannelsGet	Returns the total number of CAN channels per CAN module.
	PLIB_CAN_TotalFiltersGet	Returns the total number of CAN Filters per CAN module.
	PLIB_CAN_TotalMasksGet	Returns the total number of CAN masks per CAN module.
	PLIB_CAN_TransmissionIsAborted	Returns 'true' if the transmit abort operation is complete.
	PLIB_CAN_TransmitBufferGet	Returns a pointer to an empty transmit buffer.
	PLIB_CAN_TransmitChannelFlush	Causes all messages in the specified transmit channel to be transmitted.
	PLIB_CAN_TransmitChannelStatusGet	Returns the condition of the transmit channel.
	PLIB_CAN_TransmitErrorCountGet	Returns the CAN transmit error count

Macros

	Name	Description
	CAN_RX_DATA_ONLY_SIZE_BYTES	Used as the size of the CAN data-only receive message buffer in bytes.
	CAN_TX_RX_MESSAGE_SIZE_BYTES	Used as the array size of the CAN transmit and full receive message buffer.

Structures

	Name	Description
	CAN_MSG_EID	Defines the structure of the EID word section of the transmit and receive message.
	CAN_RX_MSG_SID	Defines the structure of the SID word section of the receive message.
	CAN_TX_MSG_SID	Defines the structure of the SID word section of the transmit message.

Unions

	Name	Description
	CAN_RX_MSG_BUFFER	Defines the structure of the CAN receive message buffer
	CAN_TX_MSG_BUFFER	Defines the structure of the CAN transmit message buffer.

File Name

plib_can.h

Company

Microchip Technology Inc.

5.6.5 Comparator Peripheral Library

5.6.5.1 Introduction

Comparator Peripheral Library

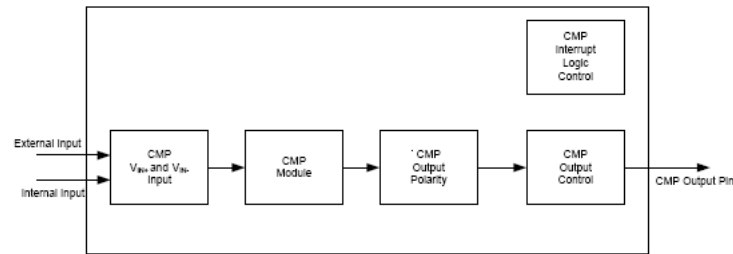
for

Microchip Microcontrollers

The Comparator module is comprised of several identical analog comparator blocks, each complete with its own supporting input selectors and output logic. Each Comparator can be configured in a variety of ways, independent of the other comparators. The input voltage to the module can be either fed externally or it can be configured to use the internal voltage.

Description

This library provides a low-level abstraction of the Comparator module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby abstracting hardware differences from one microcontroller variant to another facilitating future migration.



5.6.5.2 Release Notes

MPLAB Harmony Version: v0.70b **Comparator Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.5.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.5.4 Using the Library

This topic describes the basic architecture of the Comparator Peripheral Library and provides information and examples on how to use it.

Description

Using the Library

Interface Header File: [plib_cmp.h](#)

The interface to the Comparator Peripheral Library is defined in the "[plib_cmp.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the Comparator Peripheral Library must include "peripheral.h".

Library File:

The Comparator Peripheral Library is part of the processor-specific peripheral library installed with the compiler. This library is automatically available (in the default search path) for any project built using the Microchip compilers.

Peripheral Module IDs

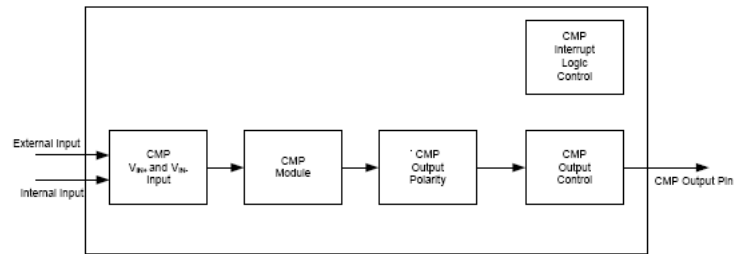
Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. The first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). For the Comparator Peripheral Library, the module instance IDs are defined by the Comparator module enumeration.

5.6.5.4.1 Hardware Abstraction Model

This library provides a low-level abstraction of the Comparator (CMP) module on Microchip PIC microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model Diagram



Hardware Abstraction Model Description

The Comparator Peripheral Library provides interface routines to interact with the blocks shown in the diagram above.

The Comparator VIN+ and VIN- block selects the Comparator non-inverting and inverting inputs, which can be either external or internal inputs.

The Comparator module block compares the voltage level on the inputs and produce an output. If the input VIN+ > VIN-, the output will be high. The output will be low when VIN+ < VIN-.

The Comparator Output Polarity block selects the Comparator output polarity.

The Comparator Output Control block enables/disables the Comparator output pin.

The Comparator Interrupt Logic Control block decide the comparator interrupt generation.The interrupt can happen at Low-to-high, High-to-Low, and for both transitions of the Comparator output.

5.6.5.4.2 Library Usage Model

The set of interface routines for the CMP Peripheral Library are:

Library Interface Section	Description

5.6.5.4.3 How the Library Works

The following processes are involved while using a Comparator module:

- Initialization
- Configuring the Comparator interrupt (optional)
- Power-saving modes
- Turning on the Comparator module

Name	Description
Initialization	Initialize the comparator module.
Configuring the Comparator Interrupts	Enable the Comparator interrupt (optional)
Power-Saving Modes	Enable the Comparator power-saving modes (optional)

5.6.5.4.3.1 Initialization

This section describes Comparator module initialization.

Description

The following steps are involved while using a Comparator module.

#	Description	Function Associated
1	Select the Comparator inverting input.	PLIB_CMP_InvertingInputChannelSelect
2	Select the Comparator non-inverting input.	PLIB_CMP_NonInvertingInputChannelSelect
3	Select the Comparator output polarity.	PLIB_CMP_InvertedOutputSelect PLIB_CMP_NonInvertedOutputSelect
4	Select the Comparator output pin.	PLIB_CMP_OutputEnable PLIB_CMP_OutputDisable
5	Turn on the Comparator module.	PLIB_CMP_Enable

Example

```
#define MY_CMP_ID          CMP_ID_1

//Select the comparator inverting input
PLIB_CMP_InvertingInputChannelSelect(MY_CMP_INSTANCE, CMP_INVERTING_INPUT_CHANNEL_B);

//Select the comparator non-inverting input
PLIB_CMP_NonInvertingInputChannelSelect(MY_CMP_INSTANCE, CMP_NON_INVERTING_INPUT_A);

//Select the comparator output polarity
PLIB_CMP_NonInvertedOutputSelect(MY_CMP_INSTANCE);

//Enable the comparator output pin
PLIB_CMP_OutputEnable(MY_CMP_INSTANCE);

//Turn on the comparator module
PLIB_CMP_Enable(MY_CMP_INSTANCE);
```

5.6.5.4.3.2 Configuring the Comparator Interrupts

This section describes the configuration of Comparator module interrupts.

Description

The following steps are involved while configuring a Comparator interrupt.

The comparator interrupt can happen for three events. The user must configure the event-detection logic to report changes to the output state of the Comparator, which can in turn be used for event or interrupt generation. The options include event generation on rising state changes (low-to-high), falling state changes (high-to-low), and all state changes. When a configured event occurs, it is flagged by the event detection flag. The user must clear this flag for the next interrupt generation.

#	Description	Function Associated
1	Comparator event select.	PLIB_CMP_InterruptEventSelect
2	Comparator event flag read/clear	PLIB_CMP_EventHasOccurred PLIB_CMP_EventStatusClear

Example

```
#define MY_CMP_ID          CMP_ID_1

//This code expects the comparator interrupt
//enabled in the interrupt module

//Select the comparator interrupt event
```

```

PLIB_CMP_InterruptEventSelect(MY_CMP_INSTANCE, CMP_LOW_TO_HIGH);

//This part of the code should be in ISR
if(PLIB_CMP_EventHasOccurred(CMP_ID_1))
{
    //Do some operations
    \
    //Clear the event flag
    PLIB_CMP_EventStatusClear(CMP_ID_1);
}

//Clear the interrupt flag in the interrupt module

```

Note: Not all functionality is available on all devices. Please refer to the specific device data sheet for availability.

5.6.5.4.3.3 Power-Saving Modes

This section provides information on the Power-Saving modes for the Comparator module.

Description

Operation in Sleep mode: If the Comparator interrupt is enabled, the device will wake up from Sleep mode on the Comparator interrupt. If the Comparator interrupt is disabled, the comparator will work, but it will not wake up the device from sleep.

Operation in Idle mode: The functions [PLIB_CMP_StopInIdleModeEnable](#) and [PLIB_CMP_StopInIdleModeDisable](#) determine if the Comparator module stops or continues operation in Idle mode. If the function [PLIB_CMP_StopInIdleModeDisable](#) is used, the module will continue operation in the Idle mode. If the Comparator interrupt is enabled, the device will wake up from Idle mode on the Comparator interrupt. If the function [PLIB_CMP_StopInIdleModeEnable](#) is used, the module will work in Idle mode but the interrupt will not generate if it is enabled.

#	Decscription	Function Associated
1	Enable Comparator Stop in Idle mode.	PLIB_CMP_StopInIdleModeEnable
2	Disable Comparator Stop in Idle mode.	PLIB_CMP_StopInIdleModeDisable

Example

```

//Select comparator stop in idle mode
PLIB_CMP_StopInIdleModeEnable(MY_CMP_INSTANCE);

```

Note: Not all functionality is available on all devices. Please refer to the specific device data sheet for availability.

5.6.5.5 Library Interface

Data Types and Constants

	Name	Description
	CMP_CLOCK_DIVIDE	Defines CMP Filter Clock Divide
	CMP_CVREF_BANDGAP_SELECT	Lists the bandgap selection options.
	CMP_CVREF_REFERENCE_SELECT	Lists the reference selection options.
	CMP_CVREF_VALUE	Lists the voltage reference value selection options.
	CMP_CVREF_VOLTAGE_SOURCE	Lists the Voltage source options.
	CMP_FILTER_CLOCK	Defines CMP filter input clock
	CMP_INTERRUPT_EVENT	Defines CMP interrupt events.
	CMP_INVERTING_INPUT	Defines the list of CMP inverting Input
	CMP_MASK_A	Defines CMP Mask A Input.

	CMP_MASK_B	Defines CMP Mask B Input.
	CMP_MASK_C	Defines CMP Mask C Input.
	CMP_MODULE_ID	Identifies the CMP modules supported
	CMP_NON_INVERTING_INPUT	Defines the list of CMP non inverting Input
	CMP_SPEED_POWER	Defines CMP Speed/Power

Comparator AND Gate Configuration Functions

	Name	Description
	PLIB_CMP_AndGateAInputDisable	A input disabled as input to the AND gate
	PLIB_CMP_AndGateAInputEnable	A input enabled as input to the AND gate.
	PLIB_CMP_AndGateBInputDisable	B input disabled as input to the AND gate.
	PLIB_CMP_AndGateBInputEnable	B input enabled as input to the AND gate.
	PLIB_CMP_AndGateCInputEnable	C input enabled as input to the AND gate.
	PLIB_CMP_AndGateInvertedAInputDisable	Inverted A input disabled as input to the AND gate.
	PLIB_CMP_AndGateInvertedAInputEnable	Inverted A input enabled as input to the AND gate.
	PLIB_CMP_AndGateInvertedBInputDisable	Inverted B input disabled as input to the AND gate.
	PLIB_CMP_AndGateInvertedBInputEnable	Inverted B input enabled as input to the AND gate.
	PLIB_CMP_AndGateInvertedCInputDisable	Inverted C input disabled as input to the AND gate.
	PLIB_CMP_AndGateInvertedCInputEnable	Inverted C input enabled as input to the AND gate.
	PLIB_CMP_NegativeAndGateOutputDisable	Negative AND gate output disable.
	PLIB_CMP_NegativeAndGateOutputEnable	Negative AND gate output enable.
	PLIB_CMP_PositiveAndGateOutputDisable	Positive AND gate output disable.
	PLIB_CMP_PositiveAndGateOutputEnable	Positive AND gate output enable.
	PLIB_CMP_AndGateCInputDisable	C input disabled as input to the AND gate.

Comparator Feature Configuration Functions

	Name	Description
	PLIB_CMP_Cmp1OutputHighSelect	Comparator output bit is set to '1'.
	PLIB_CMP_Cmp1OutputLowSelect	Comparator output bit is set to '0'.
	PLIB_CMP_Cmp2OutputHighSelect	Comparator output bit is set to '1'.
	PLIB_CMP_Cmp2OutputLowSelect	Comparator output bit is set to '0'.
	PLIB_CMP_HysteresisDisable	Comparator hysteresis disable.
	PLIB_CMP_HysteresisEnable	Comparator hysteresis enable.
	PLIB_CMP_OutputHighSelect	Comparator output bit is set to '1'.
	PLIB_CMP_OutputLowSelect	Comparator output bit is set to '0'.
	PLIB_CMP_OutputSynchronousWithTimer1Disable	Comparator output is asynchronous.
	PLIB_CMP_OutputSynchronousWithTimer1Enable	Comparator Output Synchronous mode select.
	PLIB_CMP_StopInIdleModeDisable	Disables Stop in Idle mode.
	PLIB_CMP_StopInIdleModeEnable	Enables Stop in Idle mode.

Comparator Filter Clock Configuration Functions

	Name	Description
	PLIB_CMP_FilterClockDivideSelect	Comparator filter clock divide select.
	PLIB_CMP_FilterDisable	Comparator filter disable.
	PLIB_CMP_FilterEnable	Comparator filter enable.
	PLIB_CMP_FilterInputClockSelect	Comparator filter input clock select.

Comparator Mask Select Configuration Functions

	Name	Description
	PLIB_CMP_HighLevelMaskSelect	High-level masking select.
	PLIB_CMP_LowLevelMaskSelect	Low-level masking select.

Comparator Op amp Mode Configuration Functions

	Name	Description
	PLIB_CMP_OpAmpModeDisable	Op amp operation mode disable.
	PLIB_CMP_OpAmpModeEnable	Op amp operation mode enable.
	PLIB_CMP_OpAmpOutputDisable	Op amp output disable.
	PLIB_CMP_OpAmpOutputEnable	Op amp output enable.

Comparator OR Gate Configuration Functions

	Name	Description
	PLIB_CMP_OrGateAInputDisable	OR gate A input disable.
	PLIB_CMP_OrGateAInputEnable	A input connected to OR gate.
	PLIB_CMP_OrGateBInputDisable	OR gate B input disable.
	PLIB_CMP_OrGateBInputEnable	B input connected to OR gate.
	PLIB_CMP_OrGateCInputDisable	OR gate C input disable.
	PLIB_CMP_OrGateCInputEnable	C input connected to OR gate.
	PLIB_CMP_OrGateInvertedAInputDisable	OR gate inverted A input disable.
	PLIB_CMP_OrGateInvertedAInputEnable	Inverted A input connected to OR gate.
	PLIB_CMP_OrGateInvertedBInputDisable	OR gate inverted B input disable.
	PLIB_CMP_OrGateInvertedBInputEnable	Inverted B input connected to OR gate
	PLIB_CMP_OrGateInvertedCInputDisable	OR gate inverted C input disable.
	PLIB_CMP_OrGateInvertedCInputEnable	Inverted C input connected to OR gate

Feature Existence Functions

	Name	Description
	PLIB_CMP_ExistsCVREFBGRRefVoltageRangeSelect	Identifies whether the CVREFBGRRefVoltageRangeSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFEnableControl	Identifies whether the CVREFEnableControl feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFOutputEnableControl	Identifies whether the CVREFOutputEnableControl feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFRefVoltageRangeSelect	Identifies whether the CVREFRefVoltageRangeSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFValueSelect	Identifies whether the CVREFValueSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFVoltageRangeSelect	Identifies whether the CVREFVoltageRangeSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFWideRangeControl	Identifies whether the CVREFWideRangeControl feature exists on the Comparator module.
	PLIB_CMP_ExistsEnableControl	Identifies whether the ComparatorEnableControl feature exists on the Comparator module.
	PLIB_CMP_ExistsInterruptEventSelect	Identifies whether the InterruptEventSelect feature exists on the Comparator module.

	PLIB_CMP_ExistsInvertingInputSelect	Identifies whether the InvertingInputSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsInvertOutputControl	Identifies whether the InvertOutputSelectControl feature exists on the Comparator module.
	PLIB_CMP_ExistsNonInvertingInputSelect	Identifies whether the NonInvertingInputSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsOutputEnableControl	Identifies whether the ComparatorOutputEnableControl feature exists on the Comparator module.
	PLIB_CMP_ExistsOutputLevelControl	Identifies whether the OutputLevelSelectControl feature exists on the Comparator module.
	PLIB_CMP_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Comparator module.

General Configuration Functions

	Name	Description
	PLIB_CMP_CVREF_BandGapReferenceSourceSelect	Selects the band gap reference voltage source.
	PLIB_CMP_CVREF_Enable	Enables the voltage reference of the Comparator module.
	PLIB_CMP_CVREF_Output2Disable	Disables the Vsource/2 voltage output.
	PLIB_CMP_CVREF_Output2Enable	Enables the Vsource/2 voltage output.
	PLIB_CMP_CVREF_OutputDisable	Disables the output voltage.
	PLIB_CMP_CVREF_OutputEnable	Enables the voltage output.
	PLIB_CMP_CVREF_ReferenceVoltageSelect	Selects the voltage reference value.
	PLIB_CMP_CVREF_SourceNegativeInputSelect	Configures the Comparator module to use the selected input as a negative reference.
	PLIB_CMP_CVREF_SourceVoltageSelect	Connects the Comparator module to the selected voltage source.
	PLIB_CMP_CVREF_ValueGet	Returns the voltage setting value.
	PLIB_CMP_CVREF_ValueSelect	Selects the voltage reference value.
	PLIB_CMP_CVREF_WideRangeDisable	Disables the wide range.
	PLIB_CMP_CVREF_WideRangeEnable	Enables the wide range.
	PLIB_CMP_CVREF_WideRangeIsEnabled	Returns whether the wide range is selected for the reference voltage.
	PLIB_CMP_Disable	Disables the Comparator module.
	PLIB_CMP_Enable	Enables the Comparator module.
	PLIB_CMP_EventHasOccurred	Comparator event status.
	PLIB_CMP_EventStatusClear	Comparator event clear.
	PLIB_CMP_InterruptEventSelect	Comparator interrupt event select.
	PLIB_CMP_InvertedOutputSelect	Comparator output is inverted.
	PLIB_CMP_InvertingInputChannelSelect	Comparator inverting input channel select.
	PLIB_CMP_NonInvertedOutputSelect	Comparator output is non-inverted.
	PLIB_CMP_NonInvertingInputChannelSelect	Comparator input channel select.
	PLIB_CMP_OutputDisable	Disables the Comparator output.
	PLIB_CMP_OutputEnable	Enables the Comparator output.
	PLIB_CMP_OutputStatusGet	Comparator output status.
	PLIB_CMP_CVREF_Disable	Disables the voltage reference of the Comparator module.

Description

This section lists and describes the functions, data types, and constants available in the Comparator Peripheral Library.

5.6.5.5.1 General Configuration Functions

5.6.5.5.1.1 PLIB_CMP_CVREF_BandGapReferenceSourceSelect Function

C

```
void PLIB_CMP_CVREF_BandGapReferenceSourceSelect(  
    CMP_MODULE_ID index,  
    CMP_CVREF_BANDGAP_SELECT select  
);
```

Description

This function selects the band gap reference voltage source from the available options from [CMP_CVREF_BANDGAP_SELECT](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
select	Select a band gap reference source from CMP_CVREF_BANDGAP_SELECT

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFBGRefVoltageRangeSelect` in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_BandGapReferenceSourceSelect ( CMP_CVREF_ID_1,  PLIB_CMP_CVREF_BANDGAP_0_6V );
```

5.6.5.5.1.2 PLIB_CMP_CVREF_Enable Function

C

```
void PLIB_CMP_CVREF_Enable(  
    CMP_MODULE_ID index  
);
```

Description

This function enables the voltage reference of the Comparator module.

Preconditions

The Comparator module should be appropriately configured before being enabled.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_CMP_CVREF_ExistsCVREFEnableControl in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_Enable(CMP_CVREF_ID_1);
```

5.6.5.5.1.3 PLIB_CMP_CVREF_Output2Disable Function**C**

```
void PLIB_CMP_CVREF_Output2Disable(  
    CMP_MODULE_ID index  
);
```

Description

This function disables the Vsource/2 voltage output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_CMP_CVREF_Output2Disable(CMP_CVREF_ID_1);
```

5.6.5.5.1.4 PLIB_CMP_CVREF_Output2Enable Function**C**

```
void PLIB_CMP_CVREF_Output2Enable(  
    CMP_MODULE_ID index  
);
```

Description

This function enables the Vsource/2 voltage output. This is independent of voltage output 1.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_CMP_CVREF_Output2Enable(CMP_CVREF_ID_1);
```

5.6.5.5.1.5 PLIB_CMP_CVREF_OutputDisable Function

C

```
void PLIB_CMP_CVREF_OutputDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function disables the reference voltage output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFOutputEnableControl` in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_VoltageOutputDisable(CMP_CVREF_ID_1);
```

5.6.5.5.1.6 PLIB_CMP_CVREF_OutputEnable Function

C

```
void PLIB_CMP_CVREF_OutputEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function enables the voltage output

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFOutputEnableControl` in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_OutputEnable(CMP_CVREF_ID_1);
```

5.6.5.5.1.7 PLIB_CMP_CVREF_ReferenceVoltageSelect Function

C

```
void PLIB_CMP_CVREF_ReferenceVoltageSelect(  
    CMP_MODULE_ID index,  
    CMP_CVREF_REFERENCE_SELECT reference  
);
```

Description

This function selects the voltage reference value. This value decides how many resistance units will be added and thus decides the output voltage.

Preconditions

Determine the correct value that should be passed.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Select value from CMP_CVREF_VALUE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFRefVoltageRangeSelect` in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_ReferenceVoltageSelect ( CMP_CVREF_ID_1,  CMP_CVREF_VALUE_13 );
```

5.6.5.5.1.8 PLIB_CMP_CVREF_SourceNegativeInputSelect Function

C

```
void PLIB_CMP_CVREF_SourceNegativeInputSelect(  
    CMP_MODULE_ID index,  
    CMP_CVREF_VOLTAGE_SOURCE_NEG_REFERENCE negInput  
);
```

Description

This function configures the Comparator module to use the selected input as a negative reference for the voltage source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured.
negInput	Select the voltage source negative reference from <code>CMP_CVREF_VOLTAGE_SOURCE_NEG_REFERENCE</code>

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability. For such devices, selecting the positive source will automatically select the negative input.

Example

```
PLIB_CMP_CVREF_SourceNegativeInputSelect (CMP_CVREF_ID_1,  
CMP_CVREF_VOLTAGE_SOURCE_NEG_REF_GROUND);
```

5.6.5.5.1.9 PLIB_CMP_CVREF_SourceVoltageSelect Function**C**

```
void PLIB_CMP_CVREF_SourceVoltageSelect(  
    CMP_MODULE_ID index,  
    CMP_CVREF_VOLTAGE_SOURCE source  
);
```

Description

This function connects the Comparator module to the selected voltage source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the voltage source from CMP_CVREF_VOLTAGE_SOURCE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFVoltageRangeSelect` in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_SourceVoltageSelect (CMP_CVREF_ID_1, CMP_CVREF_VOLTAGE_SOURCE_INTERNAL);
```

5.6.5.5.1.10 PLIB_CMP_CVREF_ValueGet Function**C**

```
CMP_CVREF_VALUE PLIB_CMP_CVREF_ValueGet(  
    CMP_MODULE_ID index  
);
```

Description

This function returns the configured reference voltage value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

[CMP_CVREF_VALUE](#) - Reference voltage value

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFValueSelect` in your application to determine whether this feature is available.

Example

```
CMP_CVREF_VALUE value;

value = PLIB_CMP_CVREF_ValueGet ( CMP_CVREF_ID_1 );
```

5.6.5.5.1.11 PLIB_CMP_CVREF_ValueSelect Function

C

```
void PLIB_CMP_CVREF_ValueSelect(
    CMP_MODULE_ID index,
    CMP_CVREF_VALUE value
);
```

Description

This function selects the voltage reference value. This value decides how many resistance units will be added and therefore, decides the output voltage.

Preconditions

Determine the correct value that should be passed.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Select value from CMP_CVREF_VALUE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFValueSelect` in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_ValueSelect ( CMP_CVREF_ID_1,  CMP_CVREF_VALUE_13 );
```

5.6.5.5.1.12 PLIB_CMP_CVREF_WideRangeDisable Function

C

```
void PLIB_CMP_CVREF_WideRangeDisable(
    CMP_MODULE_ID index
);
```

Description

This function disables the wide range for reference voltage. The range of possible voltages will become narrower, and finer voltage options can be achieved in this case.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFWideRangeControl` in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_WideRangeDisable(CMP_CVREF_ID_1);
```

5.6.5.5.1.13 PLIB_CMP_CVREF_WideRangeEnable Function**C**

```
void PLIB_CMP_CVREF_WideRangeEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function enables the wide range for reference voltage. The voltage range starts from zero if the wide range is selected.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFWideRangeControl` in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_WideRangeEnable(CMP_CVREF_ID_1);
```

5.6.5.5.1.14 PLIB_CMP_CVREF_WideRangeIsEnabled Function**C**

```
bool PLIB_CMP_CVREF_WideRangeIsEnabled(  
    CMP_MODULE_ID index  
);
```

Description

This function returns whether the wide range is enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true = The wide range is enabled
- false = The wide range is not enabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFWideRangeControl` in your application to determine whether this feature is available.

Example

```
bool range;  
  
range = PLIB_CMP_CVREF_WideRangeIsEnabled(MY_CMP_CVREF_ID);
```

5.6.5.5.1.15 PLIB_CMP_Disable Function**C**

```
void PLIB_CMP_Disable(  
    CMP_MODULE_ID index  
);
```

Description

This function disables (i.e., turns OFF) the selected Comparator module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
PLIB_CMP_Disable(MY_CMP_INSTANCE);
```

5.6.5.5.1.16 PLIB_CMP_Enable Function**C**

```
void PLIB_CMP_Enable(  
    CMP_MODULE_ID index  
);
```

Description

This function enables (i.e., turns ON) the selected Comparator module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_Enable(MY_CMP_INSTANCE);
```

5.6.5.5.1.17 PLIB_CMP_EventHasOccurred Function**C**

```
bool PLIB_CMP_EventHasOccurred(
    CMP_MODULE_ID index
);
```

Description

This function will return the current event status of the Comparator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true = The event flag is set
- false = The event flag is clear

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// bool cmp_event_status = PLIB_CMP_EventHasOccurred(MY_CMP_INSTANCE);
```

5.6.5.5.1.18 PLIB_CMP_EventStatusClear Function**C**

```
void PLIB_CMP_EventStatusClear(
    CMP_MODULE_ID index
);
```

Description

This function will clear event status bit of the Comparator. Subsequent triggers and interrupts are disabled until the bit is cleared.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_EventStatusClear(MY_CMP_INSTANCE);
```

5.6.5.5.1.19 PLIB_CMP_InterruptEventSelect Function**C**

```
void PLIB_CMP_InterruptEventSelect(  
    CMP_MODULE_ID index,  
    CMP_INTERRUPT_EVENT event  
);
```

Description

This function will select when the Comparator interrupt should occur.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
event	One of the possible values from CMP_INTERRUPT_EVENT

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsInterruptEventSelect](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// CMP_INTERRUPT_EVENT - CMP_LOW_TO_HIGH  
PLIB_CMP_InterruptEventSelect(MY_CMP_INSTANCE, CMP_LOW_TO_HIGH);
```

5.6.5.5.1.20 PLIB_CMP_InvertedOutputSelect Function

C

```
void PLIB_CMP_InvertedOutputSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function will select the inverted comparator output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_ExistsOutputEnableControlExistsInvertOutputControl` in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
PLIB_CMP_InvertedOutputSelect(MY_CMP_INSTANCE, );
```

5.6.5.5.1.21 PLIB_CMP_InvertingInputChannelSelect Function

C

```
void PLIB_CMP_InvertingInputChannelSelect(  
    CMP_MODULE_ID index,  
    CMP_INVERTING_INPUT channel  
);
```

Description

This function will select the inverting input channels for the Comparator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	One of the possible values from <code>CMP_INVERTING_INPUT</code>

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_CMP_ExistsInvertingInputSelect](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_InvertingInputChannelSelect(MY_CMP_INSTANCE, CMP_INVERTING_INPUT_CHANNEL_B);
```

5.6.5.5.1.22 PLIB_CMP_NonInvertedOutputSelect Function

C

```
void PLIB_CMP_NonInvertedOutputSelect(
    CMP_MODULE_ID index
);
```

Description

This function will select the non-inverted comparator output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsOutputEnableControlExistsInvertOutputControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_NonInvertedOutputSelect(MY_CMP_INSTANCE, );
```

5.6.5.5.1.23 PLIB_CMP_NonInvertingInputChannelSelect Function

C

```
void PLIB_CMP_NonInvertingInputChannelSelect(
    CMP_MODULE_ID index,
    CMP_NON_INVERTING_INPUT input
);
```

Description

This function will select the non-inverting input channels for the Comparator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	One of the possible values from CMP_NON_INVERTING_INPUT

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsNonInvertingInputSelect](#) in your application to determine whether this feature is available.s

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_NonInvertingInputChannelSelect(MY_CMP_INSTANCE, CMP_NON_INVERTING_INPUT_A);
```

5.6.5.5.1.24 PLIB_CMP_OutputDisable Function**C**

```
void PLIB_CMP_OutputDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function disables (i.e., turns OFF) the Comparator output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsOutputEnableControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
PLIB_CMP_OutputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.1.25 PLIB_CMP_OutputEnable Function**C**

```
void PLIB_CMP_OutputEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function enables (i.e., turns ON) the Comparator output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsOutputEnableControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.1.26 PLIB_CMP_OutputStatusGet Function**C**

```
bool PLIB_CMP_OutputStatusGet(
    CMP_MODULE_ID index
);
```

Description

This function will return the current status of the Comparator output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true = The status flag is set
- false = The status flag is clear

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// bool cmp_status=PLIB_CMP_OutputStatusGet(MY_CMP_INSTANCE);
```

5.6.5.5.1.27 PLIB_CMP_CVREF_Disable Function**C**

```
void PLIB_CMP_CVREF_Disable(
    CMP_MODULE_ID index
);
```

Description

This function disables the voltage reference of the Comparator module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_CMP_CVREF_ExistsCVREFEnableControl` in your application to determine whether this feature is available.

Example

```
PLIB_CMP_CVREF_Disable (CMP_CVREF_ID_1);
```

5.6.5.5.2 Comparator Feature Configuration Functions

5.6.5.5.2.1 PLIB_CMP_Cmp1OutputHighSelect Function

C

```
void PLIB_CMP_Cmp1OutputHighSelect(  
    CMP_MODULE_ID index  
)
```

Description

This function will set the Comparator output bit to '1'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
PLIB_CMP_Cmp1OutputHighSelect(MY_CMP_INSTANCE, );
```

5.6.5.5.2.2 PLIB_CMP_Cmp1OutputLowSelect Function

C

```
void PLIB_CMP_Cmp1OutputLowSelect(  
    CMP_MODULE_ID index  
)
```

Description

This function will set the Comparator output bit to '0'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
PLIB_CMP_Cmp1OutputLowSelect(MY_CMP_INSTANCE, );
```

5.6.5.5.2.3 PLIB_CMP_Cmp2OutputHighSelect Function**C**

```
void PLIB_CMP_Cmp2OutputHighSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function will set the Comparator output bit to '1'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
PLIB_CMP_Cmp2OutputHighSelect(MY_CMP_INSTANCE, );
```

5.6.5.5.2.4 PLIB_CMP_Cmp2OutputLowSelect Function**C**

```
void PLIB_CMP_Cmp2OutputLowSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function will set the Comparator output bit to '0'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
PLIB_CMP_Cmp2OutputLowSelect(MY_CMP_INSTANCE, );
```

5.6.5.5.2.5 PLIB_CMP_HysteresisDisable Function**C**

```
void PLIB_CMP_HysteresisDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will disable the hysteresis of the Comparator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_HysteresisDisable(MY_CMP_INSTANCE);
```

5.6.5.5.2.6 PLIB_CMP_HysteresisEnable Function**C**

```
void PLIB_CMP_HysteresisEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will enable the hysteresis of the Comparator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_HysteresisEnable(MY_CMP_INSTANCE);
```

5.6.5.5.2.7 PLIB_CMP_OutputHighSelect Function**C**

```
void PLIB_CMP_OutputHighSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function will set the Comparator output bit to '1'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsOutputLevelControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
PLIB_CMP_OutputHighSelect(MY_CMP_INSTANCE, );
```

5.6.5.5.2.8 PLIB_CMP_OutputLowSelect Function**C**

```
void PLIB_CMP_OutputLowSelect(  
    CMP_MODULE_ID index
```

```
);
```

Description

This function will set the Comparator output bit to '0'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsOutputLevelControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputLowSelect(MY_CMP_INSTANCE, );
```

5.6.5.5.2.9 PLIB_CMP_OutputSynchronousWithTimer1Disable Function

C

```
void PLIB_CMP_OutputSynchronousWithTimer1Disable(
    CMP_MODULE_ID index
);
```

Description

This function will disable the synchronized comparator output with the Timer1 clock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_OutputSynchronousWithTimer1Disable(MY_CMP_INSTANCE);
```

5.6.5.5.2.10 PLIB_CMP_OutputSynchronousWithTimer1Enable Function

C

```
void PLIB_CMP_OutputSynchronousWithTimer1Enable(
```

```
    CMP_MODULE_ID index  
);
```

Description

This function will synchronize comparator output to the rising edge of the Timer1 clock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OutputSynchronousWithTimer1Enable(MY_CMP_INSTANCE);
```

5.6.5.5.2.11 PLIB_CMP_StopInIdleModeDisable Function

C

```
void PLIB_CMP_StopInIdleModeDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will continue operation of all enabled comparators when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_StopInIdleModeDisable(MY_CMP_INSTANCE);
```

5.6.5.5.2.12 PLIB_CMP_StopInIdleModeEnable Function

C

```
void PLIB_CMP_StopInIdleModeEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will discontinue operation of all comparators when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_CMP_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_StopInIdleModeEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3 Comparator AND Gate Configuration Functions

5.6.5.5.3.1 PLIB_CMP_AndGateAInputDisable Function

C

```
void PLIB_CMP_AndGateAInputDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will disable the A input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_AndGateAInputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.3.2 PLIB_CMP_AndGateAInputEnable Function**C**

```
void PLIB_CMP_AndGateAInputEnable(
    CMP_MODULE_ID index
);
```

Description

This function will select the A input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_AndGateAInputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3.3 PLIB_CMP_AndGateBInputDisable Function**C**

```
void PLIB_CMP_AndGateBInputDisable(
    CMP_MODULE_ID index
);
```

Description

This function will disable the B input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_AndGateBInputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.3.4 PLIB_CMP_AndGateBInputEnable Function**C**

```
void PLIB_CMP_AndGateBInputEnable(
    CMP_MODULE_ID index
);
```

Description

This function will select the B input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_AndGateBInputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3.5 PLIB_CMP_AndGateCInputEnable Function**C**

```
void PLIB_CMP_AndGateCInputEnable(
    CMP_MODULE_ID index
);
```

Description

This function will select the C input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_AndGateCInputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3.6 PLIB_CMP_AndGateInvertedAInputDisable Function**C**

```
void PLIB_CMP_AndGateInvertedAInputDisable(  
    CMP_MODULE_ID index  
)
```

Description

This function will disable the inverted A input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_AndGateInvertedAInputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.3.7 PLIB_CMP_AndGateInvertedAInputEnable Function**C**

```
void PLIB_CMP_AndGateInvertedAInputEnable(  
    CMP_MODULE_ID index  
)
```

Description

This function will enable the inverted A input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_AndGateInvertedAInputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3.8 PLIB_CMP_AndGateInvertedBInputDisable Function**C**

```
void PLIB_CMP_AndGateInvertedBInputDisable(  
    CMP_MODULE_ID index  
)
```

Description

This function will disable the inverted B input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_AndGateInvertedBInputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.3.9 PLIB_CMP_AndGateInvertedBInputEnable Function**C**

```
void PLIB_CMP_AndGateInvertedBInputEnable(  
    CMP_MODULE_ID index  
)
```

Description

This function will enable the inverted B input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_AndGateInvertedBInputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3.10 PLIB_CMP_AndGateInvertedCInputDisable Function**C**

```
void PLIB_CMP_AndGateInvertedCInputDisable(
    CMP_MODULE_ID index
);
```

Description

This function will disable the inverted C input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_AndGateInvertedCInputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.3.11 PLIB_CMP_AndGateInvertedCInputEnable Function**C**

```
void PLIB_CMP_AndGateInvertedCInputEnable(
    CMP_MODULE_ID index
);
```

Description

This function will enable the inverted C input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_AndGateInvertedCInputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3.12 PLIB_CMP_NegativeAndGateOutputDisable Function**C**

```
void PLIB_CMP_NegativeAndGateOutputDisable(
    CMP_MODULE_ID index
);
```

Description

This function will disable the negative AND gate output to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_NegativeAndGateOutputDisable (MY_CMP_INSTANCE);
```

5.6.5.5.3.13 PLIB_CMP_NegativeAndGateOutputEnable Function**C**

```
void PLIB_CMP_NegativeAndGateOutputEnable(
    CMP_MODULE_ID index
);
```

Description

This function will enable the negative AND gate output to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_NegativeAndGateOutputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3.14 PLIB_CMP_PositiveAndGateOutputDisable Function**C**

```
void PLIB_CMP_PositiveAndGateOutputDisable(
    CMP_MODULE_ID index
);
```

Description

This function will disable the positive AND gate output to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_PositiveAndGateOutputDisable (MY_CMP_INSTANCE);
```

5.6.5.5.3.15 PLIB_CMP_PositiveAndGateOutputEnable Function**C**

```
void PLIB_CMP_PositiveAndGateOutputEnable(
    CMP_MODULE_ID index
);
```

Description

This function will enable the positive AND gate output to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_PositiveAndGateOutputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3.16 PLIB_CMP_AndGateCInputDisable Function**C**

```
void PLIB_CMP_AndGateCInputDisable(
    CMP_MODULE_ID index
);
```

Description

This function will disable the C input as the input to the AND gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_AndGateCInputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.4 Comparator Filter Clock Configuration Functions**5.6.5.5.4.1 PLIB_CMP_FilterClockDivideSelect Function****C**

```
void PLIB_CMP_FilterClockDivideSelect(
    CMP_MODULE_ID index,
    CMP_CLOCK_DIVIDE scale
);
```

Description

This function will select the Comparator filter clock divide.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scale	One of the possible values from CMP_CLOCK_DIVIDE

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_FilterClockDivideSelect (MY_CMP_INSTANCE, CMP_FILTER_CLOCK_DIVIDE_1_2);
```

5.6.5.5.4.2 PLIB_CMP_FilterDisable Function**C**

```
void PLIB_CMP_FilterDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will disable the Comparator digital filter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_FilterDisable (MY_CMP_INSTANCE);
```

5.6.5.5.4.3 PLIB_CMP_FilterEnable Function**C**

```
void PLIB_CMP_FilterEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will enable the Comparator digital filter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_FilterEnable (MY_CMP_INSTANCE);
```

5.6.5.5.4 PLIB_CMP_FilterInputClockSelect Function**C**

```
void PLIB_CMP_FilterInputClockSelect(  
    CMP_MODULE_ID index,  
    CMP_FILTER_CLOCK inputclock  
);
```

Description

This function will select the Comparator filter input clock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
inputclock	One of the possible values from CMP_FILTER_CLOCK

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_FilterInputClockSelect (MY_CMP_INSTANCE, CMP_FILTER_CLOCK_FP);
```

5.6.5.5.5 Comparator Op amp Mode Configuration Functions**5.6.5.5.5.1 PLIB_CMP_OpAmpModeDisable Function****C**

```
void PLIB_CMP_OpAmpModeDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will cause the circuit to operate as a Comparator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OpAmpModeDisable(MY_CMP_INSTANCE);
```

5.6.5.5.2 PLIB_CMP_OpAmpModeEnable Function**C**

```
void PLIB_CMP_OpAmpModeEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will cause the circuit to operate as an Op amp.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OpAmpModeEnable(MY_CMP_INSTANCE);
```

5.6.5.5.3 PLIB_CMP_OpAmpOutputDisable Function**C**

```
void PLIB_CMP_OpAmpOutputDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will disable the Op amp output pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OpAmpOutputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.4 PLIB_CMP_OpAmpOutputEnable Function**C**

```
void PLIB_CMP_OpAmpOutputEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will enable the Op amp output pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OpAmpOutputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.6 Comparator OR Gate Configuration Functions**5.6.5.5.6.1 PLIB_CMP_OrGateAInputDisable Function****C**

```
void PLIB_CMP_OrGateAInputDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will disable the A input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateAInputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.6.2 PLIB_CMP_OrGateAInputEnable Function**C**

```
void PLIB_CMP_OrGateAInputEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will select the A input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateAInputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.6.3 PLIB_CMP_OrGateBInputDisable Function**C**

```
void PLIB_CMP_OrGateBInputDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will disable the B input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateBInputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.6 PLIB_CMP_OrGateBInputEnable Function**C**

```
void PLIB_CMP_OrGateBInputEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will select the B input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateBInputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.6 PLIB_CMP_OrGateCInputDisable Function**C**

```
void PLIB_CMP_OrGateCInputDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will disable the C input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateCInputDisable(MY_CMP_INSTANCE);
```

5.6.5.5.6 PLIB_CMP_OrGateCInputEnable Function**C**

```
void PLIB_CMP_OrGateCInputEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will select the C input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateCInputEnable(MY_CMP_INSTANCE);
```

5.6.5.5.6.7 PLIB_CMP_OrGateInvertedAInputDisable Function**C**

```
void PLIB_CMP_OrGateInvertedAInputDisable(  
    CMP_MODULE_ID index  
);
```

Description

This function will disable the inverted A input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateInvertedAInputDisable (MY_CMP_INSTANCE);
```

5.6.5.5.6.8 PLIB_CMP_OrGateInvertedAInputEnable Function**C**

```
void PLIB_CMP_OrGateInvertedAInputEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will select the inverted A input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateInvertedAInputEnable (MY_CMP_INSTANCE);
```

5.6.5.5.6.9 PLIB_CMP_OrGateInvertedBInputDisable Function**C**

```
void PLIB_CMP_OrGateInvertedBInputDisable(  
    CMP_MODULE_ID index  
);
```


Description

This function will disable the inverted B input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_OrGateInvertedBInputDisable (MY_CMP_INSTANCE);
```

5.6.5.5.6.10 PLIB_CMP_OrGateInvertedBInputEnable Function**C**

```
void PLIB_CMP_OrGateInvertedBInputEnable(
    CMP_MODULE_ID index
);
```

Description

This function will select the inverted B input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// PLIB_CMP_OrGateInvertedBInputEnable (MY_CMP_INSTANCE);
```

5.6.5.5.6.11 PLIB_CMP_OrGateInvertedCInputDisable Function**C**

```
void PLIB_CMP_OrGateInvertedCInputDisable(
    CMP_MODULE_ID index
);
```

Description

This function will disable the inverted C input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateInvertedCInputDisable (MY_CMP_INSTANCE);
```

5.6.5.5.6.12 PLIB_CMP_OrGateInvertedCInputEnable Function**C**

```
void PLIB_CMP_OrGateInvertedCInputEnable(  
    CMP_MODULE_ID index  
);
```

Description

This function will select the inverted C input as the input to the OR gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_OrGateInvertedCInputEnable (MY_CMP_INSTANCE);
```

5.6.5.5.7 Comparator Mask Select Configuration Functions

5.6.5.5.7.1 PLIB_CMP_HighLevelMaskSelect Function

C

```
void PLIB_CMP_HighLevelMaskSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function will select the high-level masking.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_HighLevelMaskSelect(MY_CMP_INSTANCE);
```

5.6.5.5.7.2 PLIB_CMP_LowLevelMaskSelect Function

C

```
void PLIB_CMP_LowLevelMaskSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function will select the low-level masking

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the  
// application developer.  
// PLIB_CMP_LowLevelMaskSelect(MY_CMP_INSTANCE);
```

5.6.5.5.8 Data Types and Constants

5.6.5.5.8.1 CMP_CLOCK_DIVIDE Enumeration

C

```
typedef enum {
    CMP_FILTER_CLOCK_DIVIDE_1_1,
    CMP_FILTER_CLOCK_DIVIDE_1_2,
    CMP_FILTER_CLOCK_DIVIDE_1_4,
    CMP_FILTER_CLOCK_DIVIDE_1_8,
    CMP_FILTER_CLOCK_DIVIDE_1_16,
    CMP_FILTER_CLOCK_DIVIDE_1_32,
    CMP_FILTER_CLOCK_DIVIDE_1_64,
    CMP_FILTER_CLOCK_DIVIDE_1_128
} CMP_CLOCK_DIVIDE;
```

Description

CMP Filter Clock Divide Select

This macro defines the CMP filter clock divide

Members

Members	Description
CMP_FILTER_CLOCK_DIVIDE_1_1	Comparator Filter Clock Division is 1:1
CMP_FILTER_CLOCK_DIVIDE_1_2	Comparator Filter Clock Division is 1:2
CMP_FILTER_CLOCK_DIVIDE_1_4	Comparator Filter Clock Division is 1:4
CMP_FILTER_CLOCK_DIVIDE_1_8	Comparator Filter Clock Division is 1:8
CMP_FILTER_CLOCK_DIVIDE_1_16	Comparator Filter Clock Division is 1:16
CMP_FILTER_CLOCK_DIVIDE_1_32	Comparator Filter Clock Division is 1:32
CMP_FILTER_CLOCK_DIVIDE_1_64	Comparator Filter Clock Division is 1:64
CMP_FILTER_CLOCK_DIVIDE_1_128	Comparator Filter Clock Division is 1:128

5.6.5.5.8.2 CMP_CVREF_BANDGAP_SELECT Enumeration

C

```
typedef enum {
    CMP_CVREF_BANDGAP_1_2V,
    CMP_CVREF_BANDGAP_0_6V,
    CMP_CVREF_BANDGAP_VREFPLUS
} CMP_CVREF_BANDGAP_SELECT;
```

Description

CVREF bandgap select enumeration

This enumeration lists the possible Bandgap selection options.

Members

Members	Description
CMP_CVREF_BANDGAP_1_2V	Select the Band Gap Reference Source as 1.2 V
CMP_CVREF_BANDGAP_0_6V	Select the Band Gap Reference Source as 0.6 V
CMP_CVREF_BANDGAP_VREFPLUS	Select the Band Gap Reference Source as VREF

Remarks

None.

5.6.5.5.8.3 CMP_CVREF_REFERENCE_SELECT Enumeration**C**

```
typedef enum {  
    CMP_CVREF_RESISTOR_LADDER_VOLTAGE,  
    CMP_CVREF_POSITIVE_REFERENCE_VOLTAGE  
} CMP_CVREF_REFERENCE_SELECT;
```

Description

CVREF reference select enumeration

This enumeration lists the possible reference selection options.

Members

Members	Description
CMP_CVREF_RESISTOR_LADDER_VOLTAGE	Select the Band Gap Reference Source as 1.2 V
CMP_CVREF_POSITIVE_REFERENCE_VOLTAGE	Select the Band Gap Reference Source as 0.6 V

Remarks

None.

5.6.5.5.8.4 CMP_CVREF_VALUE Enumeration**C**

```
typedef enum {  
    CMP_CVREF_VALUE_0,  
    CMP_CVREF_VALUE_1,  
    CMP_CVREF_VALUE_2,  
    CMP_CVREF_VALUE_3,  
    CMP_CVREF_VALUE_4,  
    CMP_CVREF_VALUE_5,  
    CMP_CVREF_VALUE_6,  
    CMP_CVREF_VALUE_7,  
    CMP_CVREF_VALUE_8,  
    CMP_CVREF_VALUE_9,  
    CMP_CVREF_VALUE_10,  
    CMP_CVREF_VALUE_11,  
    CMP_CVREF_VALUE_12,  
    CMP_CVREF_VALUE_13,  
    CMP_CVREF_VALUE_14,  
    CMP_CVREF_VALUE_15  
} CMP_CVREF_VALUE;
```

Description

CVREF voltage reference value selection enumeration

This enumeration lists the possible Voltage reference value selection options.

Members

Members	Description
CMP_CVREF_VALUE_0	Voltage reference value 0
CMP_CVREF_VALUE_1	Voltage reference value 1
CMP_CVREF_VALUE_2	Voltage reference value 2

CMP_CVREF_VALUE_3	Voltage reference value 3
CMP_CVREF_VALUE_4	Voltage reference value 4
CMP_CVREF_VALUE_5	Voltage reference value 5
CMP_CVREF_VALUE_6	Voltage reference value 6
CMP_CVREF_VALUE_7	Voltage reference value 7
CMP_CVREF_VALUE_8	Voltage reference value 8
CMP_CVREF_VALUE_9	Voltage reference value 9
CMP_CVREF_VALUE_10	Voltage reference value 10
CMP_CVREF_VALUE_11	Voltage reference value 11
CMP_CVREF_VALUE_12	Voltage reference value 12
CMP_CVREF_VALUE_13	Voltage reference value 13
CMP_CVREF_VALUE_14	Voltage reference value 14
CMP_CVREF_VALUE_15	Voltage reference value 15

Remarks

None.

5.6.5.5.8.5 CMP_CVREF_VOLTAGE_SOURCE Enumeration**C**

```
typedef enum {
    CMP_CVREF_VOLTAGE_SOURCE_INTERNAL,
    CMP_CVREF_VOLTAGE_SOURCE_EXTERNAL
} CMP_CVREF_VOLTAGE_SOURCE;
```

Description

CVREF Voltage source selection enumeration

This enumeration lists the possible Voltage source selection options.

Members

Members	Description
CMP_CVREF_VOLTAGE_SOURCE_INTERNAL	Select vdd/vss source
CMP_CVREF_VOLTAGE_SOURCE_EXTERNAL	Select external voltage source

Remarks

None.

5.6.5.5.8.6 CMP_FILTER_CLOCK Enumeration**C**

```
typedef enum {
    CMP_FILTER_CLOCK_FP,
    CMP_FILTER_CLOCK_FOSC,
    CMP_FILTER_CLOCK_SYNC01,
    CMP_FILTER_CLOCK_T2CLK,
    CMP_FILTER_CLOCK_T3CLK,
    CMP_FILTER_CLOCK_T4CLK,
    CMP_FILTER_CLOCK_T5CLK
} CMP_FILTER_CLOCK;
```

Description

CMP Filter Input Clock Select

This macro defines the CMP filter input clock

Members

Members	Description
CMP_FILTER_CLOCK_FP	FP will be the CMP filter input clock
CMP_FILTER_CLOCK_FOSC	FOSC will be the CMP filter input clock
CMP_FILTER_CLOCK_SYNCO1	SYNCO1 will be the CMP filter input clock
CMP_FILTER_CLOCK_T2CLK	T2CLK will be the CMP filter input clock
CMP_FILTER_CLOCK_T3CLK	T3CLK will be the CMP filter input clock
CMP_FILTER_CLOCK_T4CLK	T4CLK will be the CMP filter input clock
CMP_FILTER_CLOCK_T5CLK	T5CLK will be the CMP filter input clock

5.6.5.5.8.7 CMP_INTERRUPT_EVENT Enumeration

C

```
typedef enum {
    CMP_INTERRUPT_GENERATION_DISABLED
} CMP_INTERRUPT_EVENT;
```

Description

CMP interrupt event Select

This macro defines the CMP interrupt events.

Members

Members	Description
CMP_INTERRUPT_GENERATION_DISABLED	Select vdd/vss source

5.6.5.5.8.8 CMP_INVERTING_INPUT Enumeration

C

```
typedef enum {
    CMP_INPUT_C2IN_NEGATIVE
} CMP_INVERTING_INPUT;
```

Description

CMP inverting input channel select

This macro defines the list of CMP inverting Input

Members

Members	Description
CMP_INPUT_C2IN_NEGATIVE	Select vdd/vss source

5.6.5.5.8.9 CMP_MASK_A Enumeration

C

```
typedef enum {
    CMP_MASK_A_PWM1L,
    CMP_MASK_A_PWM1H,
    CMP_MASK_A_PWM2L,
    CMP_MASK_A_PWM2H,
    CMP_MASK_A_PWM3L,
    CMP_MASK_A_PWM3H,
    CMP_MASK_A_PTG018,
```

```

    CMP_MASK_A_PTGO19,
    CMP_MASK_A_FLT2,
    CMP_MASK_A_FLT4
} CMP_MASK_A;

```

Description

CMP Mask A Input Select

This macro defines the CMP Mask A Input.

Members

Members	Description
CMP_MASK_A_PWM1L	PWM1L will be the CMP mask A input
CMP_MASK_A_PWM1H	PWM1H will be the CMP mask A input
CMP_MASK_A_PWM2L	PWM2L will be the CMP mask A input
CMP_MASK_A_PWM2H	PWM2H will be the CMP mask A input
CMP_MASK_A_PWM3L	PWM3L will be the CMP mask A input
CMP_MASK_A_PWM3H	PWM3H will be the CMP mask A input
CMP_MASK_A_PTGO18	PTGO18 will be the CMP mask A input
CMP_MASK_A_PTGO19	PTGO19 will be the CMP mask A input
CMP_MASK_A_FLT2	FLT2 will be the CMP mask A input
CMP_MASK_A_FLT4	FLT4 will be the CMP mask A input

5.6.5.5.8.10 CMP_MASK_B Enumeration**C**

```

typedef enum {
    CMP_MASK_B_PWM1L,
    CMP_MASK_B_PWM1H,
    CMP_MASK_B_PWM2L,
    CMP_MASK_B_PWM2H,
    CMP_MASK_B_PWM3L,
    CMP_MASK_B_PWM3H,
    CMP_MASK_B_PTGO18,
    CMP_MASK_B_PTGO19,
    CMP_MASK_B_FLT2,
    CMP_MASK_B_FLT4
} CMP_MASK_B;

```

Description

CMP Mask B Input Select

This macro defines the CMP Mask B Input.

Members

Members	Description
CMP_MASK_B_PWM1L	PWM1L will be the CMP mask B input
CMP_MASK_B_PWM1H	PWM1H will be the CMP mask B input
CMP_MASK_B_PWM2L	PWM2L will be the CMP mask B input
CMP_MASK_B_PWM2H	PWM2H will be the CMP mask B input
CMP_MASK_B_PWM3L	PWM3L will be the CMP mask B input
CMP_MASK_B_PWM3H	PWM3H will be the CMP mask B input
CMP_MASK_B_PTGO18	PTGO18 will be the CMP mask B input
CMP_MASK_B_PTGO19	PTGO19 will be the CMP mask B input

CMP_MASK_B_FLT2	FLT2 will be the CMP mask B input
CMP_MASK_B_FLT4	FLT4 will be the CMP mask B input

5.6.5.5.8.11 CMP_MASK_C Enumeration

C

```
typedef enum {
    CMP_MASK_C_PWM1L,
    CMP_MASK_C_PWM1H,
    CMP_MASK_C_PWM2L,
    CMP_MASK_C_PWM2H,
    CMP_MASK_C_PWM3L,
    CMP_MASK_C_PWM3H,
    CMP_MASK_C_PTGO18,
    CMP_MASK_C_PTGO19,
    CMP_MASK_C_FLT2,
    CMP_MASK_C_FLT4
} CMP_MASK_C;
```

Description

CMP Mask C Input Select

This macro defines the CMP Mask C Input.

Members

Members	Description
CMP_MASK_C_PWM1L	PWM1L will be the CMP mask C input
CMP_MASK_C_PWM1H	PWM1H will be the CMP mask C input
CMP_MASK_C_PWM2L	PWM2L will be the CMP mask C input
CMP_MASK_C_PWM2H	PWM2H will be the CMP mask C input
CMP_MASK_C_PWM3L	PWM3L will be the CMP mask C input
CMP_MASK_C_PWM3H	PWM3H will be the CMP mask C input
CMP_MASK_C_PTGO18	PTGO18 will be the CMP mask C input
CMP_MASK_C_PTGO19	PTGO19 will be the CMP mask C input
CMP_MASK_C_FLT2	FLT2 will be the CMP mask C input
CMP_MASK_C_FLT4	FLT4 will be the CMP mask C input

5.6.5.5.8.12 CMP_MODULE_ID Enumeration

C

```
typedef enum {
    CMP_ID_1,
    CMP_ID_2,
    CMP_ID_3
} CMP_MODULE_ID;
```

Description

CMP Module ID

This enumeration identifies the CMP modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

Members

Members	Description
CMP_ID_1	CMP Module 1 ID

CMP_ID_2	CMP Module 2 ID
CMP_ID_3	CMP Module 3 ID

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

5.6.5.5.8.13 CMP_NON_INVERTING_INPUT Enumeration**C**

```
typedef enum {
    CMP_INPUT_C2IN_POSITIVE,
    CMP_INPUT_INTERNAL_CVREF
} CMP_NON_INVERTING_INPUT;
```

Description

CMP Non inverting input channel select

This macro defines the list of CMP non inverting Input

Members

Members	Description
CMP_INPUT_C2IN_POSITIVE	Select vdd/vss source
CMP_INPUT_INTERNAL_CVREF	Select external voltage source

5.6.5.5.8.14 CMP_SPEED_POWER Enumeration**C**

```
typedef enum {
    CMP_SPEED_POWER_LOWER,
    CMP_SPEED_POWER_HIGHER
} CMP_SPEED_POWER;
```

Description

CMP Speed/Power Select

This macro defines the Speed/Power of CMP

Members

Members	Description
CMP_SPEED_POWER_LOWER	CMP low-power, low-speed
CMP_SPEED_POWER_HIGHER	CMP normal power, higher speed

5.6.5.5.9 Feature Existence Functions**5.6.5.5.9.1 PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect Function****C**

```
bool PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect(
    CMP_MODULE_ID index
);
```

Description

This function identifies whether the CVREFBGRefVoltageRangeSelect feature is available on the Comparator module. When this function returns true, these functions are supported on the device:

- [PLIB_CMP_CVREF_BandGapReferenceSourceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The CVREFBGRefVoltageRangeSelect feature is supported on the device
- false = The CVREFBGRefVoltageRangeSelect feature is not supported on the device

Remarks

None.

5.6.5.5.9.2 PLIB_CMP_ExistsCVREFEnableControl Function**C**

```
bool PLIB_CMP_ExistsCVREFEnableControl(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the CVREFEnableControl feature is available on the Comparator module. When this function returns true, these functions are supported on the device:

- [PLIB_CMP_CVREF_Enable](#)
- [PLIB_CMP_CVREF_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The CVREFEnableControl feature is supported on the device
- false = The CVREFEnableControl feature is not supported on the device

Remarks

None.

5.6.5.5.9.3 PLIB_CMP_ExistsCVREFOutputEnableControl Function**C**

```
bool PLIB_CMP_ExistsCVREFOutputEnableControl(  
    CMP_MODULE_ID index
```

```
);
```

Description

This function identifies whether the CVREFOutputEnableControl feature is available on the Comparator module. When this function returns true, these functions are supported on the device:

- [PLIB_CMP_CVREF_OutputEnable](#)
- [PLIB_CMP_CVREF_OutputDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The CVREFOutputEnableControl feature is supported on the device
- false = The CVREFOutputEnableControl feature is not supported on the device

Remarks

None.

5.6.5.5.9.4 PLIB_CMP_ExistsCVREFRefVoltageRangeSelect Function

C

```
bool PLIB_CMP_ExistsCVREFRefVoltageRangeSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the CVREFRefVoltageRangeSelect feature is available on the Comparator module. When this function returns true, the following function is supported on the device:

- [PLIB_CMP_CVREF_ReferenceVoltageSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The CVREFRefVoltageRangeSelect feature is supported on the device
- false = The CVREFRefVoltageRangeSelect feature is not supported on the device

Remarks

None.

5.6.5.5.9.5 PLIB_CMP_ExistsCVREFValueSelect Function

C

```
bool PLIB_CMP_ExistsCVREFValueSelect(  

```

```
    CMP_MODULE_ID index
);
```

Description

This function identifies whether the CVREFValueSelect feature is available on the Comparator module. When this function returns true, the following function is supported on the device:

- [PLIB_CMP_CVREF_ValueSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The CVREFValueSelect feature is supported on the device
- false = The CVREFValueSelect feature is not supported on the device

Remarks

None.

5.6.5.5.9.6 PLIB_CMP_ExistsCVREFVoltageRangeSelect Function

C

```
bool PLIB_CMP_ExistsCVREFVoltageRangeSelect(
    CMP_MODULE_ID index
);
```

Description

This function identifies whether the CVREFVoltageRangeSelect feature is available on the Comparator module. When this function returns true, the following function is supported on the device:

- [PLIB_CMP_CVREF_SourceVoltageSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The CVREFVoltageRangeSelect feature is supported on the device
- false = The CVREFVoltageRangeSelect feature is not supported on the device

Remarks

None.

5.6.5.5.9.7 PLIB_CMP_ExistsCVREFWideRangeControl Function

C

```
bool PLIB_CMP_ExistsCVREFWideRangeControl(
    CMP_MODULE_ID index
);
```

```
);
```

Description

This function identifies whether the CVREFWideRangeControl feature is available on the Comparator module. When this function returns true, these functions are supported on the device:

- [PLIB_CMP_CVREF_WideRangeEnable](#)
- [PLIB_CMP_CVREF_WideRangeDisable](#)
- [PLIB_CMP_CVREF_WideRangeIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The CVREFWideRangeControl feature is supported on the device
- false = The CVREFWideRangeControl feature is not supported on the device

Remarks

None.

5.6.5.5.9.8 PLIB_CMP_ExistsEnableControl Function

C

```
bool PLIB_CMP_ExistsEnableControl(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the ComparatorEnableControl feature is available on the Comparator module. When this function returns true, these functions are supported on the device:

- [PLIB_CMP_Enable](#)
- [PLIB_CMP_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ComparatorEnableControl feature is supported on the device
- false = The ComparatorEnableControl feature is not supported on the device

Remarks

None.

5.6.5.5.9.9 PLIB_CMP_ExistsInterruptEventSelect Function

C

```
bool PLIB_CMP_ExistsInterruptEventSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the InterruptEventSelect feature is available on the Comparator module. When this function returns true, this function is supported on the device:

- [PLIB_CMP_InterruptEventSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The InterruptEventSelect feature is supported on the device
- false = The InterruptEventSelect feature is not supported on the device

Remarks

None.

5.6.5.5.9.10 PLIB_CMP_ExistsInvertingInputSelect Function

C

```
bool PLIB_CMP_ExistsInvertingInputSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the InvertingInputSelect feature is available on the Comparator module. When this function returns true, this function is supported on the device:

- [PLIB_CMP_InvertingInputChannelSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The InvertingInputSelect feature is supported on the device
- false = The InvertingInputSelect feature is not supported on the device

Remarks

None.

5.6.5.5.9.11 PLIB_CMP_ExistsInvertOutputControl Function

C

```
bool PLIB_CMP_ExistsInvertOutputControl(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the InvertOutputSelectControl feature is available on the Comparator module. When this function returns true, these functions are supported on the device:

- [PLIB_CMP_InvertedOutputSelect](#)
- [PLIB_CMP_NonInvertedOutputSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The InvertOutputSelectControl feature is supported on the device
- false = The InvertOutputSelectControl feature is not supported on the device

Remarks

None.

5.6.5.5.9.12 PLIB_CMP_ExistsNonInvertingInputSelect Function

C

```
bool PLIB_CMP_ExistsNonInvertingInputSelect(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the NonInvertingInputSelect feature is available on the Comparator module. When this function returns true, this function is supported on the device:

- PLIB_CMP_adc

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The NonInvertingInputSelect feature is supported on the device
- false = The NonInvertingInputSelect feature is not supported on the device

Remarks

None.

5.6.5.5.9.13 PLIB_CMP_ExistsOutputEnableControl Function

C

```
bool PLIB_CMP_ExistsOutputEnableControl(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the ComparatorOutputEnableControl feature is available on the Comparator module. When this function returns true, these functions are supported on the device:

- [PLIB_CMP_OutputEnable](#)
- [PLIB_CMP_OutputDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The ComparatorOutputEnableControl feature is supported on the device
- false = The ComparatorOutputEnableControl feature is not supported on the device

Remarks

None.

5.6.5.5.9.14 PLIB_CMP_ExistsOutputLevelControl Function

C

```
bool PLIB_CMP_ExistsOutputLevelControl(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the OutputLevelSelectControl feature is available on the Comparator module. When this function returns true, these functions are supported on the device:

- [PLIB_CMP_OutputHighSelect](#)
- [PLIB_CMP_OutputLowSelect](#)
- [PLIB_CMP_Cmp1OutputHighSelect](#)
- [PLIB_CMP_Cmp1OutputLowSelect](#)
- [PLIB_CMP_Cmp2OutputHighSelect](#)
- [PLIB_CMP_Cmp2OutputLowSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The OutputLevelSelectControl feature is supported on the device
- false = The OutputLevelSelectControl feature is not supported on the device

Remarks

None.

5.6.5.5.9.15 PLIB_CMP_ExistsStopInIdle Function**C**

```
bool PLIB_CMP_ExistsStopInIdle(  
    CMP_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the Comparator module. When this function returns true, these functions are supported on the device:

- [PLIB_CMP_StopInIdleModeEnable](#)
- [PLIB_CMP_StopInIdleModeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true = The StopInIdle feature is supported on the device
- false = The StopInIdle feature is not supported on the device

Remarks

None.

5.6.5.6 Files**Files**

Name	Description
plib_cmp.h	Comparator Peripheral Library Interface Header for Comparator module definitions.

Description**5.6.5.6.1 plib_cmp.h**

Comparator Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Comparator Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the

Comparator peripheral.

Functions

	Name	Description
≡	PLIB_CMP_AndGateAInputDisable	A input disabled as input to the AND gate
≡	PLIB_CMP_AndGateAInputEnable	A input enabled as input to the AND gate.
≡	PLIB_CMP_AndGateBInputDisable	B input disabled as input to the AND gate.
≡	PLIB_CMP_AndGateBInputEnable	B input enabled as input to the AND gate.
≡	PLIB_CMP_AndGateCInputDisable	C input disabled as input to the AND gate.
≡	PLIB_CMP_AndGateCInputEnable	C input enabled as input to the AND gate.
≡	PLIB_CMP_AndGateInvertedAInputDisable	Inverted A input disabled as input to the AND gate.
≡	PLIB_CMP_AndGateInvertedAInputEnable	Inverted A input enabled as input to the AND gate.
≡	PLIB_CMP_AndGateInvertedBInputDisable	Inverted B input disabled as input to the AND gate.
≡	PLIB_CMP_AndGateInvertedBInputEnable	Inverted B input enabled as input to the AND gate.
≡	PLIB_CMP_AndGateInvertedCInputDisable	Inverted C input disabled as input to the AND gate.
≡	PLIB_CMP_AndGateInvertedCInputEnable	Inverted C input enabled as input to the AND gate.
≡	PLIB_CMP_Cmp1OutputHighSelect	Comparator output bit is set to '1'.
≡	PLIB_CMP_Cmp1OutputLowSelect	Comparator output bit is set to '0'.
≡	PLIB_CMP_Cmp2OutputHighSelect	Comparator output bit is set to '1'.
≡	PLIB_CMP_Cmp2OutputLowSelect	Comparator output bit is set to '0'.
≡	PLIB_CMP_CVREF_BandGapReferenceSourceSelect	Selects the band gap reference voltage source.
≡	PLIB_CMP_CVREF_Disable	Disables the voltage reference of the Comparator module.
≡	PLIB_CMP_CVREF_Enable	Enables the voltage reference of the Comparator module.
≡	PLIB_CMP_CVREF_Output2Disable	Disables the Vsource/2 voltage output.
≡	PLIB_CMP_CVREF_Output2Enable	Enables the Vsource/2 voltage output.
≡	PLIB_CMP_CVREF_OutputDisable	Disables the output voltage.
≡	PLIB_CMP_CVREF_OutputEnable	Enables the voltage output.
≡	PLIB_CMP_CVREF_ReferenceVoltageSelect	Selects the voltage reference value.
≡	PLIB_CMP_CVREF_SourceNegativeInputSelect	Configures the Comparator module to use the selected input as a negative reference.
≡	PLIB_CMP_CVREF_SourceVoltageSelect	Connects the Comparator module to the selected voltage source.
≡	PLIB_CMP_CVREF_ValueGet	Returns the voltage setting value.
≡	PLIB_CMP_CVREF_ValueSelect	Selects the voltage reference value.
≡	PLIB_CMP_CVREF_WideRangeDisable	Disables the wide range.
≡	PLIB_CMP_CVREF_WideRangeEnable	Enables the wide range.
≡	PLIB_CMP_CVREF_WideRangeIsEnabled	Returns whether the wide range is selected for the reference voltage.
≡	PLIB_CMP_Disable	Disables the Comparator module.
≡	PLIB_CMP_Enable	Enables the Comparator module.
≡	PLIB_CMP_EventHasOccurred	Comparator event status.
≡	PLIB_CMP_EventStatusClear	Comparator event clear.
≡	PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect	Identifies whether the CVREFBGRefVoltageRangeSelect feature exists on the Comparator module.
≡	PLIB_CMP_ExistsCVREFEnableControl	Identifies whether the CVREFEnableControl feature exists on the Comparator module.

	PLIB_CMP_ExistsCVREFOutputEnableControl	Identifies whether the CVREFOutputEnableControl feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFRefVoltageRangeSelect	Identifies whether the CVREFRefVoltageRangeSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFValueSelect	Identifies whether the CVREFValueSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFVoltageRangeSelect	Identifies whether the CVREFVoltageRangeSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsCVREFWideRangeControl	Identifies whether the CVREFWideRangeControl feature exists on the Comparator module.
	PLIB_CMP_ExistsEnableControl	Identifies whether the ComparatorEnableControl feature exists on the Comparator module.
	PLIB_CMP_ExistsInterruptEventSelect	Identifies whether the InterruptEventSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsInvertingInputSelect	Identifies whether the InvertingInputSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsInvertOutputControl	Identifies whether the InvertOutputSelectControl feature exists on the Comparator module.
	PLIB_CMP_ExistsNonInvertingInputSelect	Identifies whether the NonInvertingInputSelect feature exists on the Comparator module.
	PLIB_CMP_ExistsOutputEnableControl	Identifies whether the ComparatorOutputEnableControl feature exists on the Comparator module.
	PLIB_CMP_ExistsOutputLevelControl	Identifies whether the OutputLevelSelectControl feature exists on the Comparator module.
	PLIB_CMP_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Comparator module.
	PLIB_CMP_FilterClockDivideSelect	Comparator filter clock divide select.
	PLIB_CMP_FilterDisable	Comparator filter disable.
	PLIB_CMP_FilterEnable	Comparator filter enable.
	PLIB_CMP_FilterInputClockSelect	Comparator filter input clock select.
	PLIB_CMP_HighLevelMaskSelect	High-level masking select.
	PLIB_CMP_HysteresisDisable	Comparator hysteresis disable.
	PLIB_CMP_HysteresisEnable	Comparator hysteresis enable.
	PLIB_CMP_InterruptEventSelect	Comparator interrupt event select.
	PLIB_CMP_InvertedOutputSelect	Comparator output is inverted.
	PLIB_CMP_InvertingInputChannelSelect	Comparator inverting input channel select.
	PLIB_CMP_LowLevelMaskSelect	Low-level masking select.
	PLIB_CMP_NegativeAndGateOutputDisable	Negative AND gate output disable.
	PLIB_CMP_NegativeAndGateOutputEnable	Negative AND gate output enable.
	PLIB_CMP_NonInvertedOutputSelect	Comparator output is non-inverted.
	PLIB_CMP_NonInvertingInputChannelSelect	Comparator input channel select.
	PLIB_CMP_OpAmpModeDisable	Op amp operation mode disable.
	PLIB_CMP_OpAmpModeEnable	Op amp operation mode enable.
	PLIB_CMP_OpAmpOutputDisable	Op amp output disable.
	PLIB_CMP_OpAmpOutputEnable	Op amp output enable.
	PLIB_CMP_OrGateAInputDisable	OR gate A input disable.
	PLIB_CMP_OrGateAInputEnable	A input connected to OR gate.
	PLIB_CMP_OrGateBInputDisable	OR gate B input disable.

	PLIB_CMP_OrGateBInputEnable	B input connected to OR gate.
	PLIB_CMP_OrGateCInputDisable	OR gate C input disable.
	PLIB_CMP_OrGateCInputEnable	C input connected to OR gate.
	PLIB_CMP_OrGateInvertedAInputDisable	OR gate inverted A input disable.
	PLIB_CMP_OrGateInvertedAInputEnable	Inverted A input connected to OR gate.
	PLIB_CMP_OrGateInvertedBInputDisable	OR gate inverted B input disable.
	PLIB_CMP_OrGateInvertedBInputEnable	Inverted B input connected to OR gate
	PLIB_CMP_OrGateInvertedCInputDisable	OR gate inverted C input disable.
	PLIB_CMP_OrGateInvertedCInputEnable	Inverted C input connected to OR gate
	PLIB_CMP_OutputDisable	Disables the Comparator output.
	PLIB_CMP_OutputEnable	Enables the Comparator output.
	PLIB_CMP_OutputHighSelect	Comparator output bit is set to '1'.
	PLIB_CMP_OutputLowSelect	Comparator output bit is set to '0'.
	PLIB_CMP_OutputStatusGet	Comparator output status.
	PLIB_CMP_OutputSynchronousWithTimer1Disable	Comparator output is asynchronous.
	PLIB_CMP_OutputSynchronousWithTimer1Enable	Comparator Output Synchronous mode select.
	PLIB_CMP_PositiveAndGateOutputDisable	Positive AND gate output disable.
	PLIB_CMP_PositiveAndGateOutputEnable	Positive AND gate output enable.
	PLIB_CMP_StopInIdleModeDisable	Disables Stop in Idle mode.
	PLIB_CMP_StopInIdleModeEnable	Enables Stop in Idle mode.

File Name

plib_cmp.h

Company

Microchip Technology Inc.

5.6.6 DMA Peripheral Library Help

5.6.6.1 Introduction

DMA Peripheral Library

for

Microchip Microcontrollers

Direct Memory Access (DMA) is a feature (sub-system) of the microcontroller that allows certain hardware sub-systems within the microcontroller to access the system memory independently of the processor (or the CPU).

Description

DMA also assists in accessing peripheral memory along with the system memory (data RAM). The DMA sub-system in the PIC family of microcontrollers is optimized for high-performance, real-time, embedded applications where determinism and system latency are priorities.

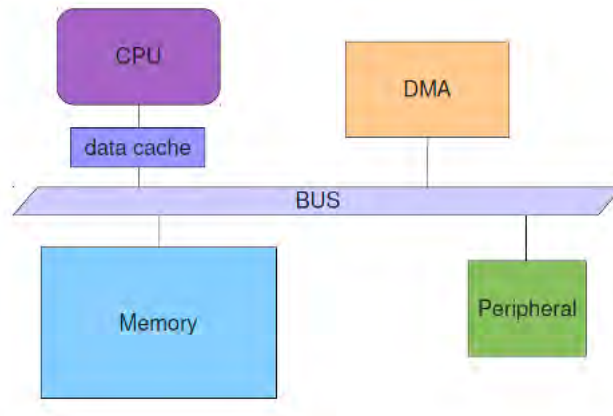


Figure-1: DMA Hardware Abstraction Model

5.6.6.2 Release Notes

MPLAB Harmony Version: v0.70b **DMA PLIB Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.6.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.6.4 Using the Library

This topic describes the basic architecture of the DMA Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_dma.h](#)

The interface to the DMA Peripheral Library is defined in the "[plib_dma.h](#)" header file. Any C language source (.c) file that uses the DMA Peripheral library should include "[plib_dma.h](#)".

Library File:

The DMA Peripheral Library (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the peripheral interacts with the framework.

5.6.6.4.1 Hardware Abstraction Model

This library provides the low-level abstraction of the DMA module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

The DMA Peripheral Library software is depicted using the following block diagram.

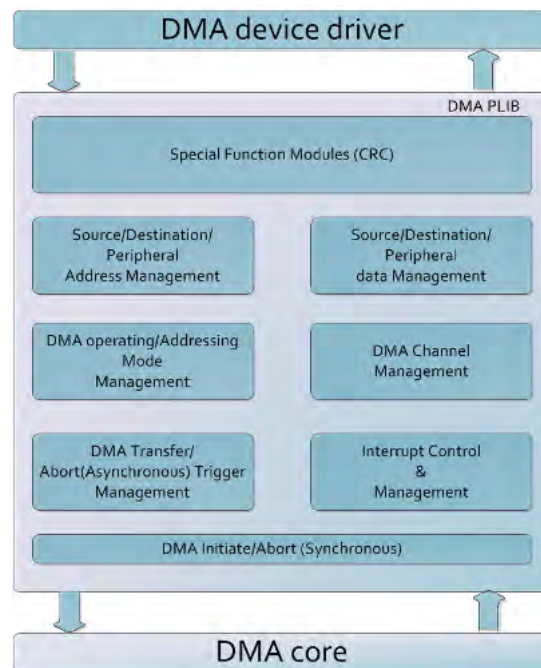


Figure-2: DMA Software Abstraction Block Diagram

The DMA Peripheral Library takes requests from application software or the DMA device driver and controls the DMA core (the DMA hardware) as per the inputs passed to the library.

The major components of the DMA Peripheral Library are:

- Source/Destination/Peripheral Address Management
- Source/Destination/Peripheral Data Management
- DMA Operating/Addressing Mode Management
- DMA Channel Management
- Interrupt Control and Management

The topic [Library Overview](#) explains in detail each block of the DMA Peripheral Library software architecture.

Note: The interface provided is a superset of all the functionality of the available DMA module on the device. Refer to the specific device data sheet or related family reference manual to determine the set of functions that are supported for each DMA module on your device.

5.6.6.4.2 Library Usage Model

The library interface routines are divided into various topics, each of which addresses one of the blocks or the overall operation of the DMA module.

Library Interface Section	Description
Transfer/Abort (Synchronous)	Handles the synchronous DMA transfer start and End. This feature request is controlled fully by the software.
Transfer/Abort (Asynchronous) Trigger Management	Each DMA transaction can be asynchronously initiated or aborted by an interrupt. This feature handles the required configuration.
Interrupt Control and Management	Each DMA channel can be configured to respond to events such as address error (upper address error, lower address error, etc.), transfer count half done, transfer count done, source data completely transferred, etc.
Operating/Addressing Mode Management	Before initiating a DMA transaction, the addressing and operating modes of the DMA module need to be configured (per channel basis). As an example, the DMA module can be configured for one shot mode with peripheral indirect addressing mode for a DMA transaction.
Channel Management	Each DMA channel can be individually configured for a DMA transaction. Priorities can be assigned, a channel can be enabled or disabled, channel chaining can be used, etc.
Source/Destination/Peripheral Address Management	For each DMA transaction to start, the DMA core needs to know the source and destination addresses. This feature handles this type of configuration.
Source/Destination/Peripheral Data Management	For each DMA transaction, the amount of data to be transferred, the source and destination size, the transaction type (byte/word), and transaction size can be configured by this feature.
Special Function Modules (CRC)	Certain high-end controllers support this CRC generator. This feature can be assigned to any of the available DMA channels. This feature handles the configuration of these aspects.
Status (Including Channel)	This feature specifies the application to access the status information of the DMA and each channel.

5.6.6.4.3 How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine which modes are supported.

5.6.6.4.3.1 General Configuration

DMA General configuration includes enabling, disabling DMA module globally. It also involves DMA suspend control and behavior in Idle mode.

Enable/Disable DMA

Enabling and disabling the DMA are the basic routines that may be called during initialization and exit routines of the application.

DMA Enable/Disable Example:


```
// Enable the DMA controller
PLIB_DMA_Enable ( DMA_ID_0 );
// Disable the DMA
PLIB_DMA_Disable ( DMA_ID_0 );
// Check the enable status of the DMA controller.
if(PLIB_DMA_IsEnabled( DMA_ID_0 ))
{
    // application code
}
```

5.6.6.4.3.2 Transfer/Abort (Synchronous)

A DMA transaction can be initiated/aborted asynchronously by configuring an interrupt for a channel. Refer to [Transfer/Abort \(Asynchronous\) Trigger Management](#) for details.

In many instances, an application or a device driver needs to initiate a DMA transfer or abort an ongoing DMA transfer for reasons not necessarily known. To handle such cases, the DMA core supports a forceful DMA start and abort.

Note: The DMA abort is not supported by all devices. Refer to the specific device data sheet to determine availability of this feature.

When a DMA force start request is sent and the channel is already processing/pending to process an interrupt-based DMA transfer, the force transfer request will be ignored. In certain devices, a flag is raised to indicate a collision of DMA requests. Refer to [Status \(Including Channel\)](#) for details. In certain devices, an interrupt is issued to indicate that the active channel may not be able to keep up with the demands from the peripheral module being serviced. For more details refer to [Interrupt Control and Management](#).

A DMA force transfer is initiated by the [PLIB_DMA_StartTransferSet Function](#).

A DMA force abort is initiated by the [PLIB_DMA_AbortTransferSet Function](#).

DMA Transfer/Abort (Synchronous) Configuration

A DMA transfer can be initiated or aborted forcefully by the software.

Example:

```
// Initiate a forced transfer on channel-3
PLIB_DMA_StartTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
// Abort a transfer on channel-3
PLIB_DMA_AbortTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
```

5.6.6.4.3.3 Transfer/Abort (Asynchronous) Trigger Management

DMA Start IRQ

Each channel can be configured to initiate a DMA transaction based on the interrupt from a source. This can be enabled/disable per channel basis. To initiate a DMA transfer upon an interrupt trigger, the IRQ number needs to be configured as the start IRQ for this channel. Which source can act as the start IRQ is device-dependent. In certain devices, any of the available IRQs can be configured as the starting IRQ, while in other devices, there is restriction on which interrupt source can act as the start IRQ. Refer to the specific device data sheet to determine feature availability.

The starting IRQ is enabled by the [PLIB_DMA_ChannelXTriggerEnable Function](#). A DMA channel can be configured to start a DMA transfer based on the event on an IRQ using the [PLIB_DMA_ChannelXStartIRQSet Function](#).

DMA Abort IRQ

Each channel can be configured to abort a DMA transaction based on the interrupt from a source. This can be enabled/disabled per channel basis. To abort a DMA transfer upon an interrupt trigger, the IRQ number needs to be configured as the abort IRQ for this channel.

The abort IRQ is enabled by the [PLIB_DMA_ChannelXTriggerEnable Function](#). A DMA channel can be configured to abort a

DMA transfer based on the event on an IRQ using the `ChannelXAbortIRQSet` Function.

DMA Pattern Matching Termination

Pattern Match Termination mode allows the user to end a transfer if a byte of data written during a transaction matches a specific pattern. This feature is useful in applications where a variable data size is required and eases the setup of the DMA channel. The UART module is a good example of where this can be effectively used. For details about the pattern data configuration refer to [Channel Management](#).

DMA Pattern match mode is enabled using the [PLIB_DMA_ChannelXTriggerEnable Function](#). It can also be disabled using the [PLIB_DMA_ChannelXTriggerDisable Function](#).

DMA Transfer/Abort (Asynchronous) Configuration

A DMA transfer can also be triggered/aborted asynchronously by using an interrupt event. The interrupt event required to initiate/abort a DMA transfer for a channel needs to be set. Prior to this, the start IRQ and abort IRQ both need to be enabled.

Example:

```
// Enable the start IRQ. An interrupt event can trigger a DMA transfer.
PLIB_DMA_ChannelXTriggerEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_TRANSFER_START );
// Enable the abort IRQ. An interrupt event can abort a DMA transfer on channel-3
PLIB_DMA_ChannelXTriggerEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_TRANSFER_ABORT );
// Set the start IRQ to UART-2 RX Interrupt
PLIB_DMA_ChannelXStartIRQSet ( DMA_ID_0, DMA_CHANNEL_3, DMA_TRIGGER_USART_2_RECEIVE );
// Set the abort IRQ to UART-2 error Interrupt
PLIB_DMA_ChannelXAbortIRQSet ( DMA_ID_0, DMA_CHANNEL_3, DMA_TRIGGER_USART_2_ERROR );
```

In some devices, abort IRQ trigger is not supported. Refer to the specific device data sheet to determine availability.

DMA Transfer Pattern Match Abort Configuration

The DMA transfer can also be aborted on a pattern match trigger (supported on PIC32 devices). Refer to Transfer/Abort (Asynchronous) Trigger Management for more information on pattern match abort. To use this feature, it must first be enabled. The pattern data needs to be programmed for the required channel. Once this is done, when a DMA transfer is in progress on that particular channel, if the DMA controller comes across the data matching the programmed data the transfer is automatically aborted.

Example:

```
// Enable the pattern match abort for channel-3
PLIB_DMA_ChannelXTriggerEnable ( DMA_ID_0, DMA_CHANNEL_3,
DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT );
// Set a pattern data to match
PLIB_DMA_ChannelXPatternDataSet ( DMA_ID_0, DMA_CHANNEL_3, 0xEE );
```

5.6.6.4.3.4 Interrupt Control and Management

The DMA module has the ability to generate interrupts reflecting the events that occur during the channel's data transfer.

The following are all of the possible interrupts generated during DMA operation. This topic provides information for configuring these interrupts.

Error Interrupts

This type of event is signalled when an address error has occurred during the channel transfer operation. An address error also can occur if a DMA operation accesses an address beyond configured address limits (low and high).

Abort Interrupts

This type of event is signalled when a DMA transfer is aborted because of a system event matching the abort IRQ configured for the current channel.

DMA Completion Interrupts (Block Completion Interrupts)

This event occurs when a channel transfers a block of configured data.

Call Complete Interrupts

This event occurs when a channel completes a cell sized data transfer.

DMA Midpoint Interrupts

This event occurs when the channel is at the midpoint of completing the data transfer.

Overrun Interrupts

When a DMA channel receives a trigger (software or hardware) while it is servicing another request, an overrun condition occurs. This indicates that the channel is being requested before its current transaction is finished. An overrun condition will not result in termination of the current transaction.

Source Address Pointer Activity Interrupts

This event occurs when the channel source pointer reached the end of the source or when it reaches the midpoint of the source depending on the way this interrupt is configured.

Destination Address Pointer Activity Interrupts

This event occurs when the channel destination pointer reached the end of the destination or when it reaches the midpoint of the destination depending on the way this interrupt is configured.

This feature helps in configuring the previously listed interrupts. Depending on the device selected, all or some of the interrupts can be individually enabled and disabled. In addition, this feature obtains the status of the interrupts previously mentioned.

All of the interrupts belonging to the DMA channel map to the corresponding channel interrupt vector. In general, each channel of the DMA has a dedicated vectored interrupt. Therefore, the ISR for each channel should internally check for the status of the previously mentioned interrupts to respond to the corresponding events.

DMA Interrupt Control and Management

Each DMA channel has a vectored interrupt. However, each DMA channel internally generates various interrupts, which will invoke the same channel interrupt based on various events. This feature helps to enable/disable them and is also used to obtain the interrupt status.

Example:

```
// Enable the channel-3 source done interrupt
PLIB_DMA_ChannelXINTSourceEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_INT_SOURCE_DONE );
// Get the status of address error on channel-3
if(PLIB_DMA_ChannelXINTSourceFlagGet ( DMA_ID_0, DMA_CHANNEL_3, DMA_INT_ADDRESS_ERROR ))
{
    // application code for error handling
}
// Disable the channel abort interrupt
PLIB_DMA_ChannelXINTSourceDisable (DMA_ID_0, DMA_CHANNEL_3, DMA_INT_TRANSFER_ABORT);
```

5.6.6.4.3.5 Operating/Addressing Mode Management

DMA Operating Modes

The various operating/transfer modes supported are:

- One-Shot mode

- Repeated One-Shot mode
- Continuous mode
- Repeated Continuous mode
- Ping-Pong mode
- Null Data Write mode

One-Shot Mode

In One-Shot mode, a single transfer is performed for each trigger event. The trigger event can be synchronous (refer to [Transfer/Abort \(Synchronous\)](#) or asynchronous (refer to [Transfer/Abort \(Asynchronous\) Trigger Management](#)). When a single one-shot transfer occurs and the DMA count is decremented to zero, this disables the channel and requires the channel to be re-enabled to perform the next transaction.

Repeated One-Shot Mode

In Repeated One-Shot mode, single transfers occurs repeatedly so long as triggers are being provided. When the DMA count reaches zero, the channel is not disabled as in One-Shot mode, but the count is reloaded and transfers begin again.

Continuous Mode

In Continuous mode, a single trigger starts a sequence of back-to-back transfers. These continue with each transfer decrementing the DMA count until it reaches zero. At this point, and like one-shot mode, the channel is disabled.

Repeated Continuous Mode

This mode can be seen as a combination of Continuous and Repeated One-Shot modes. Data transfers keep occurring so as long as triggers are provided and multiple transfers can occur with each trigger. When the DMA count reaches zero, the address and data registers are reloaded and the process is repeated.

Ping-Pong Mode

Certain devices support Ping-Pong mode, which allows the CPU to process one buffer while a second buffer operates with the DMA channel. The net result is that the CPU has the entire DMA block transfer time to process the buffer that is currently not being used by the DMA channel.

The following figure demonstrates the data flow for four DMA transactions using Ping-Pong mode.

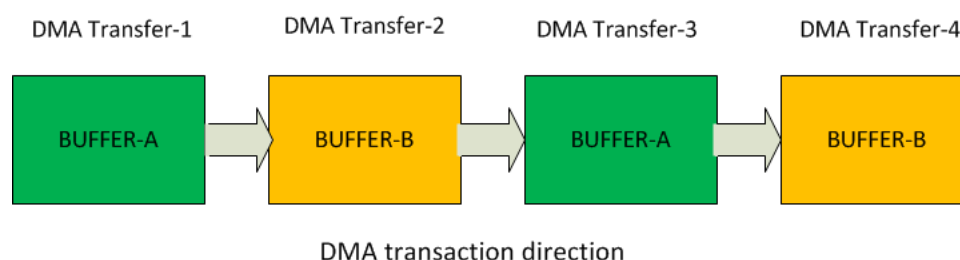


Figure: Ping-Pong Mode

When the DMA is filling buffer-B, the CPU can process buffer-A and vice-versa.

Ping-Pong mode can be combined with One-Shot and Continuous modes (previously discussed) to create other combinations.

Null Data Write Mode

A DMA transfer operates only in one direction from the source address to the destination address. However, some communication protocols require symmetrical buffer accesses, meaning that for every read operation performed on a buffer, there must be an accompanying write operation. The Null Write mode is designed to satisfy this requirement. This mode works

by transferring data from the source to the destination like any other DMA operation. Once this is done, the transferred data still with the DMA buffer is written back to the source.

DMA Addressing Modes

The addressing modes can be broadly classified into the following categories:

- Register Indirect with Post Increment Addressing mode
- Register Indirect without Post Increment Addressing mode
- Source/destination increment based on size
- Source/destination decrement based on size
- Peripheral Indirect Addressing mode

Register Indirect With Post Increment Addressing Mode

In this mode, the source data address is incremented by one location after each DMA transaction. The starting address is provided by a start address offset register.

Register Indirect Without Post Increment Addressing Mode

In this mode, the source data address is not updated after each DMA transaction. Therefore, the next DMA transfer is initiated from the same RAM address. The starting address is provided by a start address offset register.

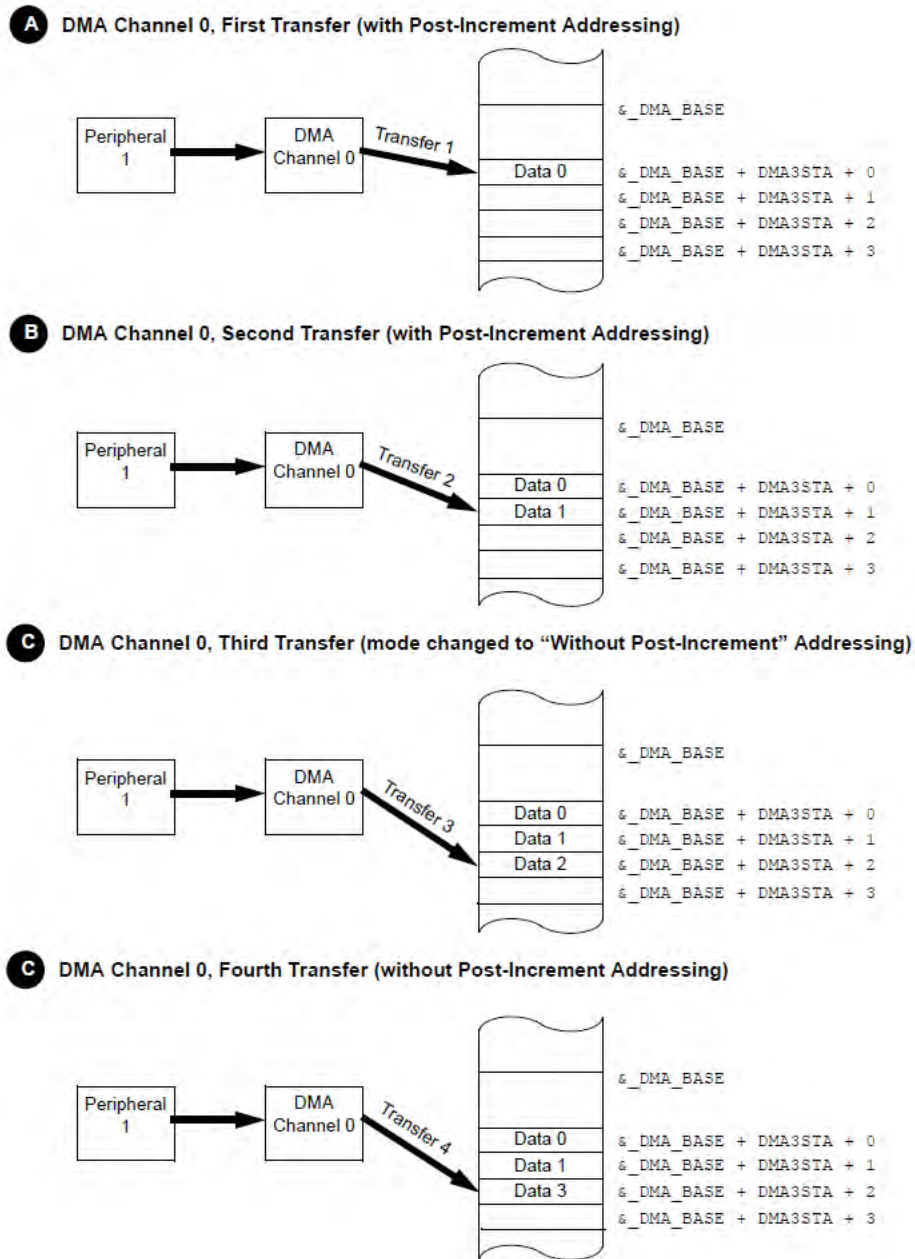


Figure: Register Indirect Addressing with and without post-increment.

Source/Destination Increment Based on Size

After each DMA transaction, the source/destination (a separate configuration exist for each) address increments by the size of data (word/byte) configured.

Source/Destination Decrement Based on Size

After each DMA transaction, the source/destination (a separate configuration exist for each) address decrements by the size of data (word/byte) configured.

Peripheral Indirect Addressing Mode

Peripheral Indirect Addressing (PIA) is a special auto-incrementing mode for the transfer of data to and from a multi-level peripheral buffer. This mode is only available for specific peripherals designed with its use in mind. Refer to the specific device data sheet to determine availability of this mode.

PIA-capable peripherals - When selected, the PIA-enabled peripheral generates a short Indirect Address (IA) (size defined by the peripheral) to the DMA channel. The IA is logically ORed with either the contents of the source address or the destination address to define a specific address for the peripheral inside the DMA address space.

DMA Operating/Transfer Modes:

Refer to Operating/Addressing Mode Management for descriptions of the available operating modes.

The following code demonstrates usage of Continuous mode.

Example:

```
// Enable the DMA
PLIB_DMA_Enable ( DMA_ID_0 );
// disable the selected channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3);
// program the source and destination addresses
// Set the source starting address to 0x1000
PLIB_DMA_ChannelXSourceStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000 );
// Set the destination starting address to 0x2000
PLIB_DMA_ChannelXDestinationStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000 );
// Set the transfer count to 100.
PLIB_DMA_ChannelXTransferCountSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// set the data transfer mode to one-shot
PLIB_DMA_ChannelXOperatingTransferModeSelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_MODE_CONTINUOUS );
// select the source addressing mode
PLIB_DMA_ChannelXSourceAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE );
// select the destination addressing mode
PLIB_DMA_ChannelXDestinationAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE );
// Enable the channel
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Initiate the transfer
PLIB_DMA_StartTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
```

DMA Addressing Modes:

DMA addressing modes can be selected by using following code snippet.

DMA Addressing mode Example:

```
// set the source address to increment upon each transfer
PLIB_DMA_ChannelXSourceAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE );
// Set the destination address to decrement upon each transfer
PLIB_DMA_ChannelXDestinationAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_DESTINATION_DECREMENT_BASED_ON_SIZE );
```

5.6.6.4.3.6 Channel Management

Each Channel in the DMA module is can be independently configured for a specific transaction with different properties.

The following aspects of the channel can be configured using this feature.

DMA Channel Enable/Disable

Each channel can be enabled or disabled to control the transactions on the channel. A DMA transfer can only occur when the

channel is enabled. A DMA channel can be enabled using the [PLIB_DMA_ChannelXEnable Function](#). When a channel is disabled by calling the [PLIB_DMA_ChannelXDisable Function](#) function, no data transfers can occur on the channel.

DMA Channel Priority

The DMA module has a natural priority associated with each of the channels. Channel 0 has the highest priority. Each DMA channel can be assigned priority. This priority scheme changes depending on the device selected. There are three ways in which priorities are assigned in PIC microcontrollers. In the first method, two bits are allocated for each channel for priority. Therefore, priorities 0 to 3 can be assigned to channels. The second is called round-robin. In this method, the module starts by giving Channel 0 preference in any data transfer conflicts. For each successive transfer conflict, the next higher channel receives preference, continuing as a cycle through all the channels. The third method is called fixed scheme, which always gives priority to the lowest requesting channel number. The channel priority can be set using the [PLIB_DMA_ChannelPrioritySelect Function](#) or the [PLIB_DMA_ChannelXPrioritySelect Function](#) depending on the device selected.

DMA Channel Chaining

Channel chaining is an enhancement to the DMA channel operation. A channel (slave channel) can be chained to an adjacent channel (master channel). The slave channel will be enabled when a block transfer of the master channel completes. At this point, any event on the slave channel will initiate a DMA transfer. If the channel has an event pending, a DMA transfer will begin immediately. The channel chaining can be enabled or disabled using the [PLIB_DMA_ChannelXChainEnable Function](#) and the [PLIB_DMA_ChannelXChainDisable Function](#), respectively. The channel chaining higher/lower channel number can be configured using the [PLIB_DMA_ChannelXChainToHigher Function](#) and the [PLIB_DMA_ChannelXChainToLower Function](#).

DMA Channel Transfer Direction

Each DMA transfer will have a direction. It can be memory to peripheral, peripheral to memory, or memory to memory. This can be set using the [PLIB_DMA_ChannelXTransferDirectionSelect Function](#).

DMA Channel Pattern Data

On supported devices, a DMA transfer can be aborted by a pattern match (refer to [Transfer/Abort \(Asynchronous\) Trigger Management](#)). The pattern data is per channel basis. It can be set using the [PLIB_DMA_ChannelXPatternDataSet Function](#).

Channel Auto Enable

This feature can be used to keep the channel active even if a block transfer or pattern match occurs. This mode is useful for applications that do repeated pattern matching. To enable this feature use the [PLIB_DMA_ChannelXAutoEnable Function](#). To disable this feature use the [PLIB_DMA_ChannelXAutoDisable Function](#).

DMA Channel Configuration

The following example configures DMA Channel 3 by setting its priority to two, enabling the auto-enable feature (refer to Channel Management section), enabling the channel chain feature and chaining Channel 3 and Channel 2, and finally enabling Channel 3.

Example:

```
// Disable the channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// set the channel priority to 2
PLIB_DMA_ChannelXPrioritySelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_PRIORITY_2 );
// Configure the channel auto enable feature
PLIB_DMA_ChannelXAutoEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Enable the channel chain feature
PLIB_DMA_ChannelXChainEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// chain channel 3 to channel 2
PLIB_DMA_ChannelXChainToLower ( DMA_ID_0, DMA_CHANNEL_3 );
// Enable channel-3
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_3 );
```

In some devices it is required to select the DMA transfer direction before initiating DMA transfer. This can be done using the

following code.

Example:

```
// Disable channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// Set the data transfer direction from memory to peripheral
PLIB_DMA_ChannelXTransferDirectionSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL );
// we can also check the current channels transfer direction
// check the data transfer direction
if ( DMA_READ_FROM_PERIPHERAL_WRITE_TO_MEMORY == PLIB_DMA_ChannelXTransferDirectionGet (
DMA_ID_0, DMA_CHANNEL_3 ) )
{

// application code

}
```

5.6.6.4.3.7 Source/Destination/Peripheral Address Management

This topic describes configuring the source/destination/peripheral addresses involved in the DMA transfer.

Source/Destination Start Address

Before initiating a DMA transfer, the channel should be programmed with the source and the destination addresses. In supported devices, this can be done using the [PLIB_DMA_ChannelXSourceStartAddressSet Function](#) and the [PLIB_DMA_ChannelXDestinationStartAddressSet Function](#), respectively. At any point, the programmed addresses can be read using the [PLIB_DMA_ChannelXSourceStartAddressGet Function](#) and the [PLIB_DMA_ChannelXDestinationStartAddressGet Function](#).

Source/Destination Start Address Offset

In some devices, a dedicated region of RAM is allocated for DMA transfers typically called DMA RAM. For any DMA transfer, the source or destination address needs to be defined as an offset from the starting of the DMA RAM for such devices. Use the [PLIB_DMA_ChannelXStartAddressOffsetSet Function](#) to set the offset addresses of buffers A and B. Buffers A and B are used in Ping-Pong mode. To learn more, refer to [Operating/Addressing Mode Management](#).

Peripheral Address

When DMA is used for a peripheral to memory or a memory to peripheral transfer, the peripheral address within a register must be set. This register needs to be programmed with the peripheral location where data transfers takes place. For example this can be the I²C buffer Special Function Register (SFR) location or the UART Receive register, etc. To set the peripheral address use the [PLIB_DMA_ChannelXPeripheralAddressSet Function](#).

DMA Source/Destination/Peripheral Address Control

The first step in initiating a DMA transfer is to set the source and destination addresses.

The following example sets the source address and the destination address for Channel 3.

Example:

```
// Set the source starting address to 0x1000
PLIB_DMA_ChannelXSourceStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000 );
// Set the destination starting address to 0x2000
PLIB_DMA_ChannelXDestinationStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000 );
```

On certain devices, the source starting address needs to be given as an offset from the start of the DMA RAM address (in such devices, only memory to peripheral transfers and vice-versa are possible). In such devices, the source start address is set using the [PLIB_DMA_ChannelXStartAddressOffsetSet](#) function, as shown in the following code.

Example:

```
// Set the offset-A register to 0x1000
PLIB_DMA_ChannelXStartAddressOffsetSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000,
DMA_ADDRESS_OFFSET_PRIMARY);
// If ping-pong mode is enabled we will make use of buffer-B, and in that case
// we need to set an address for that also
// Set the offset-B to 0x2000
PLIB_DMA_ChannelXStartAddressOffsetSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000,
DMA_ADDRESS_OFFSET_SECONDARY);
```

The following example sets the peripheral address when the DMA needs to be initiated between memory and peripheral.

DMA Peripheral Address Set Example:

```
// Set the peripheral address to ADC1 buffer for channel-3 transfers.
PLIB_DMA_ChannelXPeripheralAddressSet ( DMA_ID_0, DMA_CHANNEL_3, (volatile
uint16_t)(&ADC1BUF0));
```

There are also equivalent functions available within this library that read the source/destination/peripheral address set.

5.6.6.4.3.8 Source/Destination/Peripheral Data Management

This topic describes the data configuration.

Source/Destination Transfer Size Configuration

In supported devices, each channel can be programmed to read a particular sized data from a source, and write in a particular size to a destination. Both the source and destination sizes are configurable. Use the [PLIB_DMA_ChannelXSourceSizeSet Function](#) and the [PLIB_DMA_ChannelXDestinationSizeSet Function](#) for read size and write size configuration, respectively. For example, the source size can be 200 bytes and the destination size can be 100 bytes. However, it is logical to have both set to the same value. In some devices, the amount of data that needs to be transferred is independent of the source and destination, and is called transfer count. In such devices use the [PLIB_DMA_ChannelXTransferCountSet Function](#) to set the total amount of data (block transfer data).

Cell Size Configuration

In supported devices, when DMA transfer is initiated, the amount of data configured for a cell transfer will be delivered to the destination per trigger. This can be configured using the [PLIB_DMA_ChannelXCellSizeSet Function](#).

Channel Data Size

The DMA controller can handle both byte and word (16-bit) transactions. Each DMA channel is individually configurable for the data size using the [PLIB_DMA_ChannelXDataSizeSelect Function](#).

Source/Destination Pointer Read

In supported devices, the application can read the byte in source/destination to which the DMA core is currently pointing. This can be done using the [PLIB_DMA_ChannelXSourcePointer Function](#) and the [PLIB_DMA_ChannelXDestinationPointer Function](#) for source and destination, respectively.

DMA Source/Destination/Peripheral Data Control

After setting the source/destination address the amount of data that needs to be transferred is set in a DMA operation.

However, prior to that the data transaction type (byte/word) needs to be configured on certain devices.

Example:

```
// Set channel-3 transactions as byte transactions
PLIB_DMA_ChannelXDataSizeSelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_DATA_8 );
// Set channel-3 transactions as 16-bit transactions
```

```

PLIB_DMA_ChannelXDataSizeSelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_DATA_16 );
// Get the current data size configured
if( DMA_CHANNEL_BYTE == PLIB_DMA_ChannelXDataSizeGet ( DMA_ID_0, DMA_CHANNEL_3 ))
{
    // application code
}

```

Example:

```

// Set the transfer count to 100.
PLIB_DMA_ChannelXTransferCountSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );

```

Some high-end microcontrollers (especially PIC32 devices) use the method of cell transfer and block transfer. A cell transfer is the amount of data a DMA channel transfers per an event. In such microcontrollers the application can configure the sizes of the source and destination (per channel basis). A block transfer is defined as the number of bytes transferred when a channel is enabled. A block transfer is comprised of several cell transfers. For example, if the source and destination size is 100 and the cell size is 10, it will take 10 cell transfers to complete the block transfer, which is 100 bytes (larger of source or destination sizes).

DMA cell transfer configuration Example:

```

// Set the source size to be 100
PLIB_DMA_ChannelXSourceSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// Set the destination size to be 100
PLIB_DMA_ChannelXDestinationSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// Set the cell size to be 10
PLIB_DMA_ChannelXCellSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 10 );

```

5.6.6.4.3.9 Special Function Modules (CRC)

Selected high-end devices support a special feature called the CRC generator.

Depending on the device, there is a highly configurable, 16-bit or 32-bit CRC generator. The CRC can be assigned to any available DMA channel using [PLIB_DMA_CRCChannelSelect Function](#). The CRC is enabled using the [PLIB_DMA_CRCEnable Function](#).

Optionally, the data from the source can be subjected to byte reordering using the [PLIB_DMA_CRCByteOrderSelect Function](#). The data is then optionally passed to the Linear Shift Feedback Register (LFSR) CRC or the IP header checksum blocks using the [PLIB_DMA_CRCTypeSet Function](#).

The CRC feature is attached to a channel in one of two possible modes, Background or Append.

Background Mode

The CRC is calculated in the background, with normal DMA behavior maintained. This can be selected by disabling the Append mode.

Append Mode

Data read from the source is not written to the destination, but the CRC data is accumulated in the CRC data register. The accumulated CRC is written to the location specified by the destination address when a block transfer completes. This mode is enabled by the [PLIB_DMA_CRCAppendModeEnable Function](#).

The following code shows DMA CRC calculation in Append mode.

Example:

```

// CRC of the Flash block
uint32_t blockCRC;
bool error = false;
// disable the channel 0 interrupts using the Interrupts Peripheral Library
// Enable the DMA controller
PLIB_DMA_Enable ( DMA_ID_0 );
// disable the selected channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );

```

```

// set the channel priority to 2
PLIB_DMA_ChannelXPrioritySelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_PRIORITY_2 );
// Configure the channel auto enable feature
PLIB_DMA_ChannelXAutoEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Disable the channel chain feature
PLIB_DMA_ChannelXChainDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// Disable the start IRQ.
PLIB_DMA_ChannelXTriggerDisable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_TRANSFER_START );
// Seed the CRC generator
PLIB_DMA_CRCDataWrite ( DMA_ID_0, 0xFFFF );
// use the standard CCITT CRC 16 polynomial :  $X^{16}+X^{12}+X^5+1$ 
PLIB_DMA_CRCXOREnableSet ( DMA_ID_0, 0x1021 );
// set polynomial length to 16
PLIB_DMA_CRCPolynomialLengthSet ( DMA_ID_0, 16 );
// Enable append mode
PLIB_DMA_CRCAppendModeEnable ( DMA_ID_0 );
// Attach CRC to DMA channel-3
PLIB_DMA_CRCChannelSelect ( DMA_ID_0, DMA_CHANNEL_3 );
// Enable the CRC
PLIB_DMA_CRCEnable ( DMA_ID_0 );
// Set the source starting address to 0x1000
PLIB_DMA_ChannelXSourceStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000 );
// Set the destination starting address to 0x2000
PLIB_DMA_ChannelXDestinationStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000 );
// set source and destination size to 100
PLIB_DMA_ChannelXSourceSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
PLIB_DMA_ChannelXDestinationSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// Set cell size to 100
PLIB_DMA_ChannelXCellSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
//Enable channel-3
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Force initiate a transfer
PLIB_DMA_StartTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );

```

5.6.6.4.3.10 Status (including Channel)

This module gives application access to various DMA and channel status information.

5.6.6.5 Configuring the Library

The library is configured for the supported DMA Peripheral Library.

5.6.6.6 Library Interface

Data Types and Constants

	Name	Description
	DMA_ADDRESS_OFFSET_TYPE	Lists the possible DMA address offsets.
	DMA_CHANNEL	Lists the possible DMA channels available.
	DMA_CHANNEL_ADDRESSING_MODE	Lists the possible channel addressing modes.
	DMA_CHANNEL_COLLISION	Lists the possible channel collision types.
	DMA_CHANNEL_DATA_SIZE	Lists the possible data sizes for the DMA channel.
	DMA_CHANNEL_PRIORITY	Lists the possible channel priorities.
	DMA_CHANNEL_TRANSFER_DIRECTION	Lists the possible data transfer directions.
	DMA_CHANNEL_TRIGGER_TYPE	Lists the possible DMA channel triggers.
	DMA_CRC_BIT_ORDER	Lists the possible CRC calculation bit orders.

	DMA_CRC_BYTE_ORDER	Lists the possible byte orders.
	DMA_CRC_TYPE	Lists the possible checksums.
	DMA_DESTINATION_ADDRESSING_MODE	Lists the possible destination addressing modes.
	DMA_INT_TYPE	Lists the possible Interrupt types for a particular channel-X
	DMA_MODULE_ID	Possible instances of the DMA module.
	DMA_PATTERN_LENGTH	Gives the DMA pattern length
	DMA_PING_PONG_MODE	Lists the possible ping-pong modes.
	DMA_SOURCE_ADDRESSING_MODE	Lists the possible source addressing modes.
	DMA_TRANSFER_MODE	Lists the possible DMA transfer/operating modes.
	DMA_TRIGGER_SOURCE	Lists the possible DMA trigger sources.

DMA Channel Configuration

	Name	Description
	PLIB_DMA_ChannelPrioritySelect	Sets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelPriorityGet	Gets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelXAutoDisable	Channel is disabled after a block transfer is complete.
	PLIB_DMA_ChannelXAutoEnable	Channel is continuously enabled.
	PLIB_DMA_ChannelXBusyActiveSet	Sets the Busy bit to active.
	PLIB_DMA_ChannelXBusyInActiveSet	Sets the Busy bit to inactive.
	PLIB_DMA_ChannelXChainDisable	Disables the channel chaining for the specified DMA channel.
	PLIB_DMA_ChannelXChainEnable	Channel chain feature is enabled.
	PLIB_DMA_ChannelXChainToHigher	Chains the specified channel to a channel higher in natural priority.
	PLIB_DMA_ChannelXChainToLower	Chains the specified channel to a channel lower in natural priority.
	PLIB_DMA_ChannelXDisable	Disable the specified channel.
	PLIB_DMA_ChannelXEnable	Enable the specified channel.
	PLIB_DMA_ChannelXPriorityGet	Gets the priority of the specified channel.
	PLIB_DMA_ChannelXPrioritySelect	Sets the priority of the specified channel.
	PLIB_DMA_ChannelXTransferDirectionGet	Returns the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXTransferDirectionSelect	Selects the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXDisabledDisablesEvents	Channel start/abort events will be ignored even if the channel is disabled.
	PLIB_DMA_ChannelXDisabledEnablesEvents	Channel start/abort events will be registered even if the channel is disabled.
	PLIB_DMA_ChannelXPatternIgnoreByteDisable	Disables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteEnable	Enables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled	Returns the state of the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreGet	Returns the pattern match ignore value.
	PLIB_DMA_ChannelXPatternIgnoreSet	Sets the pattern match ignore value.
	PLIB_DMA_ChannelXPatternLengthGet	Returns the pattern match length.
	PLIB_DMA_ChannelXPatternLengthSet	Sets the pattern match length.

DMA Channel Status Routines

	Name	Description
	PLIB_DMA_ChannelXAutoIsEnabled	Returns the channel automatic enable status.
	PLIB_DMA_ChannelXBufferedDataIsWritten	Returns the buffered data write status for the specified channel.

	PLIB_DMA_ChannelXBusyIsBusy	Returns the busy status of the specified channel.
	PLIB_DMA_ChannelXChainIsEnabled	Returns the chain status of the specified channel.
	PLIB_DMA_ChannelXCollisionStatus	Returns the status of the specified collision type for the specified channel.
	PLIB_DMA_ChannelXEventIsDetected	Returns the event status on the specified channel.
	PLIB_DMA_ChannelXIIsEnabled	Return the enable status of the specified channel.
	PLIB_DMA_ChannelXNullWriteModelsEnabled	Returns the enable status of the Null Write mode for the specified channel.
	PLIB_DMA_ChannelXPingPongModeGet	Returns the Ping-Pong mode status for the specified channel.
	PLIB_DMA_ChannelBitsGet	Returns the DMA channel bits.

DMA CRC Module Interface Routines

	Name	Description
	PLIB_DMA_CRCAppendModeDisable	Disables the CRC append mode.
	PLIB_DMA_CRCAppendModeEnable	Enables the CRC append mode.
	PLIB_DMA_CRCAppendModelsEnabled	Gets the enable status of the CRC append mode.
	PLIB_DMA_CRCBitOrderSelect	Selects the bit order for checksum calculation.
	PLIB_DMA_CRCByteOrderGet	Gets the current byte order selected by the DMA module CRC feature.
	PLIB_DMA_CRCByteOrderSelect	Selects the byte order.
	PLIB_DMA_CRCChannelGet	Returns the current DMA channel to which the CRC is assigned.
	PLIB_DMA_CRCChannelSelect	Assigns the CRC to the specified DMA channel.
	PLIB_DMA_CRCDataRead	Reads the contents of the DMA CRC data register.
	PLIB_DMA_CRCDataWrite	Writes the contents of the DMA CRC data register with the specified data.
	PLIB_DMA_CRCDisable	Disables the DMA module CRC feature.
	PLIB_DMA_CRCEnable	Enables the DMA module CRC feature.
	PLIB_DMA_CRCPolynomialLengthGet	Gets the current polynomial length.
	PLIB_DMA_CRCPolynomialLengthSet	Selects the polynomial length.
	PLIB_DMA_CRCTypeGet	Gets the current DMA module CRC feature type.
	PLIB_DMA_CRCTypeSet	Selects the DMA module CRC feature type.
	PLIB_DMA_CRCWriteByteOrderAlter	The source data is written to the destination reordered as defined by the BYTO<1:0> bits.
	PLIB_DMA_CRCWriteByteOrderMaintain	The source data is written to the destination unaltered.
	PLIB_DMA_CRCXOREnableGet	Reads the CRC XOR register.
	PLIB_DMA_CRCXOREnableSet	Writes to the CRC XOR enable register as per the specified enable mask.

DMA General Configuration Routines

	Name	Description
	PLIB_DMA_BusyActiveReset	Resets the BUSY bit of the DMA controller.
	PLIB_DMA_BusyActiveSet	Sets the BUSY bit of the DMA controller.
	PLIB_DMA_Disable	DMA module is disabled.
	PLIB_DMA_Enable	DMA module is enabled.
	PLIB_DMA_StopInIdleDisable	DMA transfers continue during Idle mode.
	PLIB_DMA_StopInIdleEnable	DMA transfers are halted during Idle mode.
	PLIB_DMA_SuspendDisable	DMA suspend is disabled and the DMA module operates normally.

	PLIB_DMA_SuspendEnable	DMA transfers are suspended to allow uninterrupted access by the CPU to the data bus.

DMA Global Status Routines

	Name	Description
	PLIB_DMA_CRCIsEnabled	Gets the enable status of the CRC feature.
	PLIB_DMA_IsBusy	Gets the BUSY bit of the DMA controller.
	PLIB_DMA_IsEnabled	Returns the DMA module enable status.
	PLIB_DMA_LastBusAccessIsRead	Returns true if the last DMA bus access was a read.
	PLIB_DMA_LastBusAccessIsWrite	Returns true if the last DMA bus access was a write.
	PLIB_DMA_LastChannelActive	Returns the last DMA channel that participated in the data transfer.
	PLIB_DMA_RecentAddressAccessed	Returns the address of the most recent DMA access.
	PLIB_DMA_SuspendIsEnabled	Returns the DMA suspend status.

DMA Interrupt Control and Management

	Name	Description
	PLIB_DMA_ChannelXINTSourceDisable	Disables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceEnable	Enables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagClear	Clears the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagGet	Returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagSet	Sets the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceIsEnabled	Returns the enable status of the specified interrupt source for the specified channel.

DMA Operating/Addressing Mode Configuration

	Name	Description
	PLIB_DMA_ChannelXAddressModeGet	Gets the channel address mode.
	PLIB_DMA_ChannelXAddressModeSelect	Sets the channel address mode.
	PLIB_DMA_ChannelXDestinationAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXDestinationAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXNullWriteModeDisable	Disables the Null Write mode.
	PLIB_DMA_ChannelXNullWriteModeEnable	Enables the Null Write mode.
	PLIB_DMA_ChannelXOperatingTransferModeGet	Returns the current transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXOperatingTransferModeSelect	Selects the transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXReloadDisable	Disables reloading of the address and count registers.
	PLIB_DMA_ChannelXReloadEnable	Enables reloading of the address and count registers.
	PLIB_DMA_ChannelXSourceAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXSourceAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXReloadIsEnabled	Returns the address and count registers reload enable status.

DMA Source/Destination/Peripheral Address Control Interface Routines

	Name	Description
≡	PLIB_DMA_ChannelXDestinationStartAddressGet	Reads the destination start address configured for the specified channel.
≡	PLIB_DMA_ChannelXDestinationStartAddressSet	Writes the specified destination start address into the register corresponding to the specified channel.
≡	PLIB_DMA_ChannelXPeripheralAddressGet	Gets the peripheral address configured for the specified channel.
≡	PLIB_DMA_ChannelXPeripheralAddressSet	Sets the peripheral address for the specified channel.
≡	PLIB_DMA_ChannelXSourceStartAddressGet	Reads the source start address configured for the specified channel.
≡	PLIB_DMA_ChannelXSourceStartAddressSet	Writes the specified source start address into the register corresponding to the specified channel.
≡	PLIB_DMA_ChannelXStartAddressOffsetGet	Gets the primary/secondary start address (DPSRAM) offset.
≡	PLIB_DMA_ChannelXStartAddressOffsetSet	Sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.

DMA Source/Destination/Peripheral Data Management

	Name	Description
≡	PLIB_DMA_ChannelXCellProgressPointerGet	Returns the number of bytes transferred since the last event.
≡	PLIB_DMA_ChannelXCellSizeGet	Reads the cell size (in bytes) configured for the specified channel.
≡	PLIB_DMA_ChannelXCellSizeSet	Writes the specified cell size into the register corresponding to the specified channel.
≡	PLIB_DMA_ChannelXDataSizeGet	Returns the current data size for the specified channel.
≡	PLIB_DMA_ChannelXDataSizeSelect	Selects the data size for the specified channel.
≡	PLIB_DMA_ChannelXDestinationPointerGet	Reads the current byte of the destination being pointed to for the specified channel.
≡	PLIB_DMA_ChannelXDestinationSizeGet	Reads the destination size configured for the specified channel.
≡	PLIB_DMA_ChannelXDestinationSizeSet	Writes the specified destination size into the register corresponding to the specified channel.
≡	PLIB_DMA_ChannelXPatternDataGet	Returns the pattern matching (for DMA abort) data programmed for the specified channel.
≡	PLIB_DMA_ChannelXPatternDataSet	Writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel.
≡	PLIB_DMA_ChannelXSourcePointerGet	Reads the current byte of the source being pointed to for the specified channel.
≡	PLIB_DMA_ChannelXSourceSizeGet	Reads the source size configured for the specified channel.
≡	PLIB_DMA_ChannelXSourceSizeSet	Writes the specified source size into the register corresponding to the specified channel.
≡	PLIB_DMA_ChannelXTransferCountGet	Gets the DMA data transfer count that is programmed for the specified channel.
≡	PLIB_DMA_ChannelXTransferCountSet	Sets the DMA data transfer count for the specified channel.

DMA Transfer/Abort (Asynchronous) Trigger Management

	Name	Description
≡	PLIB_DMA_ChannelXAbortIRQSet	Sets the IRQ to abort the DMA transfer on the specified channel.
≡	PLIB_DMA_ChannelXStartIRQSet	Sets the IRQ to initiate the DMA transfer on the specified channel.
≡	PLIB_DMA_ChannelXTriggerDisable	Disables the DMA transfer abort via a matching interrupt (specified by the IRQ).

	PLIB_DMA_ChannelXTriggerEnable	Enables the specified DMA channel trigger.
	PLIB_DMA_ChannelXTriggerIsEnabled	Returns the enable status of the specified DMA transfer/abort trigger.

DMA Transfer/Abort (Synchronous)

	Name	Description
	PLIB_DMA_AbortTransferSet	Aborts transfer on the specified channel.
	PLIB_DMA_StartTransferSet	Initiates transfer on the specified channel.

Feature Existence Routines

	Name	Description
	PLIB_DMA_ExistsAbortTransfer	Identifies whether the AbortTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsBusy	Identifies whether the Busy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelBits	Identifies whether the ChannelBits feature exists on the DMA module.
	PLIB_DMA_ExistsChannelX	Identifies whether the ChannelX feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAbortIRQ	Identifies whether the ChannelXAbortIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAuto	Identifies whether the ChannelXAuto feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXBusy	Identifies whether the ChannelXBusy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellProgressPointer	Identifies whether the ChannelXCellProgressPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellSize	Identifies whether the ChannelXCellSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChain	Identifies whether the ChannelXChain feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChainEnbl	Identifies whether the ChannelXChainEnbl feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationPointer	Identifies whether the ChannelXDestinationPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationSize	Identifies whether the ChannelXDestinationSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationStartAddress	Identifies whether the ChannelXDestinationStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDisabled	Identifies whether the ChannelXDisabled feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXEvent	Identifies whether the ChannelXEvent feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSource	Identifies whether the ChannelXINTSource feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSourceFlag	Identifies whether the ChannelXINTSourceFlag feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternData	Identifies whether the ChannelXPatternData feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternIgnore	Identifies whether the ChannelXPatternIgnore feature exists on the DMA module.

	<u>PLIB_DMA_ExistsChannelXPatternIgnoreByte</u>	Identifies whether the ChannelXPatternIgnoreByte feature exists on the DMA module.
	<u>PLIB_DMA_ExistsChannelXPatternLength</u>	Identifies whether the ChannelXPatternLength feature exists on the DMA module.
	<u>PLIB_DMA_ExistsChannelXPriority</u>	Identifies whether the ChannelXPriority feature exists on the DMA module.
	<u>PLIB_DMA_ExistsChannelXSourcePointer</u>	Identifies whether the ChannelXSourcePointer feature exists on the DMA module.
	<u>PLIB_DMA_ExistsChannelXSourceSize</u>	Identifies whether the ChannelXSourceSize feature exists on the DMA module.
	<u>PLIB_DMA_ExistsChannelXSourceStartAddress</u>	Identifies whether the ChannelXSourceStartAddress feature exists on the DMA module.
	<u>PLIB_DMA_ExistsChannelXStartIRQ</u>	Identifies whether the ChannelXStartIRQ feature exists on the DMA module.
	<u>PLIB_DMA_ExistsChannelXTrigger</u>	Identifies whether the ChannelXTrigger feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRC</u>	Identifies whether the CRC feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRCAppendMode</u>	Identifies whether the CRCAppendMode feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRCBitOrder</u>	Identifies whether the CRCBitOrder feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRCByteOrder</u>	Identifies whether the CRCByteOrder feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRCChannel</u>	Identifies whether the CRCChannel feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRCDData</u>	Identifies whether the CRCDData feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRCPolynomialLength</u>	Identifies whether the CRCPolynomialLength feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRCType</u>	Identifies whether the CRCType feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRCWriteByteOrder</u>	Identifies whether the CRCWriteByteOrder feature exists on the DMA module.
	<u>PLIB_DMA_ExistsCRCXOREnable</u>	Identifies whether the CRCXOREnable feature exists on the DMA module.
	<u>PLIB_DMA_ExistsEnableControl</u>	Identifies whether the EnableControl feature exists on the DMA module.
	<u>PLIB_DMA_ExistsLastBusAccess</u>	Identifies whether the LastBusAccess feature exists on the DMA module.
	<u>PLIB_DMA_ExistsRecentAddress</u>	Identifies whether the RecentAddress feature exists on the DMA module.
	<u>PLIB_DMA_ExistsStartTransfer</u>	Identifies whether the StartTransfer feature exists on the DMA module.
	<u>PLIB_DMA_ExistsStopInIdle</u>	Identifies whether the StopInIdle feature exists on the DMA module.
	<u>PLIB_DMA_ExistsSuspend</u>	Identifies whether the Suspend feature exists on the DMA module.

Description

This topic describes the functions of the DMA Peripheral Library.

Refer to each topic for a detailed description.

5.6.6.6.1 DMA General Configuration Routines

5.6.6.6.1.1 PLIB_DMA_BusyActiveReset Function

C

```
void PLIB_DMA_BusyActiveReset(  
    DMA_MODULE_ID index  
);
```

Description

This function resets the BUSY bit of the DMA controller. The DMA module is disabled and is not actively transferring data.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the Busy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsBusy](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_BusyActiveReset( DMA_ID_0 );
```

5.6.6.6.1.2 PLIB_DMA_BusyActiveSet Function

C

```
void PLIB_DMA_BusyActiveSet(  
    DMA_MODULE_ID index  
);
```

Description

This function sets the BUSY bit of the DMA controller. The DMA module is active.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the Busy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsBusy](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_BusyActiveSet( DMA_ID_0 );
```

5.6.6.6.1.3 PLIB_DMA_Disable Function

C

```
void PLIB_DMA_Disable(  
    DMA_MODULE_ID index  
);
```

Description

This function disables the DMA module.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsEnableControl](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_Disable( DMA_ID_0 );
```

5.6.6.6.1.4 PLIB_DMA_Enable Function

C

```
void PLIB_DMA_Enable(  
    DMA_MODULE_ID index  
);
```

Description

This function enables the DMA module.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsEnableControl](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_Enable( DMA_ID_0 );
```

5.6.6.6.1.5 PLIB_DMA_StopInIdleDisable Function

C

```
void PLIB_DMA_StopInIdleDisable(  
    DMA_MODULE_ID index  
);
```

Description

This function causes DMA transfers to continue during Idle mode.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsStopInIdle](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_StopInIdleDisable( DMA_ID_0 );
```

5.6.6.6.1.6 PLIB_DMA_StopInIdleEnable Function**C**

```
void PLIB_DMA_StopInIdleEnable(  
    DMA_MODULE_ID index  
);
```

Description

This function halts DMA transfers during Idle mode.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsStopInIdle](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_StopInIdleEnable( DMA_ID_0 );
```

5.6.6.6.1.7 PLIB_DMA_SuspendDisable Function**C**

```
void PLIB_DMA_SuspendDisable(  
    DMA_MODULE_ID index  
);
```

Description

This function disables the DMA suspend. The DMA module continues to operate normally.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the Suspend feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsSuspend](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_SuspendDisable( DMA_ID_0 );
```

5.6.6.1.8 PLIB_DMA_SuspendEnable Function**C**

```
void PLIB_DMA_SuspendEnable(  
    DMA_MODULE_ID index  
);
```

Description

This function suspends the DMA transfers to allow uninterrupted access by the CPU to the data bus.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the Suspend feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsSuspend](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_SuspendEnable( DMA_ID_0 );
```

5.6.6.6.2 DMA Transfer/Abort (Synchronous)**5.6.6.6.2.1 PLIB_DMA_AbortTransferSet Function****C**

```
void PLIB_DMA_AbortTransferSet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function aborts transfer on the specified channel. This is a forced abort controlled via software.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the AbortTransfer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsAbortTransfer](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_AbortTransferSet (    DMA_ID_0,  
                             DMA_CHANNEL_4 );
```

5.6.6.6.2 PLIB_DMA_StartTransferSet Function**C**

```
void PLIB_DMA_StartTransferSet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function initiates transfer on the specified channel. This is a forced transfer controlled via software.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the StartTransfer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsStartTransfer](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_StartTransferSet (    DMA_ID_0,  
                             DMA_CHANNEL_4 );
```

5.6.6.6.3 DMA Transfer/Abort (Asynchronous) Trigger Management**5.6.6.6.3.1 PLIB_DMA_ChannelXAbortIRQSet Function****C**

```
void PLIB_DMA_ChannelXAbortIRQSet(  
    DMA_MODULE_ID index,
```

```
DMA_CHANNEL channel,  
DMA_TRIGGER_SOURCE IRQ  
);
```

Description

This function sets the IRQ to abort the DMA transfer on the specified channel. (IRQ to start the channel transfer.)

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
IRQnum	The IRQ number of the trigger source of type DMA_TRIGGER_SOURCE

Returns

None.

Remarks

This function implements an operation of the ChannelXAbortIRQ feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXAbortIRQ](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_TRIGGER_SOURCE irq = DMA_TRIGGER_TIMER_CORE;  
PLIB_DMA_ChannelXAbortIRQSet ( DMA_ID_0,  
                                DMA_CHANNEL_4,  
                                irq );
```

5.6.6.6.3.2 PLIB_DMA_ChannelXStartIRQSet Function

C

```
void PLIB_DMA_ChannelXStartIRQSet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel,  
    DMA_TRIGGER_SOURCE IRQnum  
);
```

Description

This function sets the IRQ to initiate the DMA transfer on the specified channel. (IRQ to start the channel transfer.)

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
IRQnum	The IRQ number of the trigger source of type DMA_TRIGGER_SOURCE

Returns

None.

Remarks

This function implements an operation of the ChannelXStartIRQ feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXStartIRQ](#) function in your application to automatically determine whether this feature is available.

Example

```
//This code is for PIC24F family
DMA_TRIGGER_SOURCE irq = DMA_TRIGGER_OUTPUT_COMPARE_1;
PLIB_DMA_ChannelXStartIRQSet ( DMA_ID_0,
                                DMA_CHANNEL_4,
                                irq );
```

5.6.6.6.3.3 PLIB_DMA_ChannelXTriggerDisable Function**C**

```
void PLIB_DMA_ChannelXTriggerDisable(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel,
    DMA_CHANNEL_TRIGGER_TYPE trigger
);
```

Description

This function disables the DMA transfer abort via a matching interrupt (specified by the IRQ). The interrupt number IRQ is ignored and does not terminate a transfer.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
trigger	Type of trigger (transfer start/abort/pattern match abort)

Returns

None.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXTrigger](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXTriggerDisable ( DMA_ID_0,
                                   DMA_CHANNEL_4,
                                   DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT );
```

5.6.6.6.3.4 PLIB_DMA_ChannelXTriggerEnable Function**C**

```
void PLIB_DMA_ChannelXTriggerEnable(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel,
    DMA_CHANNEL_TRIGGER_TYPE trigger
);
```

Description

This function enables the the specified DMA channel trigger.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
trigger	Type of trigger (transfer start/abort/pattern match abort)

Returns

None.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXTrigger](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXTriggerEnable (    DMA_ID_0,
                                   DMA_CHANNEL_4,
                                   DMA_CHANNEL_TRIGGER_TRANSFER_ABORT );
```

5.6.6.6.3.5 PLIB_DMA_ChannelXTriggerIsEnabled Function**C**

```
bool PLIB_DMA_ChannelXTriggerIsEnabled(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel,
    DMA_CHANNEL_TRIGGER_TYPE trigger
);
```

Description

This function returns the enable status of the specified DMA transfer/abort trigger.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
trigger	Type of trigger (transfer start/abort/pattern match abort)

Returns

- true - The specified trigger is enabled
- false - The specified trigger is disabled

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXTrigger](#) function in your application to automatically determine whether this feature is available.

Example

```
bool startTriggerstatus;
startTriggerstatus = PLIB_DMA_ChannelXTriggerIsEnabled (
    DMA_ID_0,
    DMA_CHANNEL_4,
    DMA_CHANNEL_TRIGGER_TRANSFER_START );
```

5.6.6.6.4 DMA Interrupt Control and Management

5.6.6.6.4.1 PLIB_DMA_ChannelXINTSourceDisable Function

C

```
void PLIB_DMA_ChannelXINTSourceDisable(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL dmaChannel,  
    DMA_INT_TYPE dmaINTSource  
);
```

Description

This function disables the specified interrupt source for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Returns

None.

Remarks

This function implements an operation of the ChannelXINTSource feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXINTSource](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;  
PLIB_DMA_ChannelXINTSourceDisable ( DMA_ID_0,  
                                     spiDMAChannel,  
                                     DMA_INT_ADDRESS_ERROR );
```

5.6.6.6.4.2 PLIB_DMA_ChannelXINTSourceEnable Function

C

```
void PLIB_DMA_ChannelXINTSourceEnable(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL dmaChannel,  
    DMA_INT_TYPE dmaINTSource  
);
```

Description

This function enables the specified interrupt source for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Returns

None.

Remarks

This function implements an operation of the ChannelXINTSource feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXINTSource](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceEnable ( DMA_ID_0,
                                   spiDMAChannel,
                                   DMA_INT_ADDRESS_ERROR );
```

5.6.6.6.4.3 PLIB_DMA_ChannelXINTSourceFlagClear Function

C

```
void PLIB_DMA_ChannelXINTSourceFlagClear(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
    DMA_INT_TYPE dmaINTSource
);
```

Description

This function clears the interrupt flag of the specified DMA interrupt source for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Returns

None.

Remarks

This function implements an operation of the ChannelXINTSourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXINTSourceFlag](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceFlagClear ( DMA_ID_0,
                                       spiDMAChannel,
                                       DMA_INT_ADDRESS_ERROR );
```

5.6.6.6.4.4 PLIB_DMA_ChannelXINTSourceFlagGet Function

C

```
bool PLIB_DMA_ChannelXINTSourceFlagGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
```

```
    DMA_INT_TYPE dmaINTSource  
);
```

Description

This function returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Returns

- true - The interrupt flag is set
- false - The interrupt flag is not set

Remarks

This function implements an operation of the ChannelXINTSourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXINTSourceFlag](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;  
bool AddressErrorINTStatus;  
AddressErrorINTStatus = PLIB_DMA_ChannelXINTSourceFlagGet (  
    DMA_ID_0,  
    spiDMAChannel,  
    DMA_INT_ADDRESS_ERROR );
```

5.6.6.6.4.5 PLIB_DMA_ChannelXINTSourceFlagSet Function

C

```
void PLIB_DMA_ChannelXINTSourceFlagSet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL dmaChannel,  
    DMA_INT_TYPE dmaINTSource  
);
```

Description

This function sets the interrupt flag of the specified DMA interrupt source for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Returns

None.

Remarks

This function implements an operation of the ChannelXINTSourceFlag feature. This feature may not be available on all devices.

Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXINTSourceFlag](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceFlagSet ( DMA_ID_0,
                                     spiDMAChannel,
                                     DMA_INT_ADDRESS_ERROR );
```

5.6.6.6.4 PLIB_DMA_ChannelXINTSourceIsEnabled Function

C

```
bool PLIB_DMA_ChannelXINTSourceIsEnabled(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
    DMA_INT_TYPE dmaINTSource
);
```

Description

This function returns the enable status of the specified interrupt source for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Returns

- true - The interrupt is enabled
- false - The interrupt is not enabled

Remarks

This function implements an operation of the ChannelXINTSource feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXINTSource](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
bool dmaINTSourceEnableStatus;
dmaINTSourceEnableStatus = PLIB_DMA_ChannelXINTSourceIsEnabled (
    DMA_ID_0,
    spiDMAChannel,
    DMA_INT_ADDRESS_ERROR );
```

5.6.6.6.5 DMA Operating/Addressing Mode Configuration

5.6.6.6.5.1 PLIB_DMA_ChannelXAddressModeGet Function

C

```
DMA_CHANNEL_ADDRESSING_MODE PLIB_DMA_ChannelXAddressModeGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function gets the channel address mode.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

channelAddressMode - One of the possible source addressing modes listed by [DMA_CHANNEL_ADDRESSING_MODE](#)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_CHANNEL_ADDRESSING_MODE channelAddressMode;  
channelAddressMode = PLIB_DMA_ChannelXAddressModeGet ( DMA_ID_0,  
                                                         DMA_CHANNEL_4 );
```

5.6.6.6.5.2 PLIB_DMA_ChannelXAddressModeSelect Function**C**

```
void PLIB_DMA_ChannelXAddressModeSelect(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel,  
    DMA_CHANNEL_ADDRESSING_MODE channelAddressMode  
);
```

Description

This function sets the channel address mode.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
channelAddressMode	One of the possible channel addressing modes listed by DMA_CHANNEL_ADDRESSING_MODE

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_DMA_ChannelXAddressModeSelect (  
    DMA_ID_0,  
    DMA_CHANNEL_4,  
    DMA_ADDRESSING_REGISTER_INDIRECT_WITH_POST_INCREMENT );
```

5.6.6.6.5.3 PLIB_DMA_ChannelXDestinationAddressModeGet Function

C

```
DMA_DESTINATION_ADDRESSING_MODE PLIB_DMA_ChannelXDestinationAddressModeGet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function gets the source address mode of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

destinationAddressMode - One of the possible source addressing modes listed by [DMA_DESTINATION_ADDRESSING_MODE](#)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_SOURCE_ADDRESSING_MODE destinationAddressMode;  
destinationAddressMode = PLIB_DMA_ChannelXDestinationAddressModeGet (  
    DMA_ID_0,  
    DMA_CHANNEL_4 );
```

5.6.6.6.5.4 PLIB_DMA_ChannelXDestinationAddressModeSelect Function

C

```
void PLIB_DMA_ChannelXDestinationAddressModeSelect(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel,  
    DMA_DESTINATION_ADDRESSING_MODE destinationAddressMode  
);
```

Description

This function Sets the source address mode of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
destinationAddressMode	One of the possible source addressing modes listed by DMA_DESTINATION_ADDRESSING_MODE

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_DMA_ChannelXDestinationAddressModeSelect (
    DMA_ID_0,
    DMA_CHANNEL_4,
    DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE );
```

5.6.6.6.5.5 PLIB_DMA_ChannelXNullWriteModeDisable Function**C**

```
void PLIB_DMA_ChannelXNullWriteModeDisable(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function disables the Null Write mode. No dummy write is initiated.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_DMA_ChannelXNullWriteModeDisable ( DMA_ID_0,
    DMA_CHANNEL_4 );
```

5.6.6.6.5.6 PLIB_DMA_ChannelXNullWriteModeEnable Function**C**

```
void PLIB_DMA_ChannelXNullWriteModeEnable(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function enables the Null Write mode. A dummy write is initiated to the source address for every write to the destination address.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_DMA_ChannelXNullWriteModeEnable ( DMA_ID_0,
                                         DMA_CHANNEL_4 );
```

5.6.6.6.5.7 PLIB_DMA_ChannelXOperatingTransferModeGet Function

C

```
DMA_TRANSFER_MODE PLIB_DMA_ChannelXOperatingTransferModeGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function returns the current transfer/operating mode for the specified channel. (Transfer mode and operating mode are used interchangeably.)

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

channeltransferMode - One of the possible operating/transfer modes listed by [DMA_TRANSFER_MODE](#)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_TRANSFER_MODE channeltransferMode;
channeltransferMode = PLIB_DMA_ChannelXOperatingTransferModeGet (
                                                                DMA_ID_0,
                                                                DMA_CHANNEL_4 );
```

5.6.6.6.5.8 PLIB_DMA_ChannelXOperatingTransferModeSelect Function

C

```
void PLIB_DMA_ChannelXOperatingTransferModeSelect(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel,
    DMA_TRANSFER_MODE channeltransferMode
);
```

Description

This function selects the transfer/operating mode for the specified channel. (Transfer mode and operating mode are used interchangeably.)

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
channeltransferMode	One of the possible operating/transfer modes listed by DMA_TRANSFER_MODE

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_TRANSFER_MODE channeltransferMode = DMA_MODE_REPEATED_CONTINUOUS;
PLIB_DMA_ChannelXOperatingTransferModeSelect ( DMA_ID_0,
                                                DMA_CHANNEL_4,
                                                channeltransferMode );
```

5.6.6.6.5.9 PLIB_DMA_ChannelXReloadDisable Function**C**

```
void PLIB_DMA_ChannelXReloadDisable(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function disables reloading of the address and count registers. The source, destination address, and the DMA count registers are not reloaded to their previous values upon the start of the next operation.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_DMA_ChannelXReloadDisable ( DMA_ID_0,
                                  DMA_CHANNEL_4 );
```

5.6.6.6.5.10 PLIB_DMA_ChannelXReloadEnable Function**C**

```
void PLIB_DMA_ChannelXReloadEnable(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function enables reloading of the address and count registers. The source, destination address, and the DMA count registers are reloaded to their previous values upon the start of the next operation.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_DMA_ChannelXReloadEnable ( DMA_ID_0,
                                DMA_CHANNEL_4 );
```

5.6.6.6.5.11 PLIB_DMA_ChannelXSourceAddressModeGet Function**C**

```
DMA_SOURCE_ADDRESSING_MODE PLIB_DMA_ChannelXSourceAddressModeGet (
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function gets the source address mode of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

[DMA_SOURCE_ADDRESSING_MODE](#) - One of the possible source addressing modes listed by [DMA_SOURCE_ADDRESSING_MODE](#)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_SOURCE_ADDRESSING_MODE sourceAddressMode;
sourceAddressMode = PLIB_DMA_ChannelXSourceAddressModeGet ( DMA_ID_0,
                                                            DMA_CHANNEL_4 );
```

5.6.6.6.5.12 PLIB_DMA_ChannelXSourceAddressModeSelect Function**C**

```
void PLIB_DMA_ChannelXSourceAddressModeSelect (
```

```
DMA_MODULE_ID index,  
DMA_CHANNEL channel,  
DMA_SOURCE_ADDRESSING_MODE sourceAddressMode  
);
```

Description

This function sets the source address mode of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
sourceAddressMode	One of the possible source addressing modes listed by DMA_SOURCE_ADDRESSING_MODE

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_DMA_ChannelXSourceAddressModeSelect (  
    DMA_ID_0,  
    DMA_CHANNEL_4,  
    DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE );
```

5.6.6.6.5.13 PLIB_DMA_ChannelXReloadIsEnabled Function

C

```
bool PLIB_DMA_ChannelXReloadIsEnabled(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function returns the address and count registers reload enable status.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

- true - The address and count registers reload is enabled
- false - The address and count registers reload is disabled

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
bool chAddressCountReloadStatus;
```

```
chAddressCountReloadStatus = PLIB_DMA_ChannelXReloadIsEnabled (
    DMA_ID_0,
    DMA_CHANNEL_4 );
```

5.6.6.6.6 DMA Source/Destination/Peripheral Address Control Interface Routines

5.6.6.6.6.1 PLIB_DMA_ChannelXDestinationStartAddressGet Function

C

```
uint32_t PLIB_DMA_ChannelXDestinationStartAddressGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel
);
```

Description

This function reads the destination start address configured for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint32_t - The destination start address configured for this channel

Remarks

This function implements an operation of the ChannelXDestinationStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXDestinationStartAddress](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t DestinationStartAddress;
DestinationStartAddress = PLIB_DMA_ChannelXDestinationStartAddressGet (
    DMA_ID_0,
    spiDMAChannel );
```

5.6.6.6.6.2 PLIB_DMA_ChannelXDestinationStartAddressSet Function

C

```
void PLIB_DMA_ChannelXDestinationStartAddressSet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
    uint32_t destinationStartAddress
);
```

Description

This function writes the specified destination start address into the register corresponding to the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
destinationStartAddress	The destination start address

Returns

None.

Remarks

This function implements an operation of the ChannelXDestinationStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXDestinationStartAddress](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t destinationStartAddress = 0x00FDEA00;
PLIB_DMA_ChannelXDestinationStartAddressSet(
    DMA_ID_0,
    spiDMAChannel,
    destinationStartAddress );
```

5.6.6.6.3 PLIB_DMA_ChannelXPeripheralAddressGet Function**C**

```
uint16_t PLIB_DMA_ChannelXPeripheralAddressGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function gets the peripheral address configured for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint16_t - The peripheral address configured for the specified channel

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
uint16_t peripheraladdress;
peripheraladdress = PLIB_DMA_ChannelXPeripheralAddressGet ( DMA_ID_0,
    DMA_CHANNEL_4 );
```

5.6.6.6.4 PLIB_DMA_ChannelXPeripheralAddressSet Function**C**

```
void PLIB_DMA_ChannelXPeripheralAddressSet(
    DMA_MODULE_ID index,
```

```
    DMA_CHANNEL channel,
    uint16_t peripheraladdress
);
```

Description

This function sets the peripheral address for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
peripheraladdress	The peripheral address for the specified channel

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
uint16_t peripheraladdress = 0x100;
PLIB_DMA_ChannelXPeripheralAddressSet ( DMA_ID_0,
                                         DMA_CHANNEL_4,
                                         peripheraladdress );
```

5.6.6.6.5 PLIB_DMA_ChannelXSourceStartAddressGet Function

C

```
uint32_t PLIB_DMA_ChannelXSourceStartAddressGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel
);
```

Description

This function reads the source start address configured for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint32_t - The source start address configured for this channel

Remarks

This function implements an operation of the ChannelXSourceStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXSourceStartAddress](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
```



```
uint32_t SourceStartAddress;
SourceStartAddress = PLIB_DMA_ChannelXSourceStartAddressGet(DMA_ID_0,
                                                             spiDMACHannel );
```

5.6.6.6.6 PLIB_DMA_ChannelXSourceStartAddressSet Function

C

```
void PLIB_DMA_ChannelXSourceStartAddressSet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
    uint32_t sourceStartAddress
);
```

Description

This function writes the specified Source start address into the register corresponding to the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
sourceStartAddress	The source start address

Returns

None.

Remarks

This function implements an operation of the ChannelXSourceStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXSourceStartAddress](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMACHannel = DMA_CHANNEL_2;
uint32_t sourceStartAddress = 0x00FDEA00;
PLIB_DMA_ChannelXSourceStartAddressSet (    DMA_ID_0,
                                             spiDMACHannel,
                                             sourceStartAddress );
```

5.6.6.6.7 PLIB_DMA_ChannelXStartAddressOffsetGet Function

C

```
uint16_t PLIB_DMA_ChannelXStartAddressOffsetGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel,
    DMA_ADDRESS_OFFSET_TYPE offset
);
```

Description

This function gets the primary/secondary start address (DPSRAM) offset.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
offset	The type of the address offset (primary/secondary)

Returns

uint16_t - The primary/secondary DPSRAM start address offset

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
uint16_t addressOffsetA;  
addressOffsetA = PLIB_DMA_ChannelXStartAddressOffsetGet (  
    DMA_ID_0,  
    DMA_CHANNEL_4,  
    address,  
    DMA_ADDRESS_OFFSET_PRIMARY );
```

5.6.6.6.8 PLIB_DMA_ChannelXStartAddressOffsetSet Function**C**

```
void PLIB_DMA_ChannelXStartAddressOffsetSet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel,  
    uint16_t address,  
    DMA_ADDRESS_OFFSET_TYPE offset  
);
```

Description

This function sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
address	The primary/secondary DPSRAM start address offset
offset	The type of the address offset (primary/secondary)

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
uint16_t address = 0x100;  
PLIB_DMA_ChannelXStartAddressOffsetSet (    DMA_ID_0,  
    DMA_CHANNEL_4,  
    address,  
    DMA_ADDRESS_OFFSET_PRIMARY );
```

5.6.6.6.7 DMA Source/Destination/Peripheral Data Management

5.6.6.6.7.1 PLIB_DMA_ChannelXCellProgressPointerGet Function

C

```
uint16_t PLIB_DMA_ChannelXCellProgressPointerGet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL dmaChannel  
);
```

Description

This function returns the number of bytes transferred since the last event.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint16_t - The number of bytes transferred since the last event. (The cell progress pointer (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXCellProgressPointer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXCellProgressPointer](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;  
uint16_t CellProgress;  
CellProgress = PLIB_DMA_ChannelXCellProgressPointerGet ( DMA_ID_0,  
                                                         spiDMAChannel );
```

5.6.6.6.7.2 PLIB_DMA_ChannelXCellSizeGet Function

C

```
uint16_t PLIB_DMA_ChannelXCellSizeGet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL dmaChannel  
);
```

Description

This function reads the cell size (in bytes) configured for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint16_t - The cell size configured (in bytes) for this channel (The cell size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXCellSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXCellSize](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t cellSize;
cellSize = PLIB_DMA_ChannelXCellSizeGet (    DMA_ID_0,
                                             spiDMAChannel );
```

5.6.6.6.7.3 PLIB_DMA_ChannelXCellSizeSet Function**C**

```
void PLIB_DMA_ChannelXCellSizeSet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
    uint16_t CellSize
);
```

Description

This function writes the specified cell size into the register corresponding to the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
cellSize	The cell size in bytes (The cell size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Returns

None.

Remarks

This function implements an operation of the ChannelXCellSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXCellSize](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t cellSize = 0x10;
PLIB_DMA_ChannelXCellSizeSet ( DMA_ID_0, spiDMAChannel, cellSize );
```

5.6.6.6.7.4 PLIB_DMA_ChannelXDataSizeGet Function**C**

```
DMA_CHANNEL_DATA_SIZE PLIB_DMA_ChannelXDataSizeGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function returns the current data size for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

channelDataSize - One of the possible data sizes listed by [DMA_CHANNEL_DATA_SIZE](#)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_CHANNEL_DATA_SIZE channelDataSize;  
channelDataSize = PLIB_DMA_ChannelXDataSizeGet( DMA_ID_0,  
                                                  DMA_CHANNEL_4 );
```

5.6.6.6.7.5 PLIB_DMA_ChannelXDataSizeSelect Function**C**

```
void PLIB_DMA_ChannelXDataSizeSelect(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel,  
    DMA_CHANNEL_DATA_SIZE channelDataSize  
);
```

Description

This function selects the data size for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
channelDataSize	One of the possible data sizes listed by DMA_CHANNEL_DATA_SIZE

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_CHANNEL_DATA_SIZE channelDataSize = DMA_CHANNEL_DATA_8;  
PLIB_DMA_ChannelXDataSizeSelect ( DMA_ID_0,  
                                   DMA_CHANNEL_4,  
                                   channelDataSize );
```

5.6.6.6.7.6 PLIB_DMA_ChannelXDestinationPointerGet Function

C

```
uint16_t PLIB_DMA_ChannelXDestinationPointerGet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL dmaChannel  
);
```

Description

This function reads the current byte of the destination being pointed to for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint16_t - The destination byte being pointed to for this channel (The destination pointer (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXDestinationPointer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXDestinationPointer](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;  
uint16_t destinationbyte;  
destinationbyte = PLIB_DMA_ChannelXDestinationPointerGet ( DMA_ID_0,  
                                                         spiDMAChannel );
```

5.6.6.6.7.7 PLIB_DMA_ChannelXDestinationSizeGet Function

C

```
uint16_t PLIB_DMA_ChannelXDestinationSizeGet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL dmaChannel  
);
```

Description

This function reads the destination size configured for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint16_t - The destination size configured (in bytes) for this channel. (The destination size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXDestinationSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXDestinationSize](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t DestinationSize;
DestinationSize = PLIB_DMA_ChannelXDestinationSizeGet ( DMA_ID_0,
                                                         spiDMAChannel );
```

5.6.6.6.7.8 PLIB_DMA_ChannelXDestinationSizeSet Function**C**

```
void PLIB_DMA_ChannelXDestinationSizeSet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
    uint16_t destinationSize
);
```

Description

This function writes the specified destination size into the register corresponding to the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
destinationSize	The destination size (The destination size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Returns

None.

Remarks

This function implements an operation of the ChannelXDestinationSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXDestinationSize](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t destinationSize = 0xA00;
PLIB_DMA_ChannelXDestinationSizeSet( DMA_ID_0, spiDMAChannel, destinationSize );
```

5.6.6.6.7.9 PLIB_DMA_ChannelXPatternDataGet Function**C**

```
uint16_t PLIB_DMA_ChannelXPatternDataGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel
);
```

Description

This function returns pattern matching (for DMA abort) data programmed for the specified channel. (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Preconditions

None.

Parameters

Parameters	Description
<code>dmaChannel</code>	One of the possible DMA channels listed by DMA_CHANNEL

Returns

`uint16_t` - The pattern matching data programmed for the current channel (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXPatternData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPatternData](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t patternData;
patternData = PLIB_DMA_ChannelXPatternDataGet ( DMA_ID_0, spiDMAChannel );
```

5.6.6.6.7.10 PLIB_DMA_ChannelXPatternDataSet Function**C**

```
void PLIB_DMA_ChannelXPatternDataSet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
    uint16_t patternData
);
```

Description

This function writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel. (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Preconditions

None.

Parameters

Parameters	Description
<code>dmaChannel</code>	One of the possible DMA channels listed by DMA_CHANNEL
<code>patternData</code>	The pattern matching DATA programmed for the current channel (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Returns

None.

Remarks

This function implements an operation of the ChannelXPatternData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPatternData](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t patternData = '0';
PLIB_DMA_ChannelXPatternDataSet ( DMA_ID_0, spiDMAChannel, patternData );
```

5.6.6.6.7.11 PLIB_DMA_ChannelXSourcePointerGet Function**C**

```
uint16_t PLIB_DMA_ChannelXSourcePointerGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel
);
```

Description

This function reads the current byte of the source being pointed to for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint16_t - The source byte being pointed to for this channel. (The source pointer (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXSourcePointer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXSourcePointer](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t sourcebyte;
sourcebyte = PLIB_DMA_ChannelXSourcePointerGet ( DMA_ID_0, spiDMAChannel );
```

5.6.6.6.7.12 PLIB_DMA_ChannelXSourceSizeGet Function**C**

```
uint16_t PLIB_DMA_ChannelXSourceSizeGet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel
);
```

Description

This function reads the source size configured for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint16_t - The Source size configured (in bytes) for this channel. (The source size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXSourceSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXSourceSize](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t sourceSize;
sourceSize = PLIB_DMA_ChannelXSourceSizeGet (    DMA_ID_0,
                                                spiDMAChannel );
```

5.6.6.6.7.13 PLIB_DMA_ChannelXSourceSizeSet Function**C**

```
void PLIB_DMA_ChannelXSourceSizeSet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
    uint16_t sourceSize
);
```

Description

This function writes the specified source size into the register corresponding to the specified channel.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
sourceSize	The source size (The source size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Returns

None.

Remarks

This function implements an operation of the ChannelXSourceSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXSourceSize](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t sourceSize = 0xA00;
PLIB_DMA_ChannelXSourceSizeSet (    DMA_ID_0,
                                    spiDMAChannel,
                                    sourceSize );
```

5.6.6.6.7.14 PLIB_DMA_ChannelXTransferCountGet Function**C**

```
uint16_t PLIB_DMA_ChannelXTransferCountGet(
    DMA_MODULE_ID index,
```

```
    DMA_CHANNEL channel  
);
```

Description

This function gets the DMA data transfer count that is programmed for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

uint16_t - The DMA transfer count for the channel

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
uint16_t transferCount;  
transferCount = PLIB_DMA_ChannelXTransferCountGet ( DMA_ID_0,  
                                                    DMA_CHANNEL_4 );
```

5.6.6.6.7.15 PLIB_DMA_ChannelXTransferCountSet Function**C**

```
void PLIB_DMA_ChannelXTransferCountSet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel,  
    uint16_t transferCount  
);
```

Description

This function sets the DMA data transfer count for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
transferCount	The DMA transfer count for the channel

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
uint16_t transferCount = 0x10;  
PLIB_DMA_ChannelXTransferCountSet ( DMA_ID_0,  
                                    DMA_CHANNEL_4,  
                                    transferCount );
```

5.6.6.6.8 DMA Channel Configuration

5.6.6.6.8.1 PLIB_DMA_ChannelPrioritySelect Function

C

```
void PLIB_DMA_ChannelPrioritySelect(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL_PRIORITY channelPriority  
);
```

Description

This function sets the priority scheme of the DMA channels at the global level. This function is used in devices that do not have the per channel priority feature.

Preconditions

None.

Parameters

Parameters	Description
channelPriority	One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_CHANNEL_PRIORITY channelPriority = DMA_CHANNEL_ROUND_ROBIN;  
PLIB_DMA_ChannelPrioritySelect( DMA_ID_0, channelPriority );
```

5.6.6.6.8.2 PLIB_DMA_ChannelPriorityGet Function

C

```
DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelPriorityGet(  
    DMA_MODULE_ID index  
);
```

Description

This function gets the priority scheme of the DMA channels at the global level. This function is used in devices that do not have the per channel priority feature.

Preconditions

None.

Returns

channelPriority - One of the supported priorities listed by [DMA_CHANNEL_PRIORITY](#)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_CHANNEL_PRIORITY channelPriority;  
channelPriority = PLIB_DMA_ChannelPriorityGet( DMA_ID_0 );
```

5.6.6.6.8.3 PLIB_DMA_ChannelXAutoDisable Function

C

```
void PLIB_DMA_ChannelXAutoDisable(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function disables a channel after a block transfer is complete.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXAuto feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXAuto](#) function in your application to determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXAutoDisable( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.8.4 PLIB_DMA_ChannelXAutoEnable Function

C

```
void PLIB_DMA_ChannelXAutoEnable(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function enables the channel continuously. The channel is not automatically disabled after a block transfer is complete.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXAuto feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXAuto](#) function in your application to determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXAutoEnable( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.8.5 PLIB_DMA_ChannelXBusyActiveSet Function**C**

```
void PLIB_DMA_ChannelXBusyActiveSet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function sets the busy bit to active, indicating the channel is active or has been enabled.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXBusy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXBusy](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXBusyActiveSet ( DMA_ID_0, DMA_CHANNEL_4 );
```

5.6.6.6.8.6 PLIB_DMA_ChannelXBusyInActiveSet Function**C**

```
void PLIB_DMA_ChannelXBusyInActiveSet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function sets the busy bit to inactive, indicating the channel is inactive or has been disabled.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXBusy feature. This feature may not be available on all devices. Please

refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXBusy](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXBusyInActiveSet ( DMA_ID_0, DMA_CHANNEL_4 );
```

5.6.6.6.8.7 PLIB_DMA_ChannelXChainDisable Function**C**

```
void PLIB_DMA_ChannelXChainDisable(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function disables the channel chaining for the specified DMA channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXChainEnbl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the the [PLIB_DMA_ExistsChannelXChainEnbl](#) function in your application to determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXChainDisable( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.8.8 PLIB_DMA_ChannelXChainEnable Function**C**

```
void PLIB_DMA_ChannelXChainEnable(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function enables the channel chain feature.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXChainEnbl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXChainEnbl](#) function in your application to determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXChainEnable( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.8.9 PLIB_DMA_ChannelXChainToHigher Function**C**

```
void PLIB_DMA_ChannelXChainToHigher(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function will Chain the specified channel to a channel higher in natural priority. CH5 will be enabled by a CH4 transfer complete.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXChain feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXChain](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXChainToHigher ( DMA_ID_0, DMA_CHANNEL_4 );
```

5.6.6.6.8.10 PLIB_DMA_ChannelXChainToLower Function**C**

```
void PLIB_DMA_ChannelXChainToLower(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function will chain the specified channel to a channel lower in natural priority. CH3 will be enabled by a CH4 transfer complete.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXChain feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXChain](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXChainToLower ( DMA_ID_0, DMA_CHANNEL_4 );
```

5.6.6.6.8.11 PLIB_DMA_ChannelXDisable Function**C**

```
void PLIB_DMA_ChannelXDisable(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function will disable the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelX](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.8.12 PLIB_DMA_ChannelXEnable Function**C**

```
void PLIB_DMA_ChannelXEnable(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function will enable the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelX](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.8.13 PLIB_DMA_ChannelXPriorityGet Function**C**

```
DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelXPriorityGet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function gets the priority of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL

Returns

channelPriority - One of the supported priorities listed by [DMA_CHANNEL_PRIORITY](#)

Remarks

This function implements an operation of the ChannelXPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPriority](#) function in your application to determine whether this feature is available.

Example

```
DMA_CHANNEL channel = DMA_CHANNEL_0;  
DMA_CHANNEL_PRIORITY channelPriority;  
channelPriority = PLIB_DMA_ChannelXPriorityGet( DMA_ID_0, channel );
```

5.6.6.6.8.14 PLIB_DMA_ChannelXPrioritySelect Function**C**

```
void PLIB_DMA_ChannelXPrioritySelect(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel,
```

```
    DMA_CHANNEL_PRIORITY channelPriority  
);
```

Description

This function sets the priority of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL
channelPriority	One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Returns

None.

Remarks

This function implements an operation of the ChannelXPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPriority](#) function in your application to determine whether this feature is available.

Example

```
DMA_CHANNEL channel = DMA_CHANNEL_0;  
DMA_CHANNEL_PRIORITY channelPriority = DMA_CHANNEL_PRIORITY_3;  
PLIB_DMA_ChannelXPrioritySelect( DMA_ID_0, channel, channelPriority );
```

5.6.6.6.8.15 PLIB_DMA_ChannelXTransferDirectionGet Function**C**

```
DMA_CHANNEL_TRANSFER_DIRECTION PLIB_DMA_ChannelXTransferDirectionGet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function returns the data transfer direction of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

[DMA_CHANNEL_TRANSFER_DIRECTION](#) - The transfer direction indicated by the [DMA_CHANNEL_TRANSFER_DIRECTION](#)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_CHANNEL_TRANSFER_DIRECTION chTransferDirection;  
chTransferDirection = PLIB_DMA_ChannelXTransferDirectionGet (  
    DMA_ID_0,  
    DMA_CHANNEL_4 );
```

5.6.6.6.8.16 PLIB_DMA_ChannelXTransferDirectionSelect Function

C

```
void PLIB_DMA_ChannelXTransferDirectionSelect(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel,  
    DMA_CHANNEL_TRANSFER_DIRECTION chTransferDirection  
);
```

Description

This function selects the data transfer direction of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
chTransferDirection	The transfer direction indicated by the DMA_CHANNEL_TRANSFER_DIRECTION

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_DMA_ChannelXTransferDirectionSelect (  
    DMA_ID_0,  
    DMA_CHANNEL_4,  
    DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL );
```

5.6.6.6.8.17 PLIB_DMA_ChannelXDisabledDisablesEvents Function

C

```
void PLIB_DMA_ChannelXDisabledDisablesEvents(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function will allow the channel start/abort events to be ignored even if the channel is disabled.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXDisabled feature. This feature may not be available on all devices. Please

refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXDisabled](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXDisabledDisablesEvents ( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.8.18 PLIB_DMA_ChannelXDisabledEnablesEvents Function**C**

```
void PLIB_DMA_ChannelXDisabledEnablesEvents(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function will allow the channel register start/abort events even if the channel is disabled.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXDisabled feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXDisabled](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_ChannelXDisabledEnablesEvents ( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.8.19 PLIB_DMA_ChannelXPatternIgnoreByteDisable Function**C**

```
void PLIB_DMA_ChannelXPatternIgnoreByteDisable(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function disables the pattern match ignore byte.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXPatternIgnoreByte feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPatternIgnoreByte](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXPatternIgnoreByteDisable(DMA_ID_0, dmaChannel);
```

5.6.6.6.8.20 PLIB_DMA_ChannelXPatternIgnoreByteEnable Function**C**

```
void PLIB_DMA_ChannelXPatternIgnoreByteEnable(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function enables the pattern match ignore byte.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the ChannelXPatternIgnoreByte feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPatternIgnoreByte](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXPatternIgnoreByteEnable(DMA_ID_0, dmaChannel);
```

5.6.6.6.8.21 PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled Function**C**

```
bool PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled(
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function returns the state (enabled or disabled) of the pattern match ignore byte.

Preconditions

None.

Parameters

Parameters	Description
<code>dmaChannel</code>	One of the possible DMA channels listed by DMA_CHANNEL

Returns

- `true` - The pattern match ignore byte is enabled
- `false` - The pattern match ignore byte is disabled

Remarks

This function implements an operation of the `ChannelXPatternIgnoreByte` feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPatternIgnoreByte](#) function in your application to automatically determine whether this feature is available.

Example

```
bool patternIsEnabled;  
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;  
patternIsEnabled = PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled(DMA_ID_0, dmaChannel);
```

5.6.6.6.8.22 PLIB_DMA_ChannelXPatternIgnoreGet Function**C**

```
uint8_t PLIB_DMA_ChannelXPatternIgnoreGet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function returns the value of the pattern match ignore.

Preconditions

None.

Parameters

Parameters	Description
<code>dmaChannel</code>	One of the possible DMA channels listed by DMA_CHANNEL

Returns

`uint8_t` - Pattern match ignore value

Remarks

This function implements an operation of the `ChannelXPatternIgnore` feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPatternIgnore](#) function in your application to automatically determine whether this feature is available.

Example

```
uint8_t patternMatch;  
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;  
patternMatch = PLIB_DMA_ChannelXPatternIgnoreGet ( DMA_ID_0, dmaChannel);
```

5.6.6.6.8.23 PLIB_DMA_ChannelXPatternIgnoreSet Function**C**

```
void PLIB_DMA_ChannelXPatternIgnoreSet(  

```

```
DMA_MODULE_ID index,  
DMA_CHANNEL channel,  
uint8_t pattern  
);
```

Description

This function sets the value of the pattern match ignore.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
pattern	Pattern match ignore value

Returns

None.

Remarks

This function implements an operation of the ChannelXPatternIgnore feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPatternIgnore](#) function in your application to automatically determine whether this feature is available.

Example

```
uint8_t patternMatch = 0x8;  
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;  
PLIB_DMA_ChannelXPatternIgnoreSet ( DMA_ID_0, dmaChannel, patternMatch);
```

5.6.6.6.8.24 PLIB_DMA_ChannelXPatternLengthGet Function

C

```
DMA_PATTERN_LENGTH PLIB_DMA_ChannelXPatternLengthGet(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL dmaChannel  
);
```

Description

This function returns the length of the byte matching the CHPIGN bits during a pattern match that may be ignored during the pattern match determination when the CHPIGNEN bit is set.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

patternLen - Length of pattern match (either 1 or 2)

Remarks

This function implements an operation of the ChannelXPatternLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPatternLength](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
DMA_PATTERN_LENGTH patternLen;
patternLen = PLIB_DMA_ChannelXPatternLengthGet(DMA_ID_0, dmaChannel);
```

5.6.6.6.8.25 PLIB_DMA_ChannelXPatternLengthSet Function**C**

```
void PLIB_DMA_ChannelXPatternLengthSet(
    DMA_MODULE_ID index,
    DMA_CHANNEL dmaChannel,
    DMA_PATTERN_LENGTH patternLen
);
```

Description

This function sets the length of the pattern match ignore to 1 or 2 bytes.

Preconditions

None.

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
patternLen	Length of pattern match (either 1 or 2)

Returns

None.

Remarks

This function implements an operation of the ChannelXPatternLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXPatternLength](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
DMA_PATTERN_LENGTH patternLen;
patternLen = DMA_PATTERN_MATCH_LENGTH_1BYTE;
PLIB_DMA_ChannelXPatternLengthSet(DMA_ID_0, dmaChannel, patternLen);
```

5.6.6.6.9 DMA CRC Module Interface Routines**5.6.6.6.9.1 PLIB_DMA_CRCAppendModeDisable Function****C**

```
void PLIB_DMA_CRCAppendModeDisable(
    DMA_MODULE_ID index
);
```

Description

This function disables the CRC append mode. The DMA transfers data from the source through the CRC obeying WBO ([DMA_MODULE_ID](#) index, write byte order) as it writes the data to the destination.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the CRCAppendMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCAppendMode](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCAppendModeDisable( DMA_ID_0 );
```

5.6.6.6.9.2 PLIB_DMA_CRCAppendModeEnable Function**C**

```
void PLIB_DMA_CRCAppendModeEnable(  
    DMA_MODULE_ID index  
);
```

Description

This function enables the CRC append mode. The DMA transfers data from the source into the CRC, but not to the destination. When a block transfer completes, the DMA writes the calculated CRC value to the location specified by the CHxDSA register.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the CRCAppendMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCAppendMode](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCAppendModeEnable( DMA_ID_0 );
```

5.6.6.6.9.3 PLIB_DMA_CRCAppendModeIsEnabled Function**C**

```
bool PLIB_DMA_CRCAppendModeIsEnabled(  
    DMA_MODULE_ID index  
);
```

Description

This function gets the enable status of the CRC append mode.

Preconditions

None.

Returns

- true - CRC append mode is enabled
- false - CRC append mode is disabled

Remarks

This function implements an operation of the CRCAppendMode feature. This feature may not be available on all devices. Please

refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCAppendMode](#) function in your application to automatically determine whether this feature is available.

Example

```
bool DMAcrcAppendMode;
DMAcrcAppendMode = PLIB_DMA_CRCAppendModeIsEnabled( DMA_ID_0 );
```

5.6.6.6.9.4 PLIB_DMA_CRCBitOrderSelect Function

C

```
void PLIB_DMA_CRCBitOrderSelect(
    DMA_MODULE_ID index,
    DMA_CRC_BIT_ORDER bitOrder
);
```

Description

This function selects the bit order for checksum caluculation.

Preconditions

None.

Parameters

Parameters	Description
bitOrder	Specifies the bit order for CRC caluculation

Returns

None.

Remarks

This function implements an operation of the CRCBitOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCBitOrder](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCBitOrderSelect ( DMA_ID_0,DMA_CRC_BIT_ORDER_LSB );
```

5.6.6.6.9.5 PLIB_DMA_CRCByteOrderGet Function

C

```
DMA_CRC_BYTE_ORDER PLIB_DMA_CRCByteOrderGet(
    DMA_MODULE_ID index
);
```

Description

This function gets the current byte order selected by the DMA module CRC feature.

Preconditions

The WBO bit must be set to use this function.

Returns

byteOrder - One of the possible byte orders specified by [DMA_CRC_BYTE_ORDER](#)

Remarks

This function implements an operation of the CRCByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCByteOrder](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CRC_BYTE_ORDER byteOrder;  
byteOrder = PLIB_DMA_CRCByteOrderGet ( DMA_ID_0 );
```

5.6.6.6.9.6 PLIB_DMA_CRCByteOrderSelect Function**C**

```
void PLIB_DMA_CRCByteOrderSelect(  
    DMA_MODULE_ID index,  
    DMA_CRC_BYTE_ORDER byteOrder  
);
```

Description

This function selects the byte order.

Preconditions

The WBO bit must be set to use this function.

Parameters

Parameters	Description
byteOrder	One of the possible byte orders specified by DMA_CRC_BYTE_ORDER

Returns

None.

Remarks

This function implements an operation of the CRCByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCByteOrder](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCByteOrderSelect (    DMA_ID_0,  
                                DMA_CRC_SWAP_HALF_WORD_ON_WORD_BOUNDARY );
```

5.6.6.6.9.7 PLIB_DMA_CRCChannelGet Function**C**

```
DMA_CHANNEL PLIB_DMA_CRCChannelGet(  
    DMA_MODULE_ID index  
);
```

Description

This function returns the current DMA channel to which the CRC is assigned.

Preconditions

None.

Returns

crcChannel - One of the possible DMA channels listed by [DMA_CHANNEL](#)

Remarks

This function implements an operation of the CRCChannel feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCChannel](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CHANNEL crcChannel;  
crcChannel = PLIB_DMA_CRCChannelGet( DMA_ID_0 );
```

5.6.6.6.9.8 PLIB_DMA_CRCChannelSelect Function

C

```
void PLIB_DMA_CRCChannelSelect(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function assigns the CRC feature to the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

None.

Remarks

This function implements an operation of the CRCChannel feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCChannel](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCChannelSelect( DMA_ID_0,  
                           DMA_CHANNEL_5 );
```

5.6.6.6.9.9 PLIB_DMA_CRCDataRead Function

C

```
uint32_t PLIB_DMA_CRCDataRead(  
    DMA_MODULE_ID index  
);
```

Description

This function reads the contents of the DMA CRC data register.

Preconditions

None.

Returns

uint32_t - 32-bit CRC data (The CRC data (16-bit, 32-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the CRCData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCData](#) function in your application to automatically determine whether this feature is available.

Example

```
uint32_t DMACRCdata;  
DMACRCdata = PLIB_DMA_CRCDataRead ( DMA_ID_0 );
```

5.6.6.6.9.10 PLIB_DMA_CRCDataWrite Function**C**

```
void PLIB_DMA_CRCDataWrite(  
    DMA_MODULE_ID index,  
    uint32_t DMACRCdata  
);
```

Description

This function writes the contents of the DMA CRC data register.

Preconditions

None.

Parameters

Parameters	Description
DMACRCdata	32-bit CRC data (The CRC data (16-bit, 32-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Returns

None.

Remarks

This function implements an operation of the CRCData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCData](#) function in your application to automatically determine whether this feature is available.

Example

```
uint32_t DMACRCdata = 0x0E0E0E;  
PLIB_DMA_CRCDataWrite ( DMA_ID_0, DMACRCdata );
```

5.6.6.6.9.11 PLIB_DMA_CRCDisable Function**C**

```
void PLIB_DMA_CRCDisable(  
    DMA_MODULE_ID index  
);
```

Description

This function disables the DMA module CRC feature. The channel transfers proceed normally.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the CRC feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRC](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCDisable( DMA_ID_0 );
```

5.6.6.6.9.12 PLIB_DMA_CRCEnable Function**C**

```
void PLIB_DMA_CRCEnable(  
    DMA_MODULE_ID index  
);
```

Description

This function enables the DMA module CRC feature. The channel transfers are routed through the CRC.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the CRC feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRC](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCEnable( DMA_ID_0 );
```

5.6.6.6.9.13 PLIB_DMA_CRCPolynomialLengthGet Function**C**

```
uint8_t PLIB_DMA_CRCPolynomialLengthGet(  
    DMA_MODULE_ID index  
);
```

Description

This function gets the current polynomial length.

Preconditions

None.

Returns

uint8_t - Polynomial length

Remarks

This function implements an operation of the CRCPolynomialLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCPolynomialLength](#) function in your application to automatically determine whether this feature is available.

Example

```
uint8_t polyLength;  
polyLength = PLIB_DMA_CRCPolynomialLengthGet( DMA_ID_0 );
```

5.6.6.6.9.14 PLIB_DMA_CRCPolynomialLengthSet Function

C

```
void PLIB_DMA_CRCPolynomialLengthSet(  
    DMA_MODULE_ID index,  
    uint8_t polyLength  
);
```

Description

This function Selects the polynomial length.

Preconditions

None.

Parameters

Parameters	Description
polyLength	Polynomial length

Returns

None.

Remarks

This function implements an operation of the CRCPolynomialLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCPolynomialLength](#) function in your application to automatically determine whether this feature is available.

Example

```
uint8_t polyLength = 0x2;  
PLIB_DMA_CRCPolynomialLengthSet( DMA_ID_0, polyLength );
```

5.6.6.6.9.15 PLIB_DMA_CRCTypeGet Function

C

```
DMA_CRC_TYPE PLIB_DMA_CRCTypeGet(  
    DMA_MODULE_ID index  
);
```

Description

This function gets the DMA module CRC feature type. The CRC feature will compute either the IP header checksum or the Linear Shift Feedback Register (LFSR) checksum.

Preconditions

None.

Returns

CRCType - One of the possible CRC checksums listed by [DMA_CRC_TYPE](#).

Remarks

This function implements an operation of the CRCType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCType](#) function in your application to automatically determine whether this feature is available.

Example

```
DMA_CRC_TYPE CRCType;  
CRCType = PLIB_DMA_CRCTypeGet( DMA_ID_0 );
```


5.6.6.6.9.16 PLIB_DMA_CRCTypeSet Function

C

```
void PLIB_DMA_CRCTypeSet(  
    DMA_MODULE_ID index,  
    DMA_CRC_TYPE CRCType  
)  
;
```

Description

This function selects the DMA module CRC feature type. The CRC feature will compute either the IP header checksum or the Linear Shift Feedback Register (LFSR) checksum.

Preconditions

None.

Parameters

Parameters	Description
CRCType	One of the possible CRC checksums listed by DMA_CRC_TYPE

Returns

None.

Remarks

This function implements an operation of the CRCType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCType](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCTypeSet(DMA_ID_0,  
    DMA_CRC_IP_HEADER );
```

5.6.6.6.9.17 PLIB_DMA_CRCWriteByteOrderAlter Function

C

```
void PLIB_DMA_CRCWriteByteOrderAlter(  
    DMA_MODULE_ID index  
)  
;
```

Description

This function enables byte order alteration as specified by the BYTO<1:0> bits. The source data is written to the destination reordered as defined by the BYTO<1:0> bits.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the CRCWriteByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCWriteByteOrder](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCWriteByteOrderAlter ( DMA_ID_0 );
```

5.6.6.6.9.18 PLIB_DMA_CRCWriteByteOrderMaintain Function

C

```
void PLIB_DMA_CRCWriteByteOrderMaintain(  
    DMA_MODULE_ID index  
);
```

Description

This function disables byte order alteration. The source data is written to the destination unaltered.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the CRCWriteByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCWriteByteOrder](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_CRCWriteByteOrderMaintain ( DMA_ID_0 );
```

5.6.6.6.9.19 PLIB_DMA_CRCXOREnableGet Function

C

```
uint32_t PLIB_DMA_CRCXOREnableGet(  
    DMA_MODULE_ID index  
);
```

Description

This function reads the CRC XOR register.

Preconditions

None.

Returns

uint32_t - 32-bit CRC XOR enable mask data (The CRC data (16-bit, 32-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the CRCXOREnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCXOREnable](#) function in your application to automatically determine whether this feature is available.

Example

```
uint32_t DMACRCXORdata;  
DMACRCXORdata = PLIB_DMA_CRCXOREnableGet ( DMA_ID_0 );
```

5.6.6.6.9.20 PLIB_DMA_CRCXOREnableSet Function

C

```
void PLIB_DMA_CRCXOREnableSet(  
    DMA_MODULE_ID index,
```

```
uint32_t DMACRCXOREnableMask
);
```

Description

This function writes to the CRC XOR enable register as per the specified enable mask. Each enabled bit will be taken as input to the shift register.

Preconditions

None.

Parameters

Parameters	Description
DMACRCXOREnableMask	32-bit CRC XOR enable mask data (The CRC data (16-bit, 32-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Returns

None.

Remarks

This function implements an operation of the CRCXOREnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRCXOREnable](#) function in your application to automatically determine whether this feature is available.

Example

```
uint32_t DMACRCXOREnableMask = 0x05FFFFFF;
PLIB_DMA_CRCXOREnableSet ( DMA_ID_0, DMACRCXOREnableMask );
```

5.6.6.6.10 DMA Global Status Routines

5.6.6.6.10.1 PLIB_DMA_CRCIsEnabled Function

C

```
bool PLIB_DMA_CRCIsEnabled(
    DMA_MODULE_ID index
);
```

Description

This function gets the enable status of the CRC feature.

Preconditions

None.

Returns

- true - The CRC feature is enabled
- false - The CRC feature is disabled

Remarks

This function implements an operation of the CRC feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsCRC](#) function in your application to automatically determine whether this feature is available.

Example

```
bool DMAcrcStatus;
```

```
DMAcrcStatus = PLIB_DMA_CRCIsEnabled( DMA_ID_0 );
```

5.6.6.6.10.2 PLIB_DMA_IsBusy Function

C

```
bool PLIB_DMA_IsBusy(  
    DMA_MODULE_ID index  
);
```

Description

This function gets the BUSY bit of the DMA controller.

Preconditions

None.

Returns

- true - DMA module is active
- false - DMA module is disabled and is not actively transferring data

Remarks

This function implements an operation of the Busy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsBusy](#) function in your application to automatically determine whether this feature is available.

Example

```
bool dmaBusyStatus;  
dmaBusyStatus = PLIB_DMA_IsBusy( DMA_ID_0 );
```

5.6.6.6.10.3 PLIB_DMA_IsEnabled Function

C

```
bool PLIB_DMA_IsEnabled(  
    DMA_MODULE_ID index  
);
```

Description

This function returns the DMA module enable status.

Preconditions

None.

Returns

- true - The DMA is enabled
- false - The DMA is disabled

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsEnableControl](#) function in your application to automatically determine whether this feature is available.

Example

```
PLIB_DMA_IsEnabled( DMA_ID_0 );
```

5.6.6.6.10.4 PLIB_DMA_LastBusAccessIsRead Function

C

```
bool PLIB_DMA_LastBusAccessIsRead(  
    DMA_MODULE_ID index  
);
```

Description

This function returns true if the last DMA bus access was a read.

Preconditions

None.

Returns

- true - The last bus access was a read
- false - The last bus access was not a read

Remarks

This function implements an operation of the LastBusAccess feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsLastBusAccess](#) function in your application to automatically determine whether this feature is available.

Example

```
bool dmaLastBusAccessType;  
dmaLastBusAccessType = PLIB_DMA_LastBusAccessIsRead( DMA_ID_0 );
```

5.6.6.6.10.5 PLIB_DMA_LastBusAccessIsWrite Function

C

```
bool PLIB_DMA_LastBusAccessIsWrite(  
    DMA_MODULE_ID index  
);
```

Description

This function returns true if the last DMA bus access was a write operation.

Preconditions

None.

Returns

- true - The last bus access was a write operation
- false - The last bus access was not a write operation

Remarks

This function implements an operation of the LastBusAccess feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsLastBusAccess](#) function in your application to automatically determine whether this feature is available.

Example

```
bool dmaLastBusAccessType;  
dmaLastBusAccessType = PLIB_DMA_LastBusAccessIsWrite( DMA_ID_0 );
```

5.6.6.6.10.6 PLIB_DMA_LastChannelActive Function

C

```
DMA_CHANNEL PLIB_DMA_LastChannelActive(  
    DMA_MODULE_ID index  
);
```

Description

This function returns the last DMA channel that participated in the data transfer.

Preconditions

None.

Returns

channel - One of the existing DMA channels listed by [DMA_CHANNEL](#)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_CHANNEL lastActiveChannel;  
lastActiveChannel = PLIB_DMA_LastChannelActive( DMA_ID_0 );
```

5.6.6.6.10.7 PLIB_DMA_RecentAddressAccessed Function

C

```
uint32_t PLIB_DMA_RecentAddressAccessed(  
    DMA_MODULE_ID index  
);
```

Description

This function returns the address of the most recent DMA access.

Preconditions

None.

Returns

uint32_t - The most recent address accessed by the DMA

Remarks

This function implements an operation of the RecentAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsRecentAddress](#) function in your application to automatically determine whether this feature is available.

Example

```
uint32_t dmaLastAddressAccessed;  
dmaLastAddressAccessed = PLIB_DMA_RecentAddressAccessed( DMA_ID_0 );
```

5.6.6.6.10.8 PLIB_DMA_SuspendIsEnabled Function

C

```
bool PLIB_DMA_SuspendIsEnabled(  
    DMA_MODULE_ID index  
);
```

Description

This function returns the DMA suspend status.

Preconditions

None.

Returns

- true - The DMA transfers are suspended
- false - The DMA operates normally

Remarks

This function implements an operation of the Suspend feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsSuspend](#) function in your application to automatically determine whether this feature is available.

Example

```
bool dmaSuspendStatus;  
dmaSuspendStatus = PLIB_DMA_SuspendIsEnabled( DMA_ID_0 );
```

5.6.6.6.11 DMA Channel Status Routines

5.6.6.6.11.1 PLIB_DMA_ChannelXAutoIsEnabled Function

C

```
bool PLIB_DMA_ChannelXAutoIsEnabled(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function returns the channel automatic enable status.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

- true - Channel automatic enable is on
- false - Channel automatic enable is off

Remarks

This function implements an operation of the ChannelXAuto feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXAuto](#) function in your application to determine whether this feature is available.

Example

```
bool ChAutoEnableStatus;  
ChAutoEnableStatus = PLIB_DMA_ChannelXAutoIsEnabled(DMA_ID_0,  
    DMA_CHANNEL_2 );
```

5.6.6.6.11.2 PLIB_DMA_ChannelXBufferedDataIsWritten Function

C

```
bool PLIB_DMA_ChannelXBufferedDataIsWritten(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function returns the buffered data write status for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL

Returns

- true - The content of the DMA buffer has not been written to the location specified in the destination/source address or in Null Write mode
- false - The content of the DMA buffer has been written to the location specified in the destination/source address or in Null Write mode

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
bool chBuffWriteStatus;  
chBuffWriteStatus = PLIB_DMA_ChannelXBufferedDataIsWritten( DMA_ID_0,  
                                                             DMA_CHANNEL_3 );
```

5.6.6.6.11.3 PLIB_DMA_ChannelXBusyIsBusy Function

C

```
bool PLIB_DMA_ChannelXBusyIsBusy(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function returns the busy status of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

- true - The channel is active or has been enabled
- false - The channel is inactive or has been disabled

Remarks

This function implements an operation of the ChannelXBusy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXBusy](#) function in your application to automatically determine whether this feature is available.

Example

```
bool chBusyStatus;  
chBusyStatus = PLIB_DMA_ChannelXBusyIsBusy ( DMA_ID_0, DMA_CHANNEL_4 );
```

5.6.6.6.11.4 PLIB_DMA_ChannelXChainIsEnabled Function**C**

```
bool PLIB_DMA_ChannelXChainIsEnabled(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function returns the chain status of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

- true - The channel chain is on for this channel
- false - The channel chain is off for this channel

Remarks

This function implements an operation of the ChannelXChainEnbl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelXChainEnbl](#) function in your application to determine whether this feature is available.

Example

```
bool ChchainStatus;  
ChchainStatus = PLIB_DMA_ChannelXChainIsEnabled( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.11.5 PLIB_DMA_ChannelXCollisionStatus Function**C**

```
bool PLIB_DMA_ChannelXCollisionStatus(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel,  
    DMA_CHANNEL_COLLISION collisionType  
);
```

Description

This function returns the status of the specified collision type for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL
collisonType	Collision type listed by DMA_CHANNEL_COLLISION

Returns

- true - A collision specified by collisonType was detected
- false - No collision of type collisonType was detected

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
bool memWriteCollisionStatus;  
memWriteCollisionStatus = PLIB_DMA_ChannelXMemoryWriteCollisionStatus(  
    DMA_ID_0,  
    DMA_CHANNEL_3,  
    DMA_CHANNEL_COLLISION_MEMORY );
```

5.6.6.6.11.6 PLIB_DMA_ChannelXEventIsDetected Function**C**

```
bool PLIB_DMA_ChannelXEventIsDetected(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function returns the event status on the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

- true - An event was detected
- false - No events were detected

Remarks

This function implements an operation of the ChannelXEvent feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_DMA_ExistsChannelXEvent](#) function in your application to determine whether this feature is available.

Example

```
bool channeleventStatus;  
channeleventStatus = PLIB_DMA_ChannelXEventIsDetected( DMA_ID_0,  
    DMA_CHANNEL_2 );
```

5.6.6.6.11.7 PLIB_DMA_ChannelXIsEnabled Function

C

```
bool PLIB_DMA_ChannelXIsEnabled(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function will return the enable status of the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

- true - The specified DMA channel is enabled
- false - The specified DMA channel is disabled

Remarks

This function implements an operation of the ChannelX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelX](#) function in your application to automatically determine whether this feature is available.

Example

```
bool chEnableStatus;  
chEnableStatus = PLIB_DMA_ChannelXIsEnabled ( DMA_ID_0, DMA_CHANNEL_2 );
```

5.6.6.6.11.8 PLIB_DMA_ChannelXNullWriteModeIsEnabled Function

C

```
bool PLIB_DMA_ChannelXNullWriteModeIsEnabled(  
    DMA_MODULE_ID index,  
    DMA_CHANNEL channel  
);
```

Description

This function returns the enable status of the Null Write mode for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Returns

- true - Null write mode is enabled for this channel
- false - Null write mode is disabled for this channel

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
bool chNullWriteStatus;
chNullWriteStatus = PLIB_DMA_ChannelXNullWriteModeIsEnabled (
    DMA_ID_0,
    DMA_CHANNEL_4 );
```

5.6.6.6.11.9 PLIB_DMA_ChannelXPingPongModeGet Function

C

```
DMA_PING_PONG_MODE PLIB_DMA_ChannelXPingPongModeGet (
    DMA_MODULE_ID index,
    DMA_CHANNEL channel
);
```

Description

This function returns the Ping-Pong mode status for the specified channel.

Preconditions

None.

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL

Returns

mode - One of the possible Ping-Pong modes

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
DMA_PING_PONG_MODE chPingPongStatus;
chPingPongStatus = PLIB_DMA_ChannelXPingPongModeGet(DMA_ID_0,
    DMA_CHANNEL_3 );
if (DMA_PING_PONG_SECONDARY == chPingPongStatus)
{
    \Application
}
```

5.6.6.6.11.10 PLIB_DMA_ChannelBitsGet Function

C

```
uint8_t PLIB_DMA_ChannelBitsGet(
    DMA_MODULE_ID index
);
```

Description

This function returns the channel bits.

Preconditions

None.

Returns

uint8_t - DMA channel bits

Remarks

This function implements an operation of the ChannelBits feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the [PLIB_DMA_ExistsChannelBits](#) function in your application to automatically determine whether this feature is available.

Example

```
uint8_t dmaChBits;  
dmaChBits = PLIB_DMA_ChannelBitsGet( DMA_ID_0 );
```

5.6.6.6.12 Data Types and Constants

5.6.6.6.12.1 DMA_ADDRESS_OFFSET_TYPE Enumeration

C

```
typedef enum {  
    DMA_ADDRESS_OFFSET_PRIMARY,  
    DMA_ADDRESS_OFFSET_SECONDARY  
} DMA_ADDRESS_OFFSET_TYPE;
```

Description

DMA address offsets

This enumeration lists all the possible DMA address offsets.

Members

Members	Description
DMA_ADDRESS_OFFSET_PRIMARY	address offset-A
DMA_ADDRESS_OFFSET_SECONDARY	address offset-B

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.2 DMA_CHANNEL Enumeration

C

```
typedef enum {  
    DMA_CHANNEL_0,  
    DMA_CHANNEL_1,  
    DMA_CHANNEL_2,  
    DMA_CHANNEL_3,  
    DMA_CHANNEL_4,  
    DMA_CHANNEL_5,  
    DMA_CHANNEL_6,  
    DMA_CHANNEL_7  
} DMA_CHANNEL;
```

Description

DMA channel

This enumeration lists all the possible DMA channels available.

Members

Members	Description
DMA_CHANNEL_0	DMA Channel 0

DMA_CHANNEL_1	DMA Channel 1
DMA_CHANNEL_2	DMA Channel 2
DMA_CHANNEL_3	DMA Channel 3
DMA_CHANNEL_4	DMA Channel 4
DMA_CHANNEL_5	DMA Channel 5
DMA_CHANNEL_6	DMA Channel 6
DMA_CHANNEL_7	DMA Channel 7

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.3 DMA_CHANNEL_ADDRESSING_MODE Enumeration**C**

```
typedef enum {
    DMA_ADDRESSING_REGISTER_INDIRECT_WITH_POST_INCREMENT,
    DMA_ADDRESSING_REGISTER_INDIRECT_WITHOUT_POST_INCREMENT,
    DMA_ADDRESSING_PERIPHERAL_INDIRECT
} DMA_CHANNEL_ADDRESSING_MODE;
```

Description

DMA channel addressing modes

This enumeration lists all the possible channel addressing modes.

Members

Members	Description
DMA_ADDRESSING_REGISTER_INDIRECT_WITH_POST_INCREMENT	register indirect with post increment
DMA_ADDRESSING_REGISTER_INDIRECT_WITHOUT_POST_INCREMENT	register indirect without post increment
DMA_ADDRESSING_PERIPHERAL_INDIRECT	peripheral indirect addressing

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.4 DMA_CHANNEL_COLLISION Enumeration**C**

```
typedef enum {
    DMA_CHANNEL_COLLISION_MEMORY,
    DMA_CHANNEL_COLLISION_PERIPHERAL,
    DMA_CHANNEL_COLLISION_TRANSFER_REQUEST
} DMA_CHANNEL_COLLISION;
```

Description

DMA channel collision types

This enumeration lists all the possible channel collision types.

Members

Members	Description
DMA_CHANNEL_COLLISION_MEMORY	Memory Write Collision
DMA_CHANNEL_COLLISION_PERIPHERAL	Peripheral Write collision
DMA_CHANNEL_COLLISION_TRANSFER_REQUEST	Transfer request collision

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.5 DMA_CHANNEL_DATA_SIZE Enumeration**C**

```
typedef enum {  
    DMA_CHANNEL_DATA_8,  
    DMA_CHANNEL_DATA_16  
} DMA_CHANNEL_DATA_SIZE;
```

Description

DMA channel data size

This enumeration lists all the possible data sizes for the DMA channels.

Members

Members	Description
DMA_CHANNEL_DATA_8	8 bit data size
DMA_CHANNEL_DATA_16	16 bit data size

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.6 DMA_CHANNEL_PRIORITY Enumeration**C**

```
typedef enum {  
    DMA_CHANNEL_PRIORITY_0,  
    DMA_CHANNEL_PRIORITY_1,  
    DMA_CHANNEL_PRIORITY_2,  
    DMA_CHANNEL_PRIORITY_3,  
    DMA_CHANNEL_ROUND_ROBIN,  
    DMA_CHANNEL_FIXED_PRIORITY  
} DMA_CHANNEL_PRIORITY;
```

Description

DMA channel priority types

This enumeration lists all the possible channel priorities.

Members

Members	Description
DMA_CHANNEL_PRIORITY_0	priority 0
DMA_CHANNEL_PRIORITY_1	priority 1
DMA_CHANNEL_PRIORITY_2	priority 2
DMA_CHANNEL_PRIORITY_3	priority 3
DMA_CHANNEL_ROUND_ROBIN	round-robin scheme
DMA_CHANNEL_FIXED_PRIORITY	fixed priority scheme

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.7 DMA_CHANNEL_TRANSFER_DIRECTION Enumeration

C

```
typedef enum {
    DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL,
    DMA_READ_FROM_PERIPHERAL_WRITE_TO_MEMORY,
    DMA_READ_FROM_MEMORY_WRITE_TO_MEMORY,
    DMA_READ_FROM_PERIPHERAL_WRITE_TO_PERIPHERAL
} DMA_CHANNEL_TRANSFER_DIRECTION;
```

Description

DMA channel data transfer direction

This enumeration lists all the possible data transfer directions.

Members

Members	Description
DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL	DMA transfer happens from the memory to peripheral
DMA_READ_FROM_PERIPHERAL_WRITE_TO_MEMORY	DMA transfer happens from the peripheral to memory
DMA_READ_FROM_MEMORY_WRITE_TO_MEMORY	DMA transfer happens from the memory to memory
DMA_READ_FROM_PERIPHERAL_WRITE_TO_PERIPHERAL	DMA transfer happens from the peripheral to peripheral

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.8 DMA_CHANNEL_TRIGGER_TYPE Enumeration

C

```
typedef enum {
    DMA_CHANNEL_TRIGGER_TRANSFER_START,
    DMA_CHANNEL_TRIGGER_TRANSFER_ABORT,
    DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT
} DMA_CHANNEL_TRIGGER_TYPE;
```

Description

DMA trigger types

This enumeration lists all the possible DMA channel triggers.

Members

Members	Description
DMA_CHANNEL_TRIGGER_TRANSFER_START	Trigger for DMA transfer start
DMA_CHANNEL_TRIGGER_TRANSFER_ABORT	Trigger for DMA transfer abort
DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT	Trigger for DMA transfer abort via pattern match

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.9 DMA_CRC_BIT_ORDER Enumeration

C

```
typedef enum {
    DMA_CRC_BIT_ORDER_LSB,
    DMA_CRC_BIT_ORDER_MSB
}
```



```
} DMA_CRC_BIT_ORDER;
```

Description

DMA CRC calculation bit order

This enumeration lists all the possible CRC calculation bit orders

Members

Members	Description
DMA_CRC_BIT_ORDER_LSB	CRC is calculated least significant bit first
DMA_CRC_BIT_ORDER_MSB	CRC is calculated most significant bit first

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.10 DMA_CRC_BYTE_ORDER Enumeration

C

```
typedef enum {
    DMA_CRC_BYTEORDER_NO_SWAPPING,
    DMA_CRC_SWAP_BYTE_ON_WORD_BOUNDARY,
    DMA_CRC_SWAP_HALF_WORD_ON_WORD_BOUNDARY,
    DMA_CRC_SWAP_BYTE_ON_HALF_WORD_BOUNDARY
} DMA_CRC_BYTE_ORDER;
```

Description

CRC Byte order type

This enumeration lists all the possible byte orders handled by CRC module.

Members

Members	Description
DMA_CRC_BYTEORDER_NO_SWAPPING	No swapping, Source byte order
DMA_CRC_SWAP_BYTE_ON_WORD_BOUNDARY	Endian byte swap on word boundaries (reverse source byte order)
DMA_CRC_SWAP_HALF_WORD_ON_WORD_BOUNDARY	Swap half-words on word boundaries (reverse source half-word order with source byte order per half-word)
DMA_CRC_SWAP_BYTE_ON_HALF_WORD_BOUNDARY	Endian byte swap on half-word boundaries (reverse source half-word order with reverse source byte order per half-word)

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.11 DMA_CRC_TYPE Enumeration

C

```
typedef enum {
    DMA_CRC_IP_HEADER,
    DMA_CRC_LFSR
} DMA_CRC_TYPE;
```

Description

CRC type

This enumeration lists all the possible checksums handled by CRC module.

Members

Members	Description
DMA_CRC_IP_HEADER	CRC module will calculate IP header checksum
DMA_CRC_LFSR	CRC module will calculate Linear Feedback Shift Registers checksum

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.12 DMA_DESTINATION_ADDRESSING_MODE Enumeration**C**

```
typedef enum {
    DMA_ADDRESSING_DESTINATION_UNCHANGED,
    DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE,
    DMA_ADDRESSING_DESTINATION_DECREMENT_BASED_ON_SIZE,
    DMA_ADDRESSING_DESTINATION_PERIPHERAL_INDIRECT
} DMA_DESTINATION_ADDRESSING_MODE;
```

Description

DMA destination addressing modes

This enumeration lists all the possible destination addressing modes.

Members

Members	Description
DMA_ADDRESSING_DESTINATION_UNCHANGED	DMADST remains unchanged after a transfer completion
DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE	DMADST is incremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_DESTINATION_DECREMENT_BASED_ON_SIZE	DMADST is decremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_DESTINATION_PERIPHERAL_INDIRECT	DMADST is used in peripheral indirect addressing and remains unchanged

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.13 DMA_INT_TYPE Enumeration**C**

```
typedef enum {
    DMA_INT_ADDRESS_ERROR,
    DMA_INT_TRANSFER_ABORT,
    DMA_INT_CELL_TRANSFER_COMPLETE,
    DMA_INT_BLOCK_TRANSFER_COMPLETE,
    DMA_INT_DESTINATION_HALF_FULL,
    DMA_INT_DESTINATION_DONE,
    DMA_INT_SOURCE_HALF_EMPTY,
    DMA_INT_SOURCE_DONE,
    DMA_INT_COUNT_HALF_DONE,
    DMA_INT_CHANNEL_OVERRUN,
    DMA_INT_COMPLETE,
    DMA_INT_LOW_ADDRESS_LIMIT,
    DMA_INT_HIGH_ADDRESS_LIMIT
} DMA_INT_TYPE;
```

Description

DMA channel X interrupt types

This enumeration lists all the possible interrupt types for a channel. Not to be confused with the DMA trigger/abort interrupt sources (supported only for few parts).

Members

Members	Description
DMA_INT_ADDRESS_ERROR	Channel address error interrupt
DMA_INT_TRANSFER_ABORT	channel transfer abort interrupt
DMA_INT_CELL_TRANSFER_COMPLETE	channel cell transfer complete
DMA_INT_BLOCK_TRANSFER_COMPLETE	Channel block transfer complete
DMA_INT_DESTINATION_HALF_FULL	Channel destination half full interrupt
DMA_INT_DESTINATION_DONE	Channel destination done interrupt
DMA_INT_SOURCE_HALF_EMPTY	Channel source half empty interrupt
DMA_INT_SOURCE_DONE	Channel source done interrupt
DMA_INT_COUNT_HALF_DONE	DMA count has reached its halfway
DMA_INT_CHANNEL_OVERRUN	channel overrun. channel is triggered while it is still completing the operation based on the previous trigger
DMA_INT_COMPLETE	DMA complete operation interrupt
DMA_INT_LOW_ADDRESS_LIMIT	DMA low address limit interrupt
DMA_INT_HIGH_ADDRESS_LIMIT	DMA high address limit interrupt

Remarks

Not to be confused with the DMA trigger/abort interrupt sources (supported only for few parts). This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.14 DMA_MODULE_ID Enumeration**C**

```
typedef enum {
    DMA_ID_0
} DMA_MODULE_ID;
```

Description

DMA module ID

This data type defines the possible instances of the DMA module.

Members

Members	Description
DMA_ID_0	first instance of the DMA

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.15 DMA_PATTERN_LENGTH Enumeration**C**

```
typedef enum {
    DMA_PATTERN_MATCH_LENGTH_1BYTE,
```

```

    DMA_PATTERN_MATCH_LENGTH_2BYTES
} DMA_PATTERN_LENGTH;

```

Description

DMA pattern length

This enumeration gives the length of the pattern to be matched with CHPIGN<7:0> in DCHxCON<31:24>.

Members

Members	Description
DMA_PATTERN_MATCH_LENGTH_1BYTE	The Length of matching pattern with CHPIGN<7:0> in DCHxCON<31:24> is 1 BYTE
DMA_PATTERN_MATCH_LENGTH_2BYTES	The Length of matching pattern with CHPIGN<7:0> in DCHxCON<31:24> is 2 BYTES

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.16 DMA_PING_PONG_MODE Enumeration**C**

```

typedef enum {
    DMA_PING_PONG_PRIMARY,
    DMA_PING_PONG_SECONDARY
} DMA_PING_PONG_MODE;

```

Description

DMA PING-PONG modes

This enumeration lists all the possible ping-pong modes.

Members

Members	Description
DMA_PING_PONG_PRIMARY	DMAxSTA is selected in the ping-pong mode
DMA_PING_PONG_SECONDARY	DMAxSTB is selected in the ping-pong mode

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.17 DMA_SOURCE_ADDRESSING_MODE Enumeration**C**

```

typedef enum {
    DMA_ADDRESSING_SOURCE_UNCHANGED,
    DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE,
    DMA_ADDRESSING_SOURCE_DECREMENT_BASED_ON_SIZE,
    DMA_ADDRESSING_SOURCE_PERIPHERAL_INDIRECT
} DMA_SOURCE_ADDRESSING_MODE;

```

Description

DMA source addressing modes

This enumeration lists all the possible source addressing modes.

Members

Members	Description
DMA_ADDRESSING_SOURCE_UNCHANGED	DMASRC remains unchanged after a transfer completion
DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE	DMASRC is incremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_SOURCE_DECREMENT_BASED_ON_SIZE	DMASRC is decremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_SOURCE_PERIPHERAL_INDIRECT	DMASRC is used in peripheral indirect addressing and remains unchanged

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.18 DMA_TRANSFER_MODE Enumeration**C**

```
typedef enum {
    DMA_MODE_ONE_SHOT,
    DMA_MODE_REPEATED_ONE_SHOT,
    DMA_MODE_CONTINUOUS,
    DMA_MODE_REPEATED_CONTINUOUS,
    DMA_MODE_ONE_SHOT_PING_PONG_ENABLED,
    DMA_MODE_ONE_SHOT_PING_PONG_DISABLED,
    DMA_MODE_CONTINUOUS_PING_PONG_DISABLED,
    DMA_MODE_CONTINUOUS_PING_PONG_ENABLED
} DMA_TRANSFER_MODE;
```

Description

DMA transfer/operating mode

This enumeration lists the possible DMA transfer/operating modes.

Members

Members	Description
DMA_MODE_ONE_SHOT	one-shot transfer
DMA_MODE_REPEATED_ONE_SHOT	repeated one-shot transfer
DMA_MODE_CONTINUOUS	continuous transfer
DMA_MODE_REPEATED_CONTINUOUS	repeated continuous transfer
DMA_MODE_ONE_SHOT_PING_PONG_ENABLED	one-shot transfer with ping-pong buffers
DMA_MODE_ONE_SHOT_PING_PONG_DISABLED	one-shot transfer without the ping-pong buffers
DMA_MODE_CONTINUOUS_PING_PONG_DISABLED	continuous transfer without the ping-pong buffers
DMA_MODE_CONTINUOUS_PING_PONG_ENABLED	continuous transfer with ping-pong buffers

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.12.19 DMA_TRIGGER_SOURCE Enumeration**C**

```
typedef enum {
    DMA_TRIGGER_TIMER_CORE,
    DMA_TRIGGER_SOFTWARE_0,
    DMA_TRIGGER_SOFTWARE_1,

```

DMA_TRIGGER_EXTERNAL_0,
DMA_TRIGGER_EXTERNAL_1,
DMA_TRIGGER_EXTERNAL_2,
DMA_TRIGGER_EXTERNAL_3,
DMA_TRIGGER_EXTERNAL_4,
DMA_TRIGGER_TIMER_1,
DMA_TRIGGER_TIMER_2,
DMA_TRIGGER_TIMER_3,
DMA_TRIGGER_TIMER_4,
DMA_TRIGGER_TIMER_5,
DMA_TRIGGER_INPUT_CAPTURE_1,
DMA_TRIGGER_INPUT_CAPTURE_2,
DMA_TRIGGER_INPUT_CAPTURE_3,
DMA_TRIGGER_INPUT_CAPTURE_4,
DMA_TRIGGER_INPUT_CAPTURE_5,
DMA_TRIGGER_INPUT_CAPTURE_1_ERROR,
DMA_TRIGGER_INPUT_CAPTURE_2_ERROR,
DMA_TRIGGER_INPUT_CAPTURE_3_ERROR,
DMA_TRIGGER_INPUT_CAPTURE_4_ERROR,
DMA_TRIGGER_INPUT_CAPTURE_5_ERROR,
DMA_TRIGGER_OUTPUT_COMPARE_1,
DMA_TRIGGER_OUTPUT_COMPARE_2,
DMA_TRIGGER_OUTPUT_COMPARE_3,
DMA_TRIGGER_OUTPUT_COMPARE_4,
DMA_TRIGGER_OUTPUT_COMPARE_5,
DMA_TRIGGER_SPI_1_ERROR,
DMA_TRIGGER_SPI_1_RECEIVE,
DMA_TRIGGER_SPI_1_TRANSMIT,
DMA_TRIGGER_SPI_1A_ERROR,
DMA_TRIGGER_SPI_1A_RECEIVE,
DMA_TRIGGER_SPI_1A_TRANSMIT,
DMA_TRIGGER_SPI_2_ERROR,
DMA_TRIGGER_SPI_2_RECEIVE,
DMA_TRIGGER_SPI_2_TRANSMIT,
DMA_TRIGGER_SPI_2A_ERROR,
DMA_TRIGGER_SPI_2A_RECEIVE,
DMA_TRIGGER_SPI_2A_TRANSMIT,
DMA_TRIGGER_SPI_3_ERROR,
DMA_TRIGGER_SPI_3_RECEIVE,
DMA_TRIGGER_SPI_3_TRANSMIT,
DMA_TRIGGER_SPI_3A_ERROR,
DMA_TRIGGER_SPI_3A_RECEIVE,
DMA_TRIGGER_SPI_3A_TRANSMIT,
DMA_TRIGGER_SPI_4_ERROR,
DMA_TRIGGER_SPI_4_RECEIVE,
DMA_TRIGGER_SPI_4_TRANSMIT,
DMA_TRIGGER_I2C_1_ERROR,
DMA_TRIGGER_I2C_1_SLAVE,
DMA_TRIGGER_I2C_1_MASTER,
DMA_TRIGGER_I2C_1A_ERROR,
DMA_TRIGGER_I2C_1A_SLAVE,
DMA_TRIGGER_I2C_1A_MASTER,
DMA_TRIGGER_I2C_2_ERROR,
DMA_TRIGGER_I2C_2_SLAVE,
DMA_TRIGGER_I2C_2_MASTER,
DMA_TRIGGER_I2C_2A_ERROR,
DMA_TRIGGER_I2C_2A_SLAVE,
DMA_TRIGGER_I2C_2A_MASTER,
DMA_TRIGGER_I2C_3_ERROR,
DMA_TRIGGER_I2C_3_SLAVE,
DMA_TRIGGER_I2C_3_MASTER,
DMA_TRIGGER_I2C_3A_ERROR,
DMA_TRIGGER_I2C_3A_SLAVE,
DMA_TRIGGER_I2C_3A_MASTER,
DMA_TRIGGER_I2C_4_ERROR,
DMA_TRIGGER_I2C_4_SLAVE,

```
DMA_TRIGGER_I2C_4_MASTER,  
DMA_TRIGGER_I2C_5_ERROR,  
DMA_TRIGGER_I2C_5_SLAVE,  
DMA_TRIGGER_I2C_5_MASTER,  
DMA_TRIGGER_USART_1_ERROR,  
DMA_TRIGGER_USART_1_RECEIVE,  
DMA_TRIGGER_USART_1_TRANSMIT,  
DMA_TRIGGER_USART_1A_ERROR,  
DMA_TRIGGER_USART_1A_RECEIVE,  
DMA_TRIGGER_USART_1A_TRANSMIT,  
DMA_TRIGGER_USART_1B_ERROR,  
DMA_TRIGGER_USART_1B_RECEIVE,  
DMA_TRIGGER_USART_1B_TRANSMIT,  
DMA_TRIGGER_USART_2_ERROR,  
DMA_TRIGGER_USART_2_RECEIVE,  
DMA_TRIGGER_USART_2_TRANSMIT,  
DMA_TRIGGER_USART_2A_ERROR,  
DMA_TRIGGER_USART_2A_RECEIVE,  
DMA_TRIGGER_USART_2A_TRANSMIT,  
DMA_TRIGGER_USART_2B_ERROR,  
DMA_TRIGGER_USART_2B_RECEIVE,  
DMA_TRIGGER_USART_2B_TRANSMIT,  
DMA_TRIGGER_USART_3_ERROR,  
DMA_TRIGGER_USART_3_RECEIVE,  
DMA_TRIGGER_USART_3_TRANSMIT,  
DMA_TRIGGER_USART_3A_ERROR,  
DMA_TRIGGER_USART_3A_RECEIVE,  
DMA_TRIGGER_USART_3A_TRANSMIT,  
DMA_TRIGGER_USART_3B_ERROR,  
DMA_TRIGGER_USART_3B_RECEIVE,  
DMA_TRIGGER_USART_3B_TRANSMIT,  
DMA_TRIGGER_USART_4_ERROR,  
DMA_TRIGGER_USART_4_RECEIVE,  
DMA_TRIGGER_USART_4_TRANSMIT,  
DMA_TRIGGER_USART_5_ERROR,  
DMA_TRIGGER_USART_5_RECEIVE,  
DMA_TRIGGER_USART_5_TRANSMIT,  
DMA_TRIGGER_USART_6_ERROR,  
DMA_TRIGGER_USART_6_RECEIVE,  
DMA_TRIGGER_USART_6_TRANSMIT,  
DMA_TRIGGER_CHANGE_NOTICE,  
DMA_TRIGGER_CHANGE_NOTICE_A,  
DMA_TRIGGER_CHANGE_NOTICE_B,  
DMA_TRIGGER_CHANGE_NOTICE_C,  
DMA_TRIGGER_CHANGE_NOTICE_D,  
DMA_TRIGGER_CHANGE_NOTICE_E,  
DMA_TRIGGER_CHANGE_NOTICE_F,  
DMA_TRIGGER_CHANGE_NOTICE_G,  
DMA_TRIGGER_DMA_0,  
DMA_TRIGGER_DMA_1,  
DMA_TRIGGER_DMA_2,  
DMA_TRIGGER_DMA_3,  
DMA_TRIGGER_DMA_4,  
DMA_TRIGGER_DMA_5,  
DMA_TRIGGER_DMA_6,  
DMA_TRIGGER_DMA_7,  
DMA_TRIGGER_COMPARATOR_1,  
DMA_TRIGGER_COMPARATOR_2,  
DMA_TRIGGER_COMPARATOR_3,  
DMA_TRIGGER_ADC_1,  
DMA_TRIGGER_PARALLEL_PORT,  
DMA_TRIGGER_CAN_1,  
DMA_TRIGGER_CAN_2,  
DMA_TRIGGER_CLOCK_MONITOR,  
DMA_TRIGGER_RTCC,  
DMA_TRIGGER_FLASH_CONTROL,
```

```

DMA_TRIGGER_USB_1,
DMA_TRIGGER_ETH_1,
DMA_TRIGGER_PARALLEL_PORT_ERROR,
DMA_TRIGGER_CTMU
} DMA_TRIGGER_SOURCE;

```

Description

DMA trigger sources

This enumeration lists all the possible DMA trigger sources.

Members

Members	Description
DMA_TRIGGER_TIMER_CORE	Core timer
DMA_TRIGGER_SOFTWARE_0	Software 0
DMA_TRIGGER_SOFTWARE_1	Software 1
DMA_TRIGGER_EXTERNAL_0	External 0
DMA_TRIGGER_EXTERNAL_1	External 1
DMA_TRIGGER_EXTERNAL_2	External 2
DMA_TRIGGER_EXTERNAL_3	External 3
DMA_TRIGGER_EXTERNAL_4	External 4
DMA_TRIGGER_TIMER_1	Timer 1
DMA_TRIGGER_TIMER_2	Timer 2
DMA_TRIGGER_TIMER_3	Timer 3
DMA_TRIGGER_TIMER_4	Timer 4
DMA_TRIGGER_TIMER_5	Timer 5
DMA_TRIGGER_INPUT_CAPTURE_1	Input Capture 1
DMA_TRIGGER_INPUT_CAPTURE_2	Input Capture 2
DMA_TRIGGER_INPUT_CAPTURE_3	Input Capture 3
DMA_TRIGGER_INPUT_CAPTURE_4	Input Capture 4
DMA_TRIGGER_INPUT_CAPTURE_5	Input Capture 5
DMA_TRIGGER_INPUT_CAPTURE_1_ERROR	Input Capture 1 error
DMA_TRIGGER_INPUT_CAPTURE_2_ERROR	Input Capture 2 error
DMA_TRIGGER_INPUT_CAPTURE_3_ERROR	Input Capture 3 error
DMA_TRIGGER_INPUT_CAPTURE_4_ERROR	Input Capture 4 error
DMA_TRIGGER_INPUT_CAPTURE_5_ERROR	Input Capture 5 error
DMA_TRIGGER_OUTPUT_COMPARE_1	Output compare 1
DMA_TRIGGER_OUTPUT_COMPARE_2	Output compare 2
DMA_TRIGGER_OUTPUT_COMPARE_3	Output compare 3
DMA_TRIGGER_OUTPUT_COMPARE_4	Output compare 4
DMA_TRIGGER_OUTPUT_COMPARE_5	Output compare 5
DMA_TRIGGER_SPI_1_ERROR	SPI 1 error
DMA_TRIGGER_SPI_1_RECEIVE	SPI 1 receive
DMA_TRIGGER_SPI_1_TRANSMIT	SPI 1 transmit
DMA_TRIGGER_SPI_1A_ERROR	SPI 1A error
DMA_TRIGGER_SPI_1A_RECEIVE	SPI 1A receive
DMA_TRIGGER_SPI_1A_TRANSMIT	SPI 1A transmit
DMA_TRIGGER_SPI_2_ERROR	SPI 2 erro
DMA_TRIGGER_SPI_2_RECEIVE	SPI 2 receive

DMA_TRIGGER_SPI_2_TRANSMIT	SPI 2 transmit
DMA_TRIGGER_SPI_2A_ERROR	SPI 2A receive
DMA_TRIGGER_SPI_2A_RECEIVE	SPI 2A receive
DMA_TRIGGER_SPI_2A_TRANSMIT	SPI 2A transmit
DMA_TRIGGER_SPI_3_ERROR	SPI 3 error
DMA_TRIGGER_SPI_3_RECEIVE	SPI 3 receive
DMA_TRIGGER_SPI_3_TRANSMIT	SPI 3 transmit
DMA_TRIGGER_SPI_3A_ERROR	SPI 3A receive
DMA_TRIGGER_SPI_3A_RECEIVE	SPI 3A receive
DMA_TRIGGER_SPI_3A_TRANSMIT	SPI 3A receive
DMA_TRIGGER_SPI_4_ERROR	SPI 4 error
DMA_TRIGGER_SPI_4_RECEIVE	SPI 4 receive
DMA_TRIGGER_SPI_4_TRANSMIT	SPI 4 transmit
DMA_TRIGGER_I2C_1_ERROR	I2C 1 error
DMA_TRIGGER_I2C_1_SLAVE	I2C 1 slave
DMA_TRIGGER_I2C_1_MASTER	I2C 1 master
DMA_TRIGGER_I2C_1A_ERROR	I2C 1A error
DMA_TRIGGER_I2C_1A_SLAVE	I2C 1A slave
DMA_TRIGGER_I2C_1A_MASTER	I2C 1A master
DMA_TRIGGER_I2C_2_ERROR	I2C 2 error
DMA_TRIGGER_I2C_2_SLAVE	I2C 2 slave
DMA_TRIGGER_I2C_2_MASTER	I2C 2 master
DMA_TRIGGER_I2C_2A_ERROR	I2C 2A error
DMA_TRIGGER_I2C_2A_SLAVE	I2C 2A slave
DMA_TRIGGER_I2C_2A_MASTER	I2C 2A master
DMA_TRIGGER_I2C_3_ERROR	I2C 3 error
DMA_TRIGGER_I2C_3_SLAVE	I2C 3 slave
DMA_TRIGGER_I2C_3_MASTER	I2C 3 master
DMA_TRIGGER_I2C_3A_ERROR	I2C 3A error
DMA_TRIGGER_I2C_3A_SLAVE	I2C 3A slave
DMA_TRIGGER_I2C_3A_MASTER	I2C 3A master
DMA_TRIGGER_I2C_4_ERROR	I2C 4 error
DMA_TRIGGER_I2C_4_SLAVE	I2C 4 slave
DMA_TRIGGER_I2C_4_MASTER	I2C 4 master
DMA_TRIGGER_I2C_5_ERROR	I2C 5 error
DMA_TRIGGER_I2C_5_SLAVE	I2C 5 slave
DMA_TRIGGER_I2C_5_MASTER	I2C 5 master
DMA_TRIGGER_USART_1_ERROR	USART 1 error
DMA_TRIGGER_USART_1_RECEIVE	USART 1 receive
DMA_TRIGGER_USART_1_TRANSMIT	USART 1 transmit
DMA_TRIGGER_USART_1A_ERROR	USART 1A error
DMA_TRIGGER_USART_1A_RECEIVE	USART 1A receive
DMA_TRIGGER_USART_1A_TRANSMIT	USART 1A transmit
DMA_TRIGGER_USART_1B_ERROR	USART 1B error
DMA_TRIGGER_USART_1B_RECEIVE	USART 1B receive

DMA_TRIGGER_USART_1B_TRANSMIT	USART 1B transmit
DMA_TRIGGER_USART_2_ERROR	USART 2 error
DMA_TRIGGER_USART_2_RECEIVE	USART 2 receive
DMA_TRIGGER_USART_2_TRANSMIT	USART 2 transmit
DMA_TRIGGER_USART_2A_ERROR	USART 2A error
DMA_TRIGGER_USART_2A_RECEIVE	USART 2A receive
DMA_TRIGGER_USART_2A_TRANSMIT	USART 2A transmit
DMA_TRIGGER_USART_2B_ERROR	USART 2B error
DMA_TRIGGER_USART_2B_RECEIVE	USART 2B receive
DMA_TRIGGER_USART_2B_TRANSMIT	USART 2B transmit
DMA_TRIGGER_USART_3_ERROR	USART 3 error
DMA_TRIGGER_USART_3_RECEIVE	USART 3 receive
DMA_TRIGGER_USART_3_TRANSMIT	USART 3 transmit
DMA_TRIGGER_USART_3A_ERROR	USART 3A error
DMA_TRIGGER_USART_3A_RECEIVE	USART 3A receive
DMA_TRIGGER_USART_3A_TRANSMIT	USART 3A transmit
DMA_TRIGGER_USART_3B_ERROR	USART 3B error
DMA_TRIGGER_USART_3B_RECEIVE	USART 3B receive
DMA_TRIGGER_USART_3B_TRANSMIT	USART 3B transmit
DMA_TRIGGER_USART_4_ERROR	USART 4 error
DMA_TRIGGER_USART_4_RECEIVE	USART 4 receive
DMA_TRIGGER_USART_4_TRANSMIT	USART 4 transmit
DMA_TRIGGER_USART_5_ERROR	USART 5 error
DMA_TRIGGER_USART_5_RECEIVE	USART 5 receive
DMA_TRIGGER_USART_5_TRANSMIT	USART 5 transmit
DMA_TRIGGER_USART_6_ERROR	USART 6 error
DMA_TRIGGER_USART_6_RECEIVE	USART 6 receive
DMA_TRIGGER_USART_6_TRANSMIT	USART 6 transmit
DMA_TRIGGER_CHANGE_NOTICE	Change notice
DMA_TRIGGER_CHANGE_NOTICE_A	Change notice A
DMA_TRIGGER_CHANGE_NOTICE_B	Change notice B
DMA_TRIGGER_CHANGE_NOTICE_C	Change notice C
DMA_TRIGGER_CHANGE_NOTICE_D	Change notice D
DMA_TRIGGER_CHANGE_NOTICE_E	Change notice E
DMA_TRIGGER_CHANGE_NOTICE_F	Change notice F
DMA_TRIGGER_CHANGE_NOTICE_G	Change notice G
DMA_TRIGGER_DMA_0	DMA Channel 0
DMA_TRIGGER_DMA_1	DMA Channel 1
DMA_TRIGGER_DMA_2	DMA Channel 2
DMA_TRIGGER_DMA_3	DMA Channel 3
DMA_TRIGGER_DMA_4	DMA Channel 4
DMA_TRIGGER_DMA_5	DMA Channel 5
DMA_TRIGGER_DMA_6	DMA Channel 6
DMA_TRIGGER_DMA_7	DMA Channel 7
DMA_TRIGGER_COMPARATOR_1	Comparator 1

DMA_TRIGGER_COMPARATOR_2	DMA Channel 2
DMA_TRIGGER_COMPARATOR_3	DMA Channel 3
DMA_TRIGGER_ADC_1	ADC 1
DMA_TRIGGER_PARALLEL_PORT	Parallel port
DMA_TRIGGER_CAN_1	CAN 1
DMA_TRIGGER_CAN_2	CAN 2
DMA_TRIGGER_CLOCK_MONITOR	Clock monitor
DMA_TRIGGER_RTCC	RTCC
DMA_TRIGGER_FLASH_CONTROL	Flash control
DMA_TRIGGER_USB_1	USB 1
DMA_TRIGGER_ETH_1	ETH 1
DMA_TRIGGER_PARALLEL_PORT_ERROR	Parallel port error
DMA_TRIGGER_CTMU	CTMU

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.6.6.13 Feature Existence Routines

5.6.6.6.13.1 PLIB_DMA_ExistsAbortTransfer Function

C

```
bool PLIB_DMA_ExistsAbortTransfer(
    DMA_MODULE_ID index
);
```

Description

This function identifies whether the AbortTransfer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_AbortTransferSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AbortTransfer feature is supported on the device
- false - The AbortTransfer feature is not supported on the device

Remarks

None.

5.6.6.6.13.2 PLIB_DMA_ExistsBusy Function

C

```
bool PLIB_DMA_ExistsBusy(
```

```
    DMA_MODULE_ID index
);
```

Description

This function identifies whether the Busy feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_BusyActiveSet](#)
- [PLIB_DMA_BusyActiveReset](#)
- [PLIB_DMA_IsBusy](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Busy feature is supported on the device
- false - The Busy feature is not supported on the device

Remarks

None.

5.6.6.6.13.3 PLIB_DMA_ExistsChannelBits Function

C

```
bool PLIB_DMA_ExistsChannelBits(
    DMA_MODULE_ID index
);
```

Description

This function identifies whether the ChannelBits feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_ChannelBitsGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelBits feature is supported on the device
- false - The ChannelBits feature is not supported on the device

Remarks

None.

5.6.6.6.13.4 PLIB_DMA_ExistsChannelX Function

C

```
bool PLIB_DMA_ExistsChannelX(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelX feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXEnable](#)
- [PLIB_DMA_ChannelXIsEnabled](#)
- [PLIB_DMA_ChannelXDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelX feature is supported on the device
- false - The ChannelX feature is not supported on the device

Remarks

None.

5.6.6.6.13.5 PLIB_DMA_ExistsChannelXAbortIRQ Function

C

```
bool PLIB_DMA_ExistsChannelXAbortIRQ(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXAbortIRQ feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_ChannelXAbortIRQSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXAbortIRQ feature is supported on the device
- false - The ChannelXAbortIRQ feature is not supported on the device

Remarks

None.

5.6.6.6.13.6 PLIB_DMA_ExistsChannelXAuto Function**C**

```
bool PLIB_DMA_ExistsChannelXAuto(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXAuto feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXAutoEnable](#)
- [PLIB_DMA_ChannelXAutoDisable](#)
- [PLIB_DMA_ChannelXAUTOIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXAuto feature is supported on the device
- false - The ChannelXAuto feature is not supported on the device

Remarks

None.

5.6.6.6.13.7 PLIB_DMA_ExistsChannelXBusy Function**C**

```
bool PLIB_DMA_ExistsChannelXBusy(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXBusy feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXBusyActiveSet](#)
- [PLIB_DMA_ChannelXBusyInActiveSet](#)
- [PLIB_DMA_ChannelXBusyIsBusy](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXBusy feature is supported on the device
- false - The ChannelXBusy feature is not supported on the device

Remarks

None.

5.6.6.6.13.8 PLIB_DMA_ExistsChannelXCellProgressPointer Function**C**

```
bool PLIB_DMA_ExistsChannelXCellProgressPointer(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXCellProgressPointer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_ChannelXCellProgressPointerGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXCellProgressPointer feature is supported on the device
- false - The ChannelXCellProgressPointer feature is not supported on the device

Remarks

None.

5.6.6.6.13.9 PLIB_DMA_ExistsChannelXCellSize Function**C**

```
bool PLIB_DMA_ExistsChannelXCellSize(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXCellSize feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXCellSizeGet](#)
- [PLIB_DMA_ChannelXCellSizeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXCellSize feature is supported on the device
- false - The ChannelXCellSize feature is not supported on the device

Remarks

None.

5.6.6.6.13.10 PLIB_DMA_ExistsChannelXChain Function**C**

```
bool PLIB_DMA_ExistsChannelXChain(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXChain feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXChainToLower](#)
- [PLIB_DMA_ChannelXChainToHigher](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXChain feature is supported on the device
- false - The ChannelXChain feature is not supported on the device

Remarks

None.

5.6.6.6.13.11 PLIB_DMA_ExistsChannelXChainEnbl Function**C**

```
bool PLIB_DMA_ExistsChannelXChainEnbl(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXChainEnbl feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXChainEnable](#)
- [PLIB_DMA_ChannelXChainDisable](#)
- [PLIB_DMA_ChannelXChainIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXChainEnbl feature is supported on the device
- false - The ChannelXChainEnbl feature is not supported on the device

Remarks

None.

5.6.6.6.13.12 PLIB_DMA_ExistsChannelXDestinationPointer Function**C**

```
bool PLIB_DMA_ExistsChannelXDestinationPointer(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXDestinationPointer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_ChannelXDestinationPointerGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXDestinationPointer feature is supported on the device
- false - The ChannelXDestinationPointer feature is not supported on the device

Remarks

None.

5.6.6.6.13.13 PLIB_DMA_ExistsChannelXDestinationSize Function**C**

```
bool PLIB_DMA_ExistsChannelXDestinationSize(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXDestinationSize feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXDestinationSizeGet](#)
- [PLIB_DMA_ChannelXDestinationSizeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXDestinationSize feature is supported on the device
- false - The ChannelXDestinationSize feature is not supported on the device

Remarks

None.

5.6.6.6.13.14 PLIB_DMA_ExistsChannelXDestinationStartAddress Function**C**

```
bool PLIB_DMA_ExistsChannelXDestinationStartAddress(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXDestinationStartAddress feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXDestinationStartAddressGet](#)
- [PLIB_DMA_ChannelXDestinationStartAddressSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXDestinationStartAddress feature is supported on the device
- false - The ChannelXDestinationStartAddress feature is not supported on the device

Remarks

None.

5.6.6.6.13.15 PLIB_DMA_ExistsChannelXDisabled Function

C

```
bool PLIB_DMA_ExistsChannelXDisabled(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXDisabled feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXDisabledEnablesEvents](#)
- [PLIB_DMA_ChannelXDisabledDisablesEvents](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXDisabled feature is supported on the device
- false - The ChannelXDisabled feature is not supported on the device

Remarks

None.

5.6.6.6.13.16 PLIB_DMA_ExistsChannelXEvent Function

C

```
bool PLIB_DMA_ExistsChannelXEvent(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXEvent feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_ChannelXEventIsDetected](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXEvent feature is supported on the device
- false - The ChannelXEvent feature is not supported on the device

Remarks

None.

5.6.6.6.13.17 PLIB_DMA_ExistsChannelXINTSource Function

C

```
bool PLIB_DMA_ExistsChannelXINTSource(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXINTSource feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXINTSourceEnable](#)
- [PLIB_DMA_ChannelXINTSourceDisable](#)
- [PLIB_DMA_ChannelXINTSourceIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXINTSource feature is supported on the device
- false - The ChannelXINTSource feature is not supported on the device

Remarks

None.

5.6.6.6.13.18 PLIB_DMA_ExistsChannelXINTSourceFlag Function

C

```
bool PLIB_DMA_ExistsChannelXINTSourceFlag(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXINTSourceFlag feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXINTSourceFlagGet](#)
- [PLIB_DMA_ChannelXINTSourceFlagSet](#)
- [PLIB_DMA_ChannelXINTSourceFlagClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXINTSourceFlag feature is supported on the device

- false - The ChannelXINTSourceFlag feature is not supported on the device

Remarks

None.

5.6.6.6.13.19 PLIB_DMA_ExistsChannelXPatternData Function**C**

```
bool PLIB_DMA_ExistsChannelXPatternData(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXPatternData feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXPatternDataGet](#)
- [PLIB_DMA_ChannelXPatternDataSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXPatternData feature is supported on the device
- false - The ChannelXPatternData feature is not supported on the device

Remarks

None.

5.6.6.6.13.20 PLIB_DMA_ExistsChannelXPatternIgnore Function**C**

```
bool PLIB_DMA_ExistsChannelXPatternIgnore(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXPatternIgnore feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXPatternIgnoreSet](#)
- [PLIB_DMA_ChannelXPatternIgnoreGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXPatternIgnore feature is supported on the device
- false - The ChannelXPatternIgnore feature is not supported on the device

Remarks

None.

5.6.6.6.13.21 PLIB_DMA_ExistsChannelXPatternIgnoreByte Function**C**

```
bool PLIB_DMA_ExistsChannelXPatternIgnoreByte(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXPatternIgnoreByte feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXPatternIgnoreByteEnable](#)
- [PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled](#)
- [PLIB_DMA_ChannelXPatternIgnoreByteDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXPatternIgnoreByte feature is supported on the device
- false - The ChannelXPatternIgnoreByte feature is not supported on the device

Remarks

None.

5.6.6.6.13.22 PLIB_DMA_ExistsChannelXPatternLength Function**C**

```
bool PLIB_DMA_ExistsChannelXPatternLength(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXPatternLength feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXPatternLengthSet](#)
- [PLIB_DMA_ChannelXPatternLengthGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXPatternLength feature is supported on the device
- false - The ChannelXPatternLength feature is not supported on the device

Remarks

None.

5.6.6.6.13.23 PLIB_DMA_ExistsChannelXPriority Function**C**

```
bool PLIB_DMA_ExistsChannelXPriority(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXPriority feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXPrioritySelect](#)
- [PLIB_DMA_ChannelXPriorityGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXPriority feature is supported on the device
- false - The ChannelXPriority feature is not supported on the device

Remarks

None.

5.6.6.6.13.24 PLIB_DMA_ExistsChannelXSourcePointer Function**C**

```
bool PLIB_DMA_ExistsChannelXSourcePointer(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXSourcePointer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_ChannelXSourcePointerGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXSourcePointer feature is supported on the device
- false - The ChannelXSourcePointer feature is not supported on the device

Remarks

None.

5.6.6.6.13.25 PLIB_DMA_ExistsChannelXSourceSize Function**C**

```
bool PLIB_DMA_ExistsChannelXSourceSize(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXSourceSize feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXSourceSizeGet](#)
- [PLIB_DMA_ChannelXSourceSizeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXSourceSize feature is supported on the device
- false - The ChannelXSourceSize feature is not supported on the device

Remarks

None.

5.6.6.6.13.26 PLIB_DMA_ExistsChannelXSourceStartAddress Function**C**

```
bool PLIB_DMA_ExistsChannelXSourceStartAddress(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXSourceStartAddress feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXSourceStartAddressGet](#)
- [PLIB_DMA_ChannelXSourceStartAddressSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXSourceStartAddress feature is supported on the device
- false - The ChannelXSourceStartAddress feature is not supported on the device

Remarks

None.

5.6.6.6.13.27 PLIB_DMA_ExistsChannelXStartIRQ Function

C

```
bool PLIB_DMA_ExistsChannelXStartIRQ(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXStartIRQ feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_ChannelXStartIRQSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXStartIRQ feature is supported on the device
- false - The ChannelXStartIRQ feature is not supported on the device

Remarks

None.

5.6.6.6.13.28 PLIB_DMA_ExistsChannelXTrigger Function

C

```
bool PLIB_DMA_ExistsChannelXTrigger(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the ChannelXTrigger feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_ChannelXTriggerEnable](#)
- [PLIB_DMA_ChannelXTriggerIsEnabled](#)
- [PLIB_DMA_ChannelXTriggerDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChannelXTrigger feature is supported on the device
- false - The ChannelXTrigger feature is not supported on the device

Remarks

None.

5.6.6.6.13.29 PLIB_DMA_ExistsCRC Function**C**

```
bool PLIB_DMA_ExistsCRC(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRC feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_CRCEnable](#)
- [PLIB_DMA_CRCDisable](#)
- [PLIB_DMA_CRCIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRC feature is supported on the device
- false - The CRC feature is not supported on the device

Remarks

None.

5.6.6.6.13.30 PLIB_DMA_ExistsCRCAppendMode Function**C**

```
bool PLIB_DMA_ExistsCRCAppendMode(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRCAppendMode feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_CRCAppendModeEnable](#)
- [PLIB_DMA_CRCAppendModeDisable](#)
- [PLIB_DMA_CRCAppendModeIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRCAppendMode feature is supported on the device
- false - The CRCAppendMode feature is not supported on the device

Remarks

None.

5.6.6.6.13.31 PLIB_DMA_ExistsCRCBitOrder Function**C**

```
bool PLIB_DMA_ExistsCRCBitOrder(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRCBitOrder feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_CRCBitOrderSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRCBitOrder feature is supported on the device
- false - The CRCBitOrder feature is not supported on the device

Remarks

None.

5.6.6.6.13.32 PLIB_DMA_ExistsCRCByteOrder Function

C

```
bool PLIB_DMA_ExistsCRCByteOrder(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRCByteOrder feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_CRCByteOrderSelect](#)
- [PLIB_DMA_CRCByteOrderGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRCByteOrder feature is supported on the device
- false - The CRCByteOrder feature is not supported on the device

Remarks

None.

5.6.6.6.13.33 PLIB_DMA_ExistsCRCChannel Function

C

```
bool PLIB_DMA_ExistsCRCChannel(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRCChannel feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_CRCChannelSelect](#)
- [PLIB_DMA_CRCChannelGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRCChannel feature is supported on the device
- false - The CRCChannel feature is not supported on the device

Remarks

None.

5.6.6.6.13.34 PLIB_DMA_ExistsCRCDData Function**C**

```
bool PLIB_DMA_ExistsCRCDData(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRCDData feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_CRCDDataRead](#)
- [PLIB_DMA_CRCDDataWrite](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRCDData feature is supported on the device
- false - The CRCDData feature is not supported on the device

Remarks

None.

5.6.6.6.13.35 PLIB_DMA_ExistsCRCPolynomialLength Function**C**

```
bool PLIB_DMA_ExistsCRCPolynomialLength(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRCPolynomialLength feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_CRCPolynomialLengthSet](#)
- [PLIB_DMA_CRCPolynomialLengthGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRCPolynomialLength feature is supported on the device
- false - The CRCPolynomialLength feature is not supported on the device

Remarks

None.

5.6.6.6.13.36 PLIB_DMA_ExistsCRCType Function**C**

```
bool PLIB_DMA_ExistsCRCType(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRCType feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_CRCTypeGet](#)
- [PLIB_DMA_CRCTypeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRCType feature is supported on the device
- false - The CRCType feature is not supported on the device

Remarks

None.

5.6.6.6.13.37 PLIB_DMA_ExistsCRCWriteByteOrder Function**C**

```
bool PLIB_DMA_ExistsCRCWriteByteOrder(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRCWriteByteOrder feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_CRCWriteByteOrderAlter](#)
- [PLIB_DMA_CRCWriteByteOrderMaintain](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRCWriteByteOrder feature is supported on the device
- false - The CRCWriteByteOrder feature is not supported on the device

Remarks

None.

5.6.6.6.13.38 PLIB_DMA_ExistsCRCXOREnable Function**C**

```
bool PLIB_DMA_ExistsCRCXOREnable(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the CRCXOREnable feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_CRCXOREnableSet](#)
- [PLIB_DMA_CRCXOREnableGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CRCXOREnable feature is supported on the device
- false - The CRCXOREnable feature is not supported on the device

Remarks

None.

5.6.6.6.13.39 PLIB_DMA_ExistsEnableControl Function**C**

```
bool PLIB_DMA_ExistsEnableControl(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_Enable](#)
- [PLIB_DMA_Disable](#)

- [PLIB_DMA_IsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Remarks

None.

5.6.6.6.13.40 PLIB_DMA_ExistsLastBusAccess Function**C**

```
bool PLIB_DMA_ExistsLastBusAccess(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the LastBusAccess feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_LastBusAccessIsRead](#)
- [PLIB_DMA_LastBusAccessIsWrite](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LastBusAccess feature is supported on the device
- false - The LastBusAccess feature is not supported on the device

Remarks

None.

5.6.6.6.13.41 PLIB_DMA_ExistsRecentAddress Function**C**

```
bool PLIB_DMA_ExistsRecentAddress(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the RecentAddress feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_RecentAddressAccessed](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RecentAddress feature is supported on the device
- false - The RecentAddress feature is not supported on the device

Remarks

None.

5.6.6.6.13.42 PLIB_DMA_ExistsStartTransfer Function**C**

```
bool PLIB_DMA_ExistsStartTransfer(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the StartTransfer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- [PLIB_DMA_StartTransferSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StartTransfer feature is supported on the device
- false - The StartTransfer feature is not supported on the device

Remarks

None.

5.6.6.6.13.43 PLIB_DMA_ExistsStopInIdle Function**C**

```
bool PLIB_DMA_ExistsStopInIdle(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_StopInIdleEnable](#)

- [PLIB_DMA_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Remarks

None.

5.6.6.6.13.44 PLIB_DMA_ExistsSuspend Function

C

```
bool PLIB_DMA_ExistsSuspend(  
    DMA_MODULE_ID index  
);
```

Description

This function identifies whether the Suspend feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- [PLIB_DMA_SuspendEnable](#)
- [PLIB_DMA_SuspendDisable](#)
- [PLIB_DMA_SuspendIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Suspend feature is supported on the device
- false - The Suspend feature is not supported on the device

Remarks

None.

5.6.6.7 Files

Files

Name	Description
plib_dma.h	Defines the DMA Peripheral Library interface functions.

Description

5.6.6.7.1 plib_dma.h

DMA Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Direct Memory Access (DMA) Peripheral Library for Microchip microcontrollers. The definitions in this file are for the DMA module.

Functions

	Name	Description
≡	<u>PLIB_DMA_AbortTransferSet</u>	Aborts transfer on the specified channel.
≡	<u>PLIB_DMA_BusyActiveReset</u>	Resets the BUSY bit of the DMA controller.
≡	<u>PLIB_DMA_BusyActiveSet</u>	Sets the BUSY bit of the DMA controller.
≡	<u>PLIB_DMA_ChannelBitsGet</u>	Returns the DMA channel bits.
≡	<u>PLIB_DMA_ChannelPriorityGet</u>	Gets the priority scheme of the DMA channels.
≡	<u>PLIB_DMA_ChannelPrioritySelect</u>	Sets the priority scheme of the DMA channels.
≡	<u>PLIB_DMA_ChannelXAbortIRQSet</u>	Sets the IRQ to abort the DMA transfer on the specified channel.
≡	<u>PLIB_DMA_ChannelXAddressModeGet</u>	Gets the channel address mode.
≡	<u>PLIB_DMA_ChannelXAddressModeSelect</u>	Sets the channel address mode.
≡	<u>PLIB_DMA_ChannelXAutoDisable</u>	Channel is disabled after a block transfer is complete.
≡	<u>PLIB_DMA_ChannelXAutoEnable</u>	Channel is continuously enabled.
≡	<u>PLIB_DMA_ChannelXAutsEnabled</u>	Returns the channel automatic enable status.
≡	<u>PLIB_DMA_ChannelXBufferedDataIsWritten</u>	Returns the buffered data write status for the specified channel.
≡	<u>PLIB_DMA_ChannelXBusyActiveSet</u>	Sets the Busy bit to active.
≡	<u>PLIB_DMA_ChannelXBusyInactiveSet</u>	Sets the Busy bit to inactive.
≡	<u>PLIB_DMA_ChannelXBusyIsBusy</u>	Returns the busy status of the specified channel.
≡	<u>PLIB_DMA_ChannelXCellProgressPointerGet</u>	Returns the number of bytes transferred since the last event.
≡	<u>PLIB_DMA_ChannelXCellSizeGet</u>	Reads the cell size (in bytes) configured for the specified channel.
≡	<u>PLIB_DMA_ChannelXCellSizeSet</u>	Writes the specified cell size into the register corresponding to the specified channel.
≡	<u>PLIB_DMA_ChannelXChainDisable</u>	Disables the channel chaining for the specified DMA channel.
≡	<u>PLIB_DMA_ChannelXChainEnable</u>	Channel chain feature is enabled.
≡	<u>PLIB_DMA_ChannelXChainIsEnabled</u>	Returns the chain status of the specified channel.
≡	<u>PLIB_DMA_ChannelXChainToHigher</u>	Chains the specified channel to a channel higher in natural priority.
≡	<u>PLIB_DMA_ChannelXChainToLower</u>	Chains the specified channel to a channel lower in natural priority.
≡	<u>PLIB_DMA_ChannelXCollisionStatus</u>	Returns the status of the specified collision type for the specified channel.
≡	<u>PLIB_DMA_ChannelXDataSizeGet</u>	Returns the current data size for the specified channel.
≡	<u>PLIB_DMA_ChannelXDataSizeSelect</u>	Selects the data size for the specified channel.
≡	<u>PLIB_DMA_ChannelXDestinationAddressModeGet</u>	Gets the source address mode of the specified channel.

	PLIB_DMA_ChannelXDestinationAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXDestinationPointerGet	Reads the current byte of the destination being pointed to for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeGet	Reads the destination size configured for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeSet	Writes the specified destination size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXDestinationStartAddressGet	Reads the destination start address configured for the specified channel.
	PLIB_DMA_ChannelXDestinationStartAddressSet	Writes the specified destination start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXDisable	Disable the specified channel.
	PLIB_DMA_ChannelXDisabledDisablesEvents	Channel start/abort events will be ignored even if the channel is disabled.
	PLIB_DMA_ChannelXDisabledEnablesEvents	Channel start/abort events will be registered even if the channel is disabled.
	PLIB_DMA_ChannelXEnable	Enable the specified channel.
	PLIB_DMA_ChannelXEventIsDetected	Returns the event status on the specified channel.
	PLIB_DMA_ChannelXINTSourceDisable	Disables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceEnable	Enables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagClear	Clears the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagGet	Returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagSet	Sets the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceIsEnabled	Returns the enable status of the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXIIsEnabled	Return the enable status of the specified channel.
	PLIB_DMA_ChannelXNullWriteModeDisable	Disables the Null Write mode.
	PLIB_DMA_ChannelXNullWriteModeEnable	Enables the Null Write mode.
	PLIB_DMA_ChannelXNullWriteModeIsEnabled	Returns the enable status of the Null Write mode for the specified channel.
	PLIB_DMA_ChannelXOperatingTransferModeGet	Returns the current transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXOperatingTransferModeSelect	Selects the transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXPatternDataGet	Returns the pattern matching (for DMA abort) data programmed for the specified channel.
	PLIB_DMA_ChannelXPatternDataSet	Writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXPatternIgnoreByteDisable	Disables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteEnable	Enables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled	Returns the state of the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreGet	Returns the pattern match ignore value.
	PLIB_DMA_ChannelXPatternIgnoreSet	Sets the pattern match ignore value.
	PLIB_DMA_ChannelXPatternLengthGet	Returns the pattern match length.
	PLIB_DMA_ChannelXPatternLengthSet	Sets the pattern match length.

	PLIB_DMA_ChannelXPeripheralAddressGet	Gets the peripheral address configured for the specified channel.
	PLIB_DMA_ChannelXPeripheralAddressSet	Sets the peripheral address for the specified channel.
	PLIB_DMA_ChannelXPingPongModeGet	Returns the Ping-Pong mode status for the specified channel.
	PLIB_DMA_ChannelXPriorityGet	Gets the priority of the specified channel.
	PLIB_DMA_ChannelXPrioritySelect	Sets the priority of the specified channel.
	PLIB_DMA_ChannelXReloadDisable	Disables reloading of the address and count registers.
	PLIB_DMA_ChannelXReloadEnable	Enables reloading of the address and count registers.
	PLIB_DMA_ChannelXReloadIsEnabled	Returns the address and count registers reload enable status.
	PLIB_DMA_ChannelXSourceAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXSourceAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXSourcePointerGet	Reads the current byte of the source being pointed to for the specified channel.
	PLIB_DMA_ChannelXSourceSizeGet	Reads the source size configured for the specified channel.
	PLIB_DMA_ChannelXSourceSizeSet	Writes the specified source size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressGet	Reads the source start address configured for the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressSet	Writes the specified source start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXStartAddressOffsetGet	Gets the primary/secondary start address (DPSRAM) offset.
	PLIB_DMA_ChannelXStartAddressOffsetSet	Sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.
	PLIB_DMA_ChannelXStartIRQSet	Sets the IRQ to initiate the DMA transfer on the specified channel.
	PLIB_DMA_ChannelXTransferCountGet	Gets the DMA data transfer count that is programmed for the specified channel.
	PLIB_DMA_ChannelXTransferCountSet	Sets the DMA data transfer count for the specified channel.
	PLIB_DMA_ChannelXTransferDirectionGet	Returns the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXTransferDirectionSelect	Selects the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXTriggerDisable	Disables the DMA transfer abort via a matching interrupt (specified by the IRQ).
	PLIB_DMA_ChannelXTriggerEnable	Enables the specified DMA channel trigger.
	PLIB_DMA_ChannelXTriggerIsEnabled	Returns the enable status of the specified DMA transfer/abort trigger.
	PLIB_DMA_CRCAppendModeDisable	Disables the CRC append mode.
	PLIB_DMA_CRCAppendModeEnable	Enables the CRC append mode.
	PLIB_DMA_CRCAppendModeIsEnabled	Gets the enable status of the CRC append mode.
	PLIB_DMA_CRCBitOrderSelect	Selects the bit order for checksum calculation.
	PLIB_DMA_CRCByteOrderGet	Gets the current byte order selected by the DMA module CRC feature.
	PLIB_DMA_CRCByteOrderSelect	Selects the byte order.
	PLIB_DMA_CRCChannelGet	Returns the current DMA channel to which the CRC is assigned.
	PLIB_DMA_CRCChannelSelect	Assigns the CRC to the specified DMA channel.
	PLIB_DMA_CRCDataRead	Reads the contents of the DMA CRC data register.

	PLIB_DMA_CRCDataWrite	Writes the contents of the DMA CRC data register with the specified data.
	PLIB_DMA_CRCDisable	Disables the DMA module CRC feature.
	PLIB_DMA_CRCEnable	Enables the DMA module CRC feature.
	PLIB_DMA_CRCIsEnabled	Gets the enable status of the CRC feature.
	PLIB_DMA_CRCPolynomialLengthGet	Gets the current polynomial length.
	PLIB_DMA_CRCPolynomialLengthSet	Selects the polynomial length.
	PLIB_DMA_CRCTypeGet	Gets the current DMA module CRC feature type.
	PLIB_DMA_CRCTypeSet	Selects the DMA module CRC feature type.
	PLIB_DMA_CRCWriteByteOrderAlter	The source data is written to the destination reordered as defined by the BYTO<1:0> bits.
	PLIB_DMA_CRCWriteByteOrderMaintain	The source data is written to the destination unaltered.
	PLIB_DMA_CRCXOREnableGet	Reads the CRC XOR register.
	PLIB_DMA_CRCXOREnableSet	Writes to the CRC XOR enable register as per the specified enable mask.
	PLIB_DMA_Disable	DMA module is disabled.
	PLIB_DMA_Enable	DMA module is enabled.
	PLIB_DMA_ExistsAbortTransfer	Identifies whether the AbortTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsBusy	Identifies whether the Busy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelBits	Identifies whether the ChannelBits feature exists on the DMA module.
	PLIB_DMA_ExistsChannelX	Identifies whether the ChannelX feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAbortIRQ	Identifies whether the ChannelXAbortIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAuto	Identifies whether the ChannelXAuto feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXBusy	Identifies whether the ChannelXBusy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellProgressPointer	Identifies whether the ChannelXCellProgressPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellSize	Identifies whether the ChannelXCellSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChain	Identifies whether the ChannelXChain feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChainEnbl	Identifies whether the ChannelXChainEnbl feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationPointer	Identifies whether the ChannelXDestinationPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationSize	Identifies whether the ChannelXDestinationSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationStartAddress	Identifies whether the ChannelXDestinationStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDisabled	Identifies whether the ChannelXDisabled feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXEvent	Identifies whether the ChannelXEvent feature exists on the DMA module.

	PLIB_DMA_ExistsChannelXINTSource	Identifies whether the ChannelXINTSource feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSourceFlag	Identifies whether the ChannelXINTSourceFlag feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternData	Identifies whether the ChannelXPatternData feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternIgnore	Identifies whether the ChannelXPatternIgnore feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternIgnoreByte	Identifies whether the ChannelXPatternIgnoreByte feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternLength	Identifies whether the ChannelXPatternLength feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPriority	Identifies whether the ChannelXPriority feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourcePointer	Identifies whether the ChannelXSourcePointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceSize	Identifies whether the ChannelXSourceSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceStartAddress	Identifies whether the ChannelXSourceStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXStartIRQ	Identifies whether the ChannelXStartIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXTrigger	Identifies whether the ChannelXTrigger feature exists on the DMA module.
	PLIB_DMA_ExistsCRC	Identifies whether the CRC feature exists on the DMA module.
	PLIB_DMA_ExistsCRCAppendMode	Identifies whether the CRCAppendMode feature exists on the DMA module.
	PLIB_DMA_ExistsCRCBitOrder	Identifies whether the CRCBitOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCByteOrder	Identifies whether the CRCByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCChannel	Identifies whether the CRCChannel feature exists on the DMA module.
	PLIB_DMA_ExistsCRCDData	Identifies whether the CRCDData feature exists on the DMA module.
	PLIB_DMA_ExistsCRCPolynomialLength	Identifies whether the CRCPolynomialLength feature exists on the DMA module.
	PLIB_DMA_ExistsCRCType	Identifies whether the CRCType feature exists on the DMA module.
	PLIB_DMA_ExistsCRCWriteByteOrder	Identifies whether the CRCWriteByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCXOREnable	Identifies whether the CRCXOREnable feature exists on the DMA module.
	PLIB_DMA_ExistsEnableControl	Identifies whether the EnableControl feature exists on the DMA module.
	PLIB_DMA_ExistsLastBusAccess	Identifies whether the LastBusAccess feature exists on the DMA module.
	PLIB_DMA_ExistsRecentAddress	Identifies whether the RecentAddress feature exists on the DMA module.
	PLIB_DMA_ExistsStartTransfer	Identifies whether the StartTransfer feature exists on the DMA module.

	PLIB_DMA_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the DMA module.
	PLIB_DMA_ExistsSuspend	Identifies whether the Suspend feature exists on the DMA module.
	PLIB_DMA_IsBusy	Gets the BUSY bit of the DMA controller.
	PLIB_DMA_IsEnabled	Returns the DMA module enable status.
	PLIB_DMA_LastBusAccessIsRead	Returns true if the last DMA bus access was a read.
	PLIB_DMA_LastBusAccessIsWrite	Returns true if the last DMA bus access was a write.
	PLIB_DMA_LastChannelActive	Returns the last DMA channel that participated in the data transfer.
	PLIB_DMA_RecentAddressAccessed	Returns the address of the most recent DMA access.
	PLIB_DMA_StartTransferSet	Initiates transfer on the specified channel.
	PLIB_DMA_StopInIdleDisable	DMA transfers continue during Idle mode.
	PLIB_DMA_StopInIdleEnable	DMA transfers are halted during Idle mode.
	PLIB_DMA_SuspendDisable	DMA suspend is disabled and the DMA module operates normally.
	PLIB_DMA_SuspendEnable	DMA transfers are suspended to allow uninterrupted access by the CPU to the data bus.
	PLIB_DMA_SuspendIsEnabled	Returns the DMA suspend status.

File Name

plib_dma.h

Company

Microchip Technology Inc.

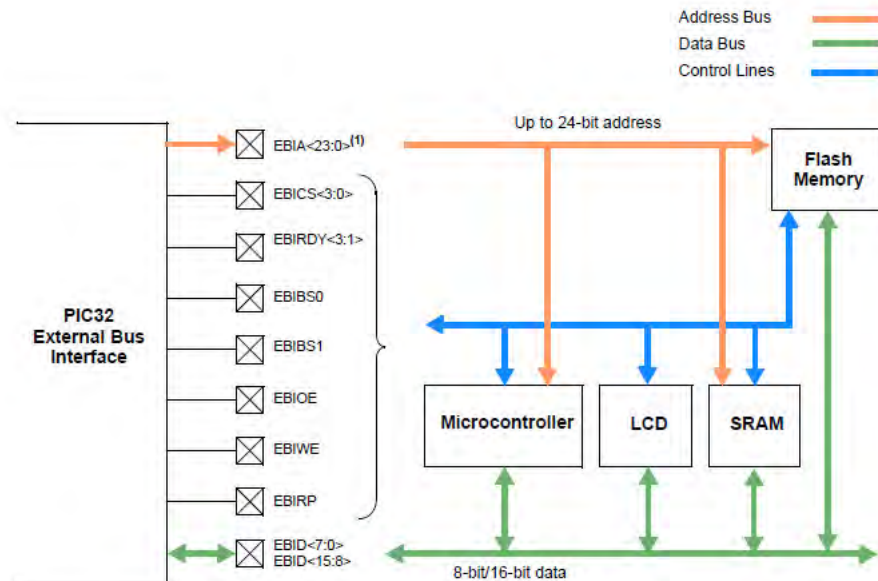
5.6.7 EBI Peripheral Library

5.6.7.1 Introduction

EBI Peripheral Library

for

Microchip Microcontrollers



Note 1: No EBIA address pins are available on 64-pin devices and the EBIA<23:20> address pins are only available on 144-pin devices.

This library provides a low-level abstraction of the EBI Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The External Bus Interface (EBI) allows for external physical memory to be mapped into the kernel's virtual memory map. The External Bus Interface (EBI) module provides a high-speed, convenient way to interface

external parallel memory devices to the PIC32 family device. With the EBI module, it is possible to **connect** asynchronous SRAM and NOR Flash devices, as well as non-memory devices such as camera sensors. The module also supports Low-Cost Controllerless (LCC) Graphics devices.

5.6.7.2 Release Notes

MPLAB Harmony Version: v0.70b **EBI Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.7.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.7.4 Using the Library

This topic describes the basic architecture of the EBI Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_ebi.h](#)

The interface to the EBI Peripheral library is defined in the "[plib_ebi.h](#)" header file. This file is included by the "peripheral.h" file. Any C language source (.c) file that uses the EBI Peripheral library should include "[plib_ebi.h](#)".

Library File:

The EBI Peripheral library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the Peripheral interacts with the framework.

5.6.7.5 How the Library Works

Before using the EBI module, it must be enabled by the application developer(s). To work correctly, the device characteristics of the attached Memory/Flash must be set with the functions in this library. The Characteristics of the Memory/Flash will be found in the specific device data sheet.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

5.6.7.6 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.7.7 Library Interface

Existence Functions

	Name	Description
	PLIB_EBI_ExistsAddressHoldTime	Identifies whether the AddressHoldTime feature exists on the EBI module
	PLIB_EBI_ExistsAddressPinEnableBits	Identifies whether the AddressPinEnableBits feature exists on the EBI module

	PLIB_EBI_ExistsAddressSetupTime	Identifies whether the AddressSetupTime feature exists on the EBI module
	PLIB_EBI_ExistsBaseAddress	Identifies whether the Base_Address feature exists on the EBI module
	PLIB_EBI_ExistsByteSelectPin	Identifies whether the ByteSelectPin feature exists on the EBI module
	PLIB_EBI_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the EBI module
	PLIB_EBI_ExistsControlEnable	Identifies whether the ControlEnable feature exists on the EBI module
	PLIB_EBI_ExistsDataEnable	Identifies whether the DataEnable feature exists on the EBI module
	PLIB_EBI_ExistsDataTurnAroundTime	Identifies whether the DataTurnAroundTime feature exists on the EBI module
	PLIB_EBI_ExistsFlashPowerDownMode	Identifies whether the FlashPowerDownMode feature exists on the EBI module
	PLIB_EBI_ExistsFlashResetPin	Identifies whether the FlashResetPin feature exists on the EBI module
	PLIB_EBI_ExistsFlashTiming	Identifies whether the FlashTiming feature exists on the EBI module
	PLIB_EBI_ExistsMemoryCharacteristics	Identifies whether the MemoryCharacteristics feature exists on the EBI module
	PLIB_EBI_ExistsMemoryPaging	Identifies whether the MemoryPaging feature exists on the EBI module
	PLIB_EBI_ExistsMemoryTimingConfig	Identifies whether the MemoryTimingConfig feature exists on the EBI module
	PLIB_EBI_ExistsPageMode	Identifies whether the PageMode feature exists on the EBI module
	PLIB_EBI_ExistsPageReadTime	Identifies whether the PageReadTime feature exists on the EBI module
	PLIB_EBI_ExistsReadCycleTime	Identifies whether the ReadCycleTime feature exists on the EBI module
	PLIB_EBI_ExistsReadyMode	Identifies whether the ReadyMode feature exists on the EBI module
	PLIB_EBI_ExistsReadyPin1Config	Identifies whether the ReadyPin1Config feature exists on the EBI module
	PLIB_EBI_ExistsReadyPin2Config	Identifies whether the ReadyPin2Config feature exists on the EBI module
	PLIB_EBI_ExistsReadyPin3Config	Identifies whether the ReadyPin3Config feature exists on the EBI module
	PLIB_EBI_ExistsReadyPinSens	Identifies whether the ReadyPinSens feature exists on the EBI module
	PLIB_EBI_ExistsStaticMemoryWidthRegister	Identifies whether the StaticMemoryWidthRegister feature exists on the EBI module
	PLIB_EBI_ExistsWriteOutputControl	Identifies whether the WriteOutputControl feature exists on the EBI module
	PLIB_EBI_ExistsWritePulseWidth	Identifies whether the WritePulseWidth feature exists on the EBI module

General Functions

	Name	Description
	PLIB_EBI_ControlEnableClear	This function gets the set power down state.

Get Functions

	Name	Description
	PLIB_EBI_AddressHoldTimeGet	This function returns the Address Hold Time

	PLIB_EBI_AddressSetupTimeGet	This function is to get the set hold time
	PLIB_EBI_BaseAddressGet	This function returns the Base Address set for each chip select
	PLIB_EBI_ControlEnableGet	This function gets the status of the enable bit for EBI.
	PLIB_EBI_DataTurnAroundTimeGet	This function gets the data turn time set for the EBI Bus.
	PLIB_EBI_FlashPowerDownModeGet	This function gets the set power down state.
	PLIB_EBI_FlashResetPinGet	This function sets the the control use of Ready pin and the inverts status
	PLIB_EBI_FlashTimingGet	This function gets the set flash timing for external flash
	PLIB_EBI_PageModeGet	This function returns Paging mode settings
	PLIB_EBI_PageReadCycleTimeGet	This function returns the cycle time for a page read.
	PLIB_EBI_PageSizeGet	This function returns Paging mode settings
	PLIB_EBI_ReadCycleTimeGet	This function returns the read time in number of clock cycles
	PLIB_EBI_ReadyModeGet	This function returns whether or not the Ready Mode was set.
	PLIB_EBI_StaticMemoryWidthRegisterGet	This function get the width of the set data bus
	PLIB_EBI_WritePulseWidthGet	This function returns the set hold time in clock cycles.

Set Functions

	Name	Description
	PLIB_EBI_AddressPinEnableBitsSet	This function sets the address pins used for EBI.
	PLIB_EBI_BaseAddressSet	This function sets the Base address for physical memory at each chip select
	PLIB_EBI_ByteSelectPinSet	This function sets the data Byte Select High [15:8] and Low [7:0] enable pins for use.
	PLIB_EBI_ChipSelectEnableSet	This function sets the Chip Select pins for use with the EBI or GPIO.
	PLIB_EBI_DataEnableSet	This function sets the use of Data Byte Select Pins, High and Low, for control with EBI or GPIO use.
	PLIB_EBI_FlashPowerDownModeSet	This function sets the pin state for flash devices on power down/reset
	PLIB_EBI_FlashResetPinSet	This function sets the the control use of Ready pin and the inverts status
	PLIB_EBI_FlashTimingSet	This function sets the timing for hold in reset for external flash
	PLIB_EBI_MemoryCharacteristicsSet	This function set the Memory Characteristics for Memory or attached devices attached to that pin
	PLIB_EBI_MemoryPagingSet	This function sets the size of the memory page if paging is used.
	PLIB_EBI_MemoryTimingConfigSet	This function is to set the cycle time for page reading
	PLIB_EBI_ReadyPin1ConfigSet	This function sets the the control use of Ready pin and the inverts status.
	PLIB_EBI_ReadyPin2ConfigSet	This function sets the the control use of Ready pin and the inverts status
	PLIB_EBI_ReadyPin3ConfigSet	This function sets the the control use of Ready pin and the inverts status
	PLIB_EBI_ReadyPinSensSet	This function sets the the control use of Ready pin and the inverts status
	PLIB_EBI_StaticMemoryWidthRegisterSet	This function set the data width of static memory.
	PLIB_EBI_WriteOutputControlSet	This function sets the Write Enable & Output Enable control pins
	PLIB_EBI_ControlEnableSet	This function sets the enable bit for turning on or off the EBI Bus.
	PLIB_EBI_ReadyModeSet	This function sets the use of ready mode for each pin. The attached device will either pull the ready pin high or low.

Description

This section describes the Application Programming Interface (API) functions of the EBI Peripheral.

Refer to each section for a detailed description.

5.6.7.7.1 General Functions

5.6.7.7.1.1 PLIB_EBI_ControlEnableClear Function

C

```
void PLIB_EBI_ControlEnableClear(  
    EBI_MODULE_ID index  
);
```

Description

EBIPINEN - in datasheet EBI Pin Enable bit 1 = EBI controls access of pins shared with PMP 0 = Pins shared with EBI are available for general use

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None

Remarks

None.

5.6.7.7.2 Get Functions

5.6.7.7.2.1 PLIB_EBI_AddressHoldTimeGet Function

C

```
int PLIB_EBI_AddressHoldTimeGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

TWR - in the datasheet.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific chip select pin

Returns

Returns a integer, the Address & Data Hold time in number of clocks.

Remarks

None.

5.6.7.7.2.2 PLIB_EBI_AddressSetupTimeGet Function

C

```
int PLIB_EBI_AddressSetupTimeGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

TAS - in the data sheet Clock cycles for address setup time. A value of '0' is only valid in the case of SSRAM.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific chip select pin

Returns

Returns a integer, the address setup time in number of clocks

Remarks

None.

5.6.7.7.2.3 PLIB_EBI_BaseAddressGet Function

C

```
uint32_t PLIB_EBI_BaseAddressGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

CSADDR- in datasheet.

This function returns the base address for each chip select pin

Parameters

Parameters	Description
index	Identifier for the device instance.
ChipSelectNumber	Identifies what chip select address pin address is being read.

Returns

Returns a unsigned integer that is the physical address of the attached device

Remarks

None.

5.6.7.7.2.4 PLIB_EBI_ControlEnableGet Function

C

```
bool PLIB_EBI_ControlEnableGet(  
    EBI_MODULE_ID index  
);
```

Description

EBIPINEN - in datasheet.

EBI Pin Enable bit 1 = EBI controls access of pins shared with PMP 0 = Pins shared with EBI are available for general use

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Returns a bool, if EBI is ON or OFF.

Remarks

None.

5.6.7.7.2.5 PLIB_EBI_DataTurnAroundTimeGet Function**C**

```
int PLIB_EBI_DataTurnAroundTimeGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

TBTA - in the datasheet.

Returns the clock cycles (0-7) for static memory between read-to-write, write-to-read, and read-to-read when Chip Select changes.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific chip select pin

Returns

Returns a integer, the number of clock cycles that is set in the Turn Around Register

Remarks

None.

5.6.7.7.2.6 PLIB_EBI_FlashPowerDownModeGet Function**C**

```
bool PLIB_EBI_FlashPowerDownModeGet(  
    EBI_MODULE_ID index  
);
```

Description

SMRP - in datasheet After a Reset, the controller internally performs a power-down for Flash, and then sets this bit to '1'. 1 = Flash is taken out of Power-down mode 0 = Flash is forced into Power-down mode

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

returns a bool, depending on how FlashPowerDownMode its set

Remarks

None.

5.6.7.7.2.7 PLIB_EBI_FlashResetPinGet Function**C**

```
bool PLIB_EBI_FlashResetPinGet(  
    EBI_MODULE_ID index  
);
```

Description

EBIRPEN - in datasheet.

EBIRP Flash Reset pin, 1 = pin is enabled for use by the EBI module 0 = pin is available for general use.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

returns a bool.

Remarks

None.

5.6.7.7.2.8 PLIB_EBI_FlashTimingGet Function**C**

```
int PLIB_EBI_FlashTimingGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

TRPD -in data sheet The number of clock cycles to hold the external Flash memory in reset.

Preconditions

MemoryType set to Flash

Returns

returns a integer, the number of clock cycles

Remarks

None.

5.6.7.7.2.9 PLIB_EBI_PageModeGet Function

C

```
bool PLIB_EBI_PageModeGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

PAGEMODE - in datasheet.

This Function returns the state of the register PAGEMODE, If the function returns a 1, Device supports Page mode. If the Function returns a 0, Device does not support Page mode .

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific chip select pin

Returns

This function Returns a bool, 1 = Page Mode is used, 0 = Page Mode is not used.

Remarks

None.

5.6.7.7.2.10 PLIB_EBI_PageReadCycleTimeGet Function

C

```
int PLIB_EBI_PageReadCycleTimeGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

TPRC - in datasheet.

Read cycle time = TPRC + 1cycle. The value set and return are the same, the controller performs the read on the next clock, that is why there is a +1.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific chip select pin

Returns

Returns a integer, the number of clock cycles used to read a memory page.

Remarks

None.

5.6.7.7.2.11 PLIB_EBI_PageSizeGet Function

C

```
EBI_PAGE_SIZE PLIB_EBI_PageSizeGet(  

```

```
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

PAGESIZE- in datasheet.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific chip select pin

Returns

This function Returns the size of the memory page.

Remarks

None.

5.6.7.7.2.12 PLIB_EBI_ReadCycleTimeGet Function

C

```
int PLIB_EBI_ReadCycleTimeGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

TRC in datasheet.

Read Cycle time = TRC +1 clock cycle, the controller will always take 1 extra clock (the next clock) for a Read Cycle. The return will be the time set by the user.

setting a the Read Cycle time to a 0 will return a 0 when read, but the controller will add 1 to that value for the next available clock.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies which chip select number the device is attached to.

Returns

returns a integer, the read cycle time in system clocks.

Remarks

None.

5.6.7.7.2.13 PLIB_EBI_ReadyModeGet Function

C

```
bool PLIB_EBI_ReadyModeGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

RDYMODE - in datasheet.

This Function returns the state of the register RDYMODE, If the function returns a 1, Ready input is used. If the Function returns a 0, Ready input is not used.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific chip select pin

Returns

This function Returns a bool, 1 = Ready input is used, 0 = Ready input is not used.

Remarks

None.

5.6.7.7.2.14 PLIB_EBI_StaticMemoryWidthRegisterGet Function**C**

```
EBI_STATIC_MEMORY_WIDTH PLIB_EBI_StaticMemoryWidthRegisterGet(  
    EBI_MODULE_ID index,  
    int RegstierNumber  
);
```

Description

SMDWIDTH - in data sheet 100 = 16 bits 000 = 8 bits

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

returns a enum type, which bus mode is used

Remarks

None.

5.6.7.7.2.15 PLIB_EBI_WritePulseWidthGet Function**C**

```
int PLIB_EBI_WritePulseWidthGet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber  
);
```

Description

TWP in the datasheet.

Write Pulse with = TWP + 1 clock cycle

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific chip select pin

Returns

Returns a integer, the number of clock cycles the write pulse width is set to.

Remarks

None.

5.6.7.7.3 Set Functions

5.6.7.7.3.1 PLIB_EBI_AddressPinEnableBitsSet Function

C

```
void PLIB_EBI_AddressPinEnableBitsSet(  
    EBI_MODULE_ID index,  
    EBI_ADDRESS_PORT AddressPort  
);
```

Description

These pins have to be enabled for the size of the data bus in a contiguous order.

EBIA23EN:EBIA0EN - in datasheet.

EBI Address Pin Enable bits,

1 = pin is enabled for use by EBI, 0 = pin has is available for general use.

Parameters

Parameters	Description
index	Identifier for the device instance.
AddressPort	Identifies how many pins are used for addressing on the EBI bus.

Returns

None.

Remarks

None.

5.6.7.7.3.2 PLIB_EBI_BaseAddressSet Function

C

```
void PLIB_EBI_BaseAddressSet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber,  
    uint32_t BaseAddress  
);
```

Description

CSADDR- in datasheet.

This function sets the base address for physical memory at each chip select pin.

Preconditions

EBI has to be supported by the microcontroller and has to have a device attached to it.

Parameters

Parameters	Description
index	Identifier for the device instance.
ChipSelectNumber	Identifies what chip select address pin is being assigned an address
BaseAddress	a physical address for memory

Returns

None.

Remarks

None.

5.6.7.7.3 PLIB_EBI_ByteSelectPinSet Function**C**

```
void PLIB_EBI_ByteSelectPinSet(  
    EBI_MODULE_ID index,  
    bool ByteSelect0,  
    bool ByteSelect1  
);
```

Description

If the system uses Byte Select High/Low pins for the data, the pins must be enabled for use in the EBI controller.

EBIBSEN0 - in datasheet.

EBIBSEN0, This Controls the pin Enable bit for the Lower Data Byte Select. Lower Data Byte is bits 8:0.

1 = pin is enabled for use by the EBI module, 0 = pin is available for general use.

EBIBSEN1 - in datasheet.

EBIBSEN1, This Controls the pin Enable bit for the Upper Data Byte Select. Upper Data Byte is bits 15:8.

1 = pin is enabled for use by the EBI module, 0 = pin is available for general use.

Preconditions

Registers EBIDEN0 & EBIDEN1 must be enabled first, this can be done in the function [PLIB_EBI_DataEnableSet\(\)](#).

Parameters

Parameters	Description
index	Identifier for the device instance
ByteSelect0	Identifies Lower Byte Select Pin for enabling.
ByteSelect1	Identifies Upper Byte Select Pin for enabling.

Returns

None.

Remarks

None.

5.6.7.7.3.4 PLIB_EBI_ChipSelectEnableSet Function

C

```
void PLIB_EBI_ChipSelectEnableSet(  
    EBI_MODULE_ID index,  
    bool ChipSelect0,  
    bool ChipSelect1,  
    bool ChipSelect2,  
    bool ChipSelect3  
);
```

Description

EBICSEN0 = EBI ChipSelect0 Pin Enable bit. 1 = pin is enabled for use by the EBI module, 0 = pin is available for general use.

EBICSEN1 = EBI ChipSelect1 Pin Enable bit. 1 = pin is enabled for use by the EBI module, 0 = pin is available for general use.

EBICSEN2 = EBI ChipSelect2 Pin Enable bit. 1 = pin is enabled for use by the EBI module, 0 = pin is available for general use.

EBICSEN3 = EBI ChipSelect3 Pin Enable bit. 1 = pin is enabled for use by the EBI module, 0 = pin is available for general use.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelect0	Identifies control of Chip Select 0 for enabling.
ChipSelect1	Identifies control of Chip Select 1 for enabling.
ChipSelect2	Identifies control of Chip Select 2 for enabling.
ChipSelect3	Identifies control of Chip Select 3 for enabling.

Returns

None.

Remarks

None.

5.6.7.7.3.5 PLIB_EBI_DataEnableSet Function

C

```
void PLIB_EBI_DataEnableSet(  
    EBI_MODULE_ID index,  
    bool DataUpperByte,  
    bool DataLowerByte  
);
```

Description

EBIDEN1 - in datasheet.

EBI Data Upper Byte Pin Enable bit.

1 = [15:8] pins are enabled for use by the EBI module, 0 = [15:8] pins have reverted to general use.

EBIDEN0 - in datasheet.

EBI Data Upper Byte Pin Enable bit.

1 = [7:0] pins are enabled for use by the EBI module, 0 = [7:0] pins have reverted to general use.

Parameters

Parameters	Description
index	Identifier for the device instance
DataUpperByte	Identifies control of Upper Data Byte for enabling.
DataLowerByte	Identifies control of Lower Data Byte for enabling.

Returns

None.

Remarks

None.

5.6.7.7.3.6 PLIB_EBI_FlashPowerDownModeSet Function**C**

```
void PLIB_EBI_FlashPowerDownModeSet(  
    EBI_MODULE_ID index,  
    bool FlashPowerDownMode  
);
```

Description

SMRP- in datasheet.

After a Reset, the controller internally performs a power-down for Flash, and then sets this bit to '1'.

1 = Flash is taken out of Power-down mode.

0 = Flash is forced into Power-down mode.

Parameters

Parameters	Description
index	Identifier for the device instance
FlashPowerDownMode	a bool, sets the power down state for flash memory

Returns

None.

Remarks

None.

5.6.7.7.3.7 PLIB_EBI_FlashResetPinSet Function**C**

```
void PLIB_EBI_FlashResetPinSet(  
    EBI_MODULE_ID index,  
    bool FlashResetPin  
);
```

Description

EBIRPEN - in datasheet.

EBIRP Flash Reset pin, 1 = pin is enabled for use by the EBI module 0 = pin is available for general use.

Parameters

Parameters	Description
index	Identifier for the device instance
FlashReadPin	Flash Rest Pin is Used

Returns

None.

Remarks

None.

5.6.7.7.3.8 PLIB_EBI_FlashTimingSet Function**C**

```
void PLIB_EBI_FlashTimingSet(  
    EBI_MODULE_ID index,  
    int FlashTiming  
);
```

Description

TRPD - in datasheet.

The number of clock cycles to hold the external Flash memory in reset.

Preconditions

NOR Flash has to be used with EBI instead of SRAM

Parameters

Parameters	Description
index	Identifier for the device instance
FlashTiming	an int, the number of clock cycles

Returns

None.

Remarks

None.

5.6.7.7.3.9 PLIB_EBI_MemoryCharacteristicsSet Function**C**

```
void PLIB_EBI_MemoryCharacteristicsSet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber,  
    EBI_MEMORY_TYPE MemoryType,  
    EBI_MEMORY_SIZE MemorySize,  
    EBI_CS_TIMING TimingReg  
);
```

Description

EBIMSKx -SRF name index - Identifier for the device instance.

ChipSelectNumber - Identifies what chip select address pin is being assigned an address

MEMTYPE - in datasheet.

MEMSIZE - in the datasheet.

REGSEL - in the datasheet.

Preconditions

EBI has to be supported by the microcontroller.

Parameters

Parameters	Description
index	Identifier for the device instance.
ChipSelectNumber	Identifies which chip select pin
MemoryType	Identifies which memory is used
MemorySize	a enum type, sets the size of the attached memory device
TimingRegSet	Identifies the timing register

Returns

None.

Remarks

None.

5.6.7.7.3.10 PLIB_EBI_MemoryPagingSet Function

C

```
void PLIB_EBI_MemoryPagingSet(  
    EBI_MODULE_ID index,  
    int ChipSelectNumber,  
    bool PageMode,  
    EBI_PAGE_SIZE MemoryPageSize  
);
```

Description

PAGESIZE in datasheet.

PAGEMODE in datasheet.

1 = Device supports Page mode

0 = Device does not support Page mode

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Sets for that Chip Select pin [0:3]
PageMode	enable or disable page mode
MemoryPageSize	size of memory pages

Returns

none

Remarks

None.

5.6.7.7.3.11 PLIB_EBI_MemoryTimingConfigSet Function

C

```
void PLIB_EBI_MemoryTimingConfigSet(
    EBI_MODULE_ID index,
    int CS_Timing_Reg,
    int PageReadTime,
    int DataTurnAroundTime,
    int WritePulseWidth,
    int AddressHoldTime,
    int AddressSetupTime,
    int ReadCycleTime
);
```

Description

SFR = EBISMT[0:2] TPRC - in datasheet. Read cycle time = TPRC + 1cycle.

TBTA - in the datasheet. Clock cycles (0-7) for static memory between read-to-write, write-to-read, and read-to-read when Chip Select changes.

TWP in the datasheet. Write Pulse with = TWP + 1 clock cycle.

TWR - in datasheet. Write Address/Data Hold Time

TAS - in datasheet. Clock cycles for address setup time. A value of '0' is only valid in the case of SSRAM.

TRC - in datasheet. Read Cycle time = TRC +1 clock cycle.

Preconditions

EBI has to be supported by the microcontroller.

Parameters

Parameters	Description
index	Identifier for the device instance
CS_Timing_Reg	Identifies which chip select Timing register to use EBISMT[0:2]
PageReadTime	The clock cycle needed for a Memory Page read
DataTurnAroundTime	Time between read-to-write, write-to-read, and read-to-read when Chip Select changes state.
WritePulseWidth	The clock cycles needed for a memory write
AddressHoldTime	The clock time needed for the address bus to hold
AddressSetupTime	The time needed for the address to settle
ReadCycleTime	_nt_

Returns

None.

Remarks

None.

5.6.7.7.3.12 PLIB_EBI_ReadyPin1ConfigSet Function

C

```
void PLIB_EBI_ReadyPin1ConfigSet(
    EBI_MODULE_ID index,
    bool ReadyPin1Enable,
    bool ReadyPin1Invert
```

```
);
```

Description

EBIRDYEN1 - in datasheet.

EBIRDY1 Pin Enable bit, 1 = pin is enabled for use by the EBI module, 0 = pin is available for general use

EBIRDYINV1 - in datasheet.

EBIRDYx Inversion Control bit 1 = Invert EBIRDYx pin before use 0 = Do not invert EBIRDYx pin before use.

Parameters

Parameters	Description
index	Identifier for the device instance
ReadyPin1Invert	Identifies the on/off for invert
ReadyPin1Enable	Identifies if this pin is used by the control part

Returns

None.

Remarks

None.

5.6.7.7.3.13 PLIB_EBI_ReadyPin2ConfigSet Function**C**

```
void PLIB_EBI_ReadyPin2ConfigSet(  
    EBI_MODULE_ID index,  
    bool ReadyPin2Enable,  
    bool ReadyPin2Invert  
);
```

Description

EBIRDYEN2 - in datasheet.

EBIRDY2 Pin Enable bit, 1 = pin is enabled for use by the EBI module, 0 = pin is available for general use

EBIRDYINV2 - in datasheet.

EBIRDYx Inversion Control bit 1 = Invert EBIRDYx pin before use 0 = Do not invert EBIRDYx pin before use.

Parameters

Parameters	Description
index	Identifier for the device instance
ReadyPin2Invert	Identifies the on/off for invert
ReadyPin2Enable	Identifies if this pin is used by the control part

Returns

None.

Remarks

None.

5.6.7.7.3.14 PLIB_EBI_ReadyPin3ConfigSet Function

C

```
void PLIB_EBI_ReadyPin3ConfigSet(  
    EBI_MODULE_ID index,  
    bool ReadyPin3Enable,  
    bool ReadyPin3Invert  
);
```

Description

EBIRDYEN2 - in datasheet.

EBIRDY2 Pin Enable bit, 1 = pin is enabled for use by the EBI module, 0 = pin is available for general use

EBIRDYINV2 - in datasheet.

EBIRDYx Inversion Control bit 1 = Invert EBIRDYx pin before use 0 = Do not invert EBIRDYx pin before use.

Parameters

Parameters	Description
index	Identifier for the device instance
ReadyPin3Invert	Identifies the on/off for invert
ReadyPin3Enable	Identifies if this pin is used by the control part

Returns

None.

Remarks

None.

5.6.7.7.3.15 PLIB_EBI_ReadyPinSensSet Function

C

```
void PLIB_EBI_ReadyPinSensSet(  
    EBI_MODULE_ID index,  
    bool SensitivityControl  
);
```

Description

EBIRDYLVL - in datasheet.

EBI Ready Pin trigger detection 1 = Use level detect for EBIRDYx pins, 0 = Use edge detect for EBIRDYx pins

Parameters

Parameters	Description
index	Identifier for the device instance
SensitivityControl	for selection edge or level detection

Returns

None.

Remarks

None.

5.6.7.7.3.16 PLIB_EBI_StaticMemoryWidthRegisterSet Function

C

```
void PLIB_EBI_StaticMemoryWidthRegisterSet(  
    EBI_MODULE_ID index,  
    int RegstierNumber,  
    EBI_STATIC_MEMORY_WIDTH StaticMemoryWidthRegister  
);
```

Description

SMDWIDTH - in the datasheet.

000 = 16 bits

100 = 8 bits

Parameters

Parameters	Description
index	Identifier for the device instance
RegstierNumber	the ID for the register being set StaticMemoryWidthRegister -identifies weither a bus width of 8bit or 16bits

Returns

None.

Remarks

None.

5.6.7.7.3.17 PLIB_EBI_WriteOutputControlSet Function

C

```
void PLIB_EBI_WriteOutputControlSet(  
    EBI_MODULE_ID index,  
    bool WriteEnable,  
    bool OutputEnable  
);
```

Description

EBIWEEN - in datasheet.

EBIWE, This controls the Enable bit for the Write Enable Pin.

1 = pin is enabled for use by the EBI module, 0 = pin is available for general use.

EBIOEEN - in datasheet.

EBIOE, This controls the enable bit for the Output Enable Pin.

1 = pin is enabled for use by the EBI module, 0 = pin is available for general use.

Parameters

Parameters	Description
index	Identifier for the device instance
WriteEnable	for enabling Write Enable
OutputEnable	for enabling Output Enable

Returns

None.

Remarks

None.

5.6.7.7.3.18 PLIB_EBI_ControlEnableSet Function**C**

```
void PLIB_EBI_ControlEnableSet(  
    EBI_MODULE_ID index,  
    bool EnableBit  
);
```

Description

EBIPINEN - in datasheet.

EBI Pin Enable bit, 1 = EBI controls access of pins shared with PMP 0 = Pins shared with EBI are available for general use.

Parameters

Parameters	Description
index	Identifier for the device instance
EnableBit	Identifies the enable bit

Returns

None

Remarks

None.

5.6.7.7.3.19 PLIB_EBI_ReadyModeSet Function**C**

```
void PLIB_EBI_ReadyModeSet(  
    EBI_MODULE_ID index,  
    bool ReadyPin0,  
    bool ReadyPin1,  
    bool ReadyPin2  
);
```

Description

RDYMODE- The device associated with EBI is a data-ready device, and will use the READY pin.

1 = Ready input is used 0 = Ready input is not used

Parameters

Parameters	Description
index	Identifier for the device instance ReadyPin0 ReadyPin1 ReadyPin2
ReadyPin(x)	Identifies the ready pin (1-3) that

Returns

None.

Remarks

None.

5.6.7.7.4 Existence Functions

5.6.7.7.4.1 PLIB_EBI_ExistsAddressHoldTime Function

C

```
bool PLIB_EBI_ExistsAddressHoldTime(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the AddressHoldTime feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_AddressHoldTimeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AddressHoldTime feature is supported on the device
- false - The AddressHoldTime feature is not supported on the device

Remarks

None.

5.6.7.7.4.2 PLIB_EBI_ExistsAddressPinEnableBits Function

C

```
bool PLIB_EBI_ExistsAddressPinEnableBits(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the AddressPinEnableBits feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_AddressPinEnableBitsSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AddressPinEnableBits feature is supported on the device
- false - The AddressPinEnableBits feature is not supported on the device

Remarks

None.

5.6.7.7.4.3 PLIB_EBI_ExistsAddressSetupTime Function**C**

```
bool PLIB_EBI_ExistsAddressSetupTime(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the AddressSetupTime feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_AddressSetupTimeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AddressSetupTime feature is supported on the device
- false - The AddressSetupTime feature is not supported on the device

Remarks

None.

5.6.7.7.4.4 PLIB_EBI_ExistsBaseAddress Function**C**

```
bool PLIB_EBI_ExistsBaseAddress(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the Base_Address feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_BaseAddressSet](#)
- [PLIB_EBI_BaseAddressGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Base_Address feature is supported on the device
- false - The Base_Address feature is not supported on the device

Remarks

None.

5.6.7.7.4.5 PLIB_EBI_ExistsByteSelectPin Function**C**

```
bool PLIB_EBI_ExistsByteSelectPin(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the ByteSelectPin feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_ByteSelectPinSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ByteSelectPin feature is supported on the device
- false - The ByteSelectPin feature is not supported on the device

Remarks

None.

5.6.7.7.4.6 PLIB_EBI_ExistsChipSelectEnable Function**C**

```
bool PLIB_EBI_ExistsChipSelectEnable(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the ChipSelectEnable feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_ChipSelectEnableSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipSelectEnable feature is supported on the device
- false - The ChipSelectEnable feature is not supported on the device

Remarks

None.

5.6.7.7.4.7 PLIB_EBI_ExistsControlEnable Function**C**

```
bool PLIB_EBI_ExistsControlEnable(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the ControlEnable feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_ControlEnableSet](#)
- [PLIB_EBI_ControlEnableGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ControlEnable feature is supported on the device
- false - The ControlEnable feature is not supported on the device

Remarks

None.

5.6.7.7.4.8 PLIB_EBI_ExistsDataEnable Function**C**

```
bool PLIB_EBI_ExistsDataEnable(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the DataEnable feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_DataEnableSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataEnable feature is supported on the device
- false - The DataEnable feature is not supported on the device

Remarks

None.

5.6.7.7.4.9 PLIB_EBI_ExistsDataTurnAroundTime Function**C**

```
bool PLIB_EBI_ExistsDataTurnAroundTime(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the DataTurnAroundTime feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_DataTurnAroundTimeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataTurnAroundTime feature is supported on the device
- false - The DataTurnAroundTime feature is not supported on the device

Remarks

None.

5.6.7.7.4.10 PLIB_EBI_ExistsFlashPowerDownMode Function**C**

```
bool PLIB_EBI_ExistsFlashPowerDownMode(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the FlashPowerDownMode feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_FlashPowerDownModeSet](#)

- [PLIB_EBI_FlashPowerDownModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FlashPowerDownMode feature is supported on the device
- false - The FlashPowerDownMode feature is not supported on the device

Remarks

None.

5.6.7.7.4.11 PLIB_EBI_ExistsFlashResetPin Function**C**

```
bool PLIB_EBI_ExistsFlashResetPin(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the FlashResetPin feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_FlashResetPinSet](#)
- [PLIB_EBI_FlashResetPinGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FlashResetPin feature is supported on the device
- false - The FlashResetPin feature is not supported on the device

Remarks

None.

5.6.7.7.4.12 PLIB_EBI_ExistsFlashTiming Function**C**

```
bool PLIB_EBI_ExistsFlashTiming(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the FlashTiming feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_FlashTimingSet](#)
- [PLIB_EBI_FlashTimingGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FlashTiming feature is supported on the device
- false - The FlashTiming feature is not supported on the device

Remarks

None.

5.6.7.7.4.13 PLIB_EBI_ExistsMemoryCharacteristics Function

C

```
bool PLIB_EBI_ExistsMemoryCharacteristics(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the MemoryCharacteristics feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_MemoryCharacteristicsSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MemoryCharacteristics feature is supported on the device
- false - The MemoryCharacteristics feature is not supported on the device

Remarks

None.

5.6.7.7.4.14 PLIB_EBI_ExistsMemoryPaging Function

C

```
bool PLIB_EBI_ExistsMemoryPaging(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the MemoryPaging feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_MemoryPagingSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MemoryPaging feature is supported on the device
- false - The MemoryPaging feature is not supported on the device

Remarks

None.

5.6.7.7.4.15 PLIB_EBI_ExistsMemoryTimingConfig Function**C**

```
bool PLIB_EBI_ExistsMemoryTimingConfig(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the MemoryTimingConfig feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_MemoryTimingConfigSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MemoryTimingConfig feature is supported on the device
- false - The MemoryTimingConfig feature is not supported on the device

Remarks

None.

5.6.7.7.4.16 PLIB_EBI_ExistsPageMode Function**C**

```
bool PLIB_EBI_ExistsPageMode(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the PageMode feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_PageModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PageMode feature is supported on the device
- false - The PageMode feature is not supported on the device

Remarks

None.

5.6.7.7.4.17 PLIB_EBI_ExistsPageReadTime Function**C**

```
bool PLIB_EBI_ExistsPageReadTime(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the PageReadTime feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_PageReadCycleTimeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PageReadTime feature is supported on the device
- false - The PageReadTime feature is not supported on the device

Remarks

None.

5.6.7.7.4.18 PLIB_EBI_ExistsReadCycleTime Function**C**

```
bool PLIB_EBI_ExistsReadCycleTime(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the ReadCycleTime feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_ReadCycleTimeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReadCycleTime feature is supported on the device
- false - The ReadCycleTime feature is not supported on the device

Remarks

None.

5.6.7.7.4.19 PLIB_EBI_ExistsReadyMode Function**C**

```
bool PLIB_EBI_ExistsReadyMode(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the ReadyMode feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_ReadyModeSet](#)
- [PLIB_EBI_ReadyModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReadyMode feature is supported on the device
- false - The ReadyMode feature is not supported on the device

Remarks

None.

5.6.7.7.4.20 PLIB_EBI_ExistsReadyPin1Config Function**C**

```
bool PLIB_EBI_ExistsReadyPin1Config(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the ReadyPin1Config feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_ReadyPin1ConfigSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReadyPin1Config feature is supported on the device
- false - The ReadyPin1Config feature is not supported on the device

Remarks

None.

5.6.7.7.4.21 PLIB_EBI_ExistsReadyPin2Config Function

C

```
bool PLIB_EBI_ExistsReadyPin2Config(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the ReadyPin2Config feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_ReadyPin2ConfigSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReadyPin2Config feature is supported on the device
- false - The ReadyPin2Config feature is not supported on the device

Remarks

None.

5.6.7.7.4.22 PLIB_EBI_ExistsReadyPin3Config Function

C

```
bool PLIB_EBI_ExistsReadyPin3Config(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the ReadyPin3Config feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_ReadyPin3ConfigSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReadyPin3Config feature is supported on the device
- false - The ReadyPin3Config feature is not supported on the device

Remarks

None.

5.6.7.7.4.23 PLIB_EBI_ExistsReadyPinSens Function**C**

```
bool PLIB_EBI_ExistsReadyPinSens(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the ReadyPinSens feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_ReadyPinSensSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReadyPinSens feature is supported on the device
- false - The ReadyPinSens feature is not supported on the device

Remarks

None.

5.6.7.7.4.24 PLIB_EBI_ExistsStaticMemoryWidthRegister Function**C**

```
bool PLIB_EBI_ExistsStaticMemoryWidthRegister(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the StaticMemoryWidthRegister feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_StaticMemoryWidthRegisterSet](#)

- [PLIB_EBI_StaticMemoryWidthRegisterGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StaticMemoryWidthRegister feature is supported on the device
- false - The StaticMemoryWidthRegister feature is not supported on the device

Remarks

None.

5.6.7.7.4.25 PLIB_EBI_ExistsWriteOutputControl Function

C

```
bool PLIB_EBI_ExistsWriteOutputControl(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the WriteOutputControl feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_WriteOutputControlSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WriteOutputControl feature is supported on the device
- false - The WriteOutputControl feature is not supported on the device

Remarks

None.

5.6.7.7.4.26 PLIB_EBI_ExistsWritePulseWidth Function

C

```
bool PLIB_EBI_ExistsWritePulseWidth(  
    EBI_MODULE_ID index  
);
```

Description

This function identifies whether the WritePulseWidth feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- [PLIB_EBI_WritePulseWidthGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WritePulseWidth feature is supported on the device
- false - The WritePulseWidth feature is not supported on the device

Remarks

None.

5.6.7.8 Files

Files

Name	Description
plib_ebi.h	External Bus Interface (EBI) Peripheral Library Interface Header.

Description

5.6.7.8.1 plib_ebi.h

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the EBI Peripheral Library.

Functions

	Name	Description
≡	PLIB_EBI_AddressHoldTimeGet	This function returns the Address Hold Time
≡	PLIB_EBI_AddressPinEnableBitsSet	This function sets the address pins used for EBI.
≡	PLIB_EBI_AddressSetupTimeGet	This function is to get the set hold time
≡	PLIB_EBI_BaseAddressGet	This function returns the Base Address set for each chip select
≡	PLIB_EBI_BaseAddressSet	This function sets the Base address for physical memory at each chip select
≡	PLIB_EBI_ByteSelectPinSet	This function sets the data Byte Select High [15:8] and Low [7:0] enable pins for use.
≡	PLIB_EBI_ChipSelectEnableSet	This function sets the Chip Select pins for use with the EBI or GPIO.
≡	PLIB_EBI_ControlEnableClear	This function gets the set power down state.
≡	PLIB_EBI_ControlEnableGet	This function gets the status of the enable bit for EBI.
≡	PLIB_EBI_ControlEnableSet	This function sets the enable bit for turning on or off the EBI Bus.
≡	PLIB_EBI_DataEnableSet	This function sets the use of Data Byte Select Pins, High and Low, for control with EBI or GPIO use.
≡	PLIB_EBI_DataTurnAroundTimeGet	This function gets the data turn time set for the EBI Bus.
≡	PLIB_EBI_ExistsAddressHoldTime	Identifies whether the AddressHoldTime feature exists on the EBI module

	PLIB_EBI_ExistsAddressPinEnableBits	Identifies whether the AddressPinEnableBits feature exists on the EBI module
	PLIB_EBI_ExistsAddressSetupTime	Identifies whether the AddressSetupTime feature exists on the EBI module
	PLIB_EBI_ExistsBaseAddress	Identifies whether the Base_Address feature exists on the EBI module
	PLIB_EBI_ExistsByteSelectPin	Identifies whether the ByteSelectPin feature exists on the EBI module
	PLIB_EBI_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the EBI module
	PLIB_EBI_ExistsControlEnable	Identifies whether the ControlEnable feature exists on the EBI module
	PLIB_EBI_ExistsDataEnable	Identifies whether the DataEnable feature exists on the EBI module
	PLIB_EBI_ExistsDataTurnAroundTime	Identifies whether the DataTurnAroundTime feature exists on the EBI module
	PLIB_EBI_ExistsFlashPowerDownMode	Identifies whether the FlashPowerDownMode feature exists on the EBI module
	PLIB_EBI_ExistsFlashResetPin	Identifies whether the FlashResetPin feature exists on the EBI module
	PLIB_EBI_ExistsFlashTiming	Identifies whether the FlashTiming feature exists on the EBI module
	PLIB_EBI_ExistsMemoryCharacteristics	Identifies whether the MemoryCharacteristics feature exists on the EBI module
	PLIB_EBI_ExistsMemoryPaging	Identifies whether the MemoryPaging feature exists on the EBI module
	PLIB_EBI_ExistsMemoryTimingConfig	Identifies whether the MemoryTimingConfig feature exists on the EBI module
	PLIB_EBI_ExistsPageMode	Identifies whether the PageMode feature exists on the EBI module
	PLIB_EBI_ExistsPageReadTime	Identifies whether the PageReadTime feature exists on the EBI module
	PLIB_EBI_ExistsReadCycleTime	Identifies whether the ReadCycleTime feature exists on the EBI module
	PLIB_EBI_ExistsReadyMode	Identifies whether the ReadyMode feature exists on the EBI module
	PLIB_EBI_ExistsReadyPin1Config	Identifies whether the ReadyPin1Config feature exists on the EBI module
	PLIB_EBI_ExistsReadyPin2Config	Identifies whether the ReadyPin2Config feature exists on the EBI module
	PLIB_EBI_ExistsReadyPin3Config	Identifies whether the ReadyPin3Config feature exists on the EBI module
	PLIB_EBI_ExistsReadyPinSens	Identifies whether the ReadyPinSens feature exists on the EBI module
	PLIB_EBI_ExistsStaticMemoryWidthRegister	Identifies whether the StaticMemoryWidthRegister feature exists on the EBI module
	PLIB_EBI_ExistsWriteOutputControl	Identifies whether the WriteOutputControl feature exists on the EBI module
	PLIB_EBI_ExistsWritePulseWidth	Identifies whether the WritePulseWidth feature exists on the EBI module
	PLIB_EBI_FlashPowerDownModeGet	This function gets the set power down state.
	PLIB_EBI_FlashPowerDownModeSet	This function sets the pin state for flash devices on power down/reset
	PLIB_EBI_FlashResetPinGet	This function sets the the control use of Ready pin and the inverts status
	PLIB_EBI_FlashResetPinSet	This function sets the the control use of Ready pin and the inverts status

	PLIB_EBI_FlashTimingGet	This function gets the set flash timing for external flash
	PLIB_EBI_FlashTimingSet	This function sets the timing for hold in reset for external flash
	PLIB_EBI_MemoryCharacteristicsSet	This function set the Memory Characteristics for Memory or attached devices attached to that pin
	PLIB_EBI_MemoryPagingSet	This function sets the size of the memory page if paging is used.
	PLIB_EBI_MemoryTimingConfigSet	This function is to set the cycle time for page reading
	PLIB_EBI_PageModeGet	This function returns Paging mode settings
	PLIB_EBI_PageReadCycleTimeGet	This function returns the cycle time for a page read.
	PLIB_EBI_PageSizeGet	This function returns Paging mode settings
	PLIB_EBI_ReadCycleTimeGet	This function returns the read time in number of clock cycles
	PLIB_EBI_ReadyModeGet	This function returns whether or not the Ready Mode was set.
	PLIB_EBI_ReadyModeSet	This function sets the use of ready mode for each pin. The attached device will either pull the ready pin high or low.
	PLIB_EBI_ReadyPin1ConfigSet	This function sets the the control use of Ready pin and the inverts status.
	PLIB_EBI_ReadyPin2ConfigSet	This function sets the the control use of Ready pin and the inverts status
	PLIB_EBI_ReadyPin3ConfigSet	This function sets the the control use of Ready pin and the inverts status
	PLIB_EBI_ReadyPinSensSet	This function sets the the control use of Ready pin and the inverts status
	PLIB_EBI_StaticMemoryWidthRegisterGet	This function get the width of the set data bus
	PLIB_EBI_StaticMemoryWidthRegisterSet	This function set the data width of static memory.
	PLIB_EBI_WriteOutputControlSet	This function sets the Write Enable & Output Enable control pins
	PLIB_EBI_WritePulseWidthGet	This function returns the set hold time in clock cycles.

File Name

plib_ebi.h

Company

Microchip Technology Inc.

5.6.8 Ethernet Peripheral Library

5.6.8.1 Introduction

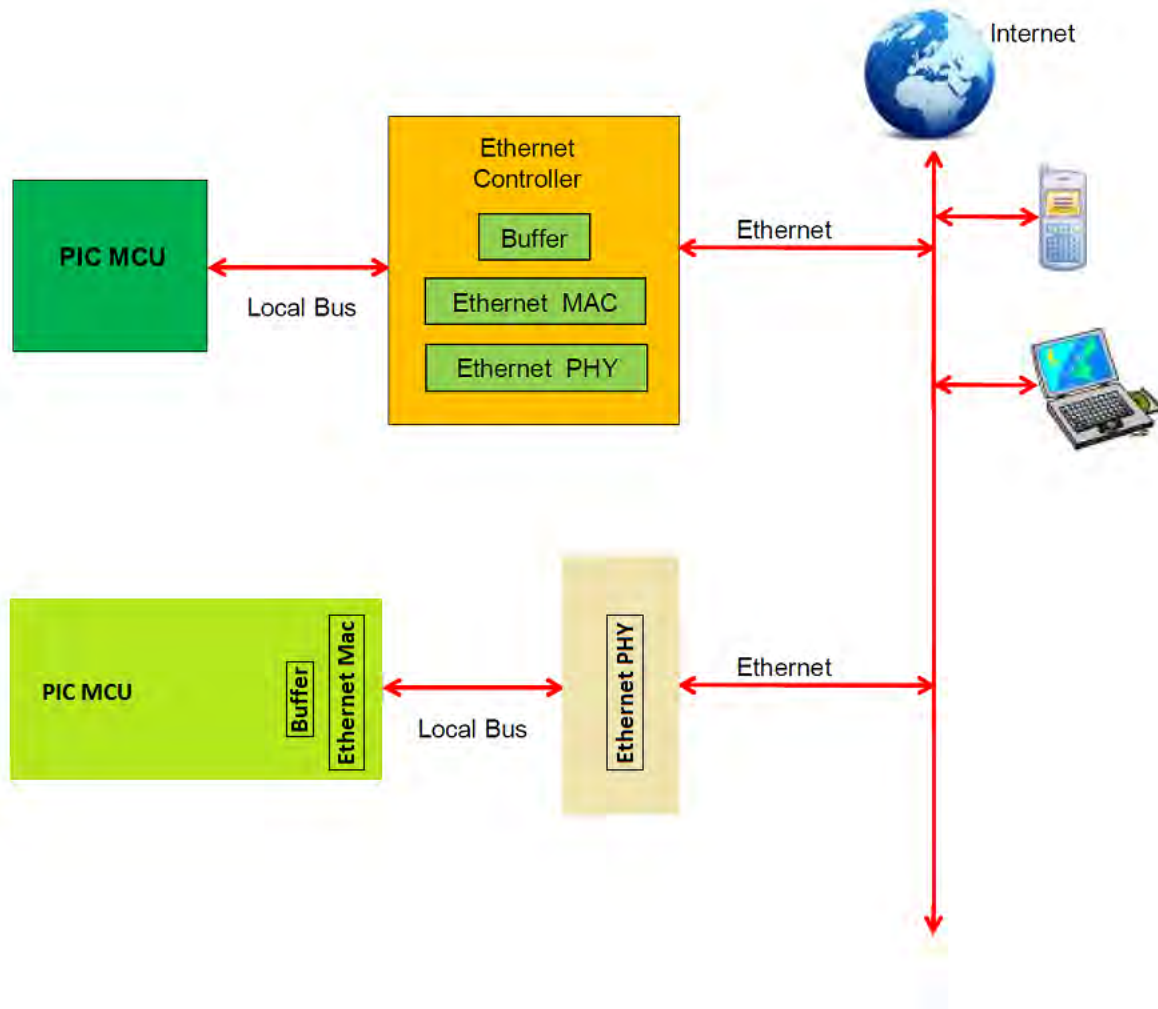
Ethernet Peripheral Library**for****Microchip Microcontrollers**

The Ethernet Peripheral Library provides a low-level abstraction of the Ethernet module in the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without interacting directly with the module's registers, thereby hiding the differences in microcontroller variations.

Description

Overview

Ethernet is the most widely deployed network in offices and homes. Ethernet's infrastructure, interoperability, and scalability serve to ease development and facilitate communications to the device. Once equipment is connected to an Ethernet network, it can be monitored or controlled through the internet with low latency – “real time” remote delivery.



The Ethernet Controller is a bus master module that interfaces with an off-chip PHY in order to implement a complete Ethernet node in a system.

Following are some of the key features of this module:

- Supports 10/100 Mbps data transfer rates
- Supports Full-Duplex and Half-Duplex operation
- Supports Reduced Media Independent Interface (RMII) and Media Independent Interface (MII) PHY interface
- Supports MII Management (MIIM) PHY Management interface

- Supports both manual and automatic flow control
- Supports Auto-MDIX and enabled PHYs
- RAM descriptor based DMA operation for both receive and transmit path
- Fully configurable interrupts
- Configurable receive packet filtering:
 - CRC Check
 - 64-byte Pattern Match
 - Broadcast, Multicast and Unicast packets
 - Magic Packet™
 - 64-bit Hash Table
 - Runt Packet
- Supports Packet Payload Checksum calculation
- Supports various hardware statistics counters

5.6.8.2 Release Notes

MPLAB Harmony Version: v0.70b **Ethernet Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.8.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.8.4 Using the Library

This topic describes the basic architecture of the Ethernet Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_eth.h](#)

The interface to the Ethernet Peripheral Library is defined in the "[plib_eth.h](#)" header file. Any C language source (.c) file that uses the Ethernet Peripheral Library should include "[plib_eth.h](#)".

Library File:

The Ethernet Peripheral Library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the Peripheral interacts with the framework.

5.6.8.4.1 Hardware Abstraction Model

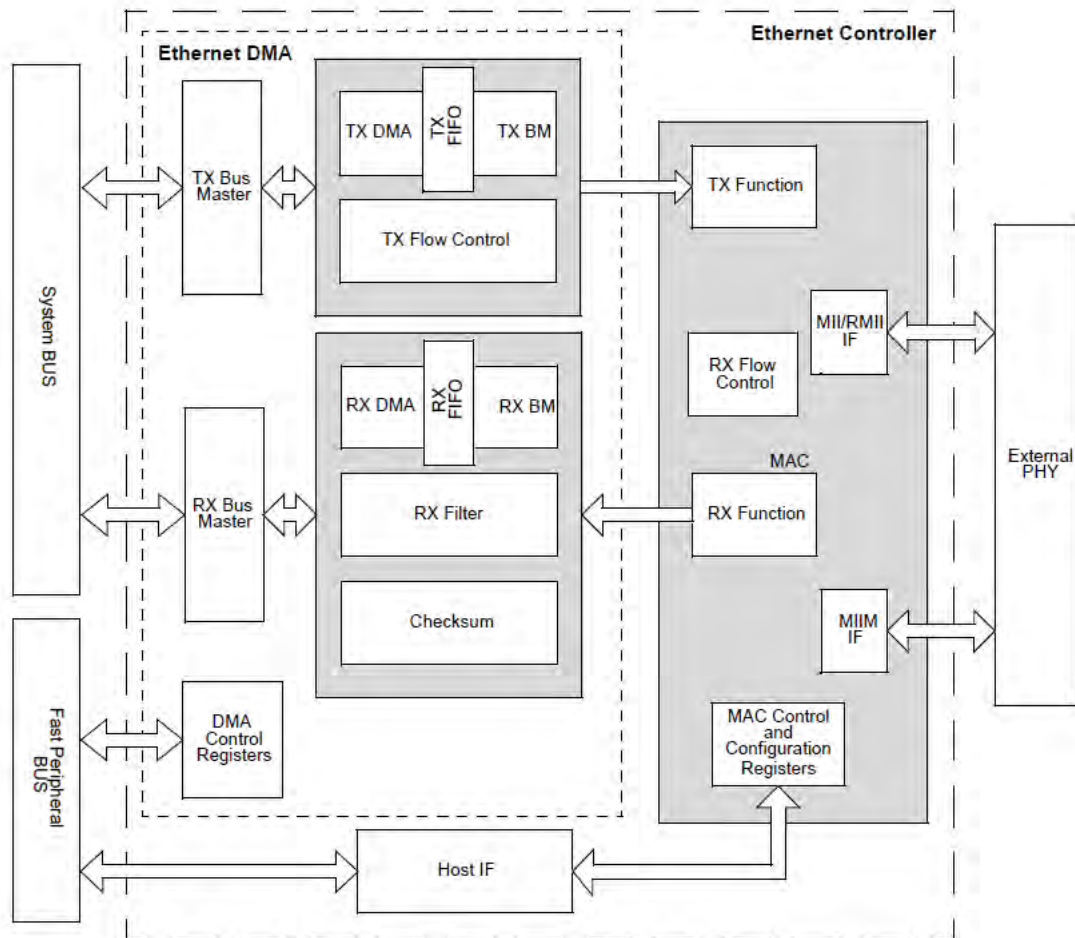
This section describes how the low-level abstraction is modeled in the software and introduces the library interface. The interface provided is a superset of all the functionality of the available Ethernet on the device. Refer to the specific data sheet or family reference manual to determine the set of functions that are supported for each Ethernet module on your device.

Description

Hardware Model

This library provides the low-level abstraction of the Ethernet module on Microchip family of microcontrollers with a convenient C language interface.

Ethernet Controller Block Diagram



Note: For information on off-chip options to provide Ethernet functionality, refer to the specific device data sheet.

5.6.8.4.2 How the Library Works

This section provides information on the operation of the Ethernet Peripheral Library.

Description

The Peripheral Library is an abstraction of hardware which can be used to control hardware and obtain status. It is intended to allow the programmer access to hardware registers without knowing the exact bit or register names which may vary on individual devices. For example, instead of turning on the Ethernet module using:

ETHCON1bits.EON = 1; the programmer can turn on the Ethernet module using: `PLIB_ETH_Enable(moduleId)`. This allows the programmer to use the same command for any device containing an Ethernet module without having to learn the individual device register names.

Basic Overview

The Ethernet Controller provides the modules needed to implement a 10/100 Mbps Ethernet node using an external PHY chip. To off-load the CPU from moving packet data to and from the module, internal descriptor-based DMA engines are included in the controller.

The Ethernet Controller consists of the following modules:

- Media Access Control (MAC) block: Responsible for implementing the MAC functions of the Ethernet IEEE 802.3 Specification
- Flow Control (FC) block, which provides:
 - Manual flow control
 - Automatic flow control
 - Transmission of PAUSE frames
 - Reception of PAUSE frame is handled within the MAC
- RX Filter (RXF) block: This module performs filtering on every receive packet to determine whether each packet should be accepted or rejected and includes:
 - CRC Check
 - 64-byte Pattern Match
 - Broadcast, Multicast and Unicast packets
 - Magic Packet™
 - 64-bit Hash Table
 - Runt Packet
- TX DMA/TX BM Engine: The TX DMA and TX Buffer Management engines perform data transfers from the memory (using descriptor tables) to the MAC Transmit Interface
- RX DMA/RX BM Engine: The RX DMA and RX Buffer Management engines transfer receive packets from the MAC to the memory (using descriptor tables)

Basic Ethernet Controller Operation

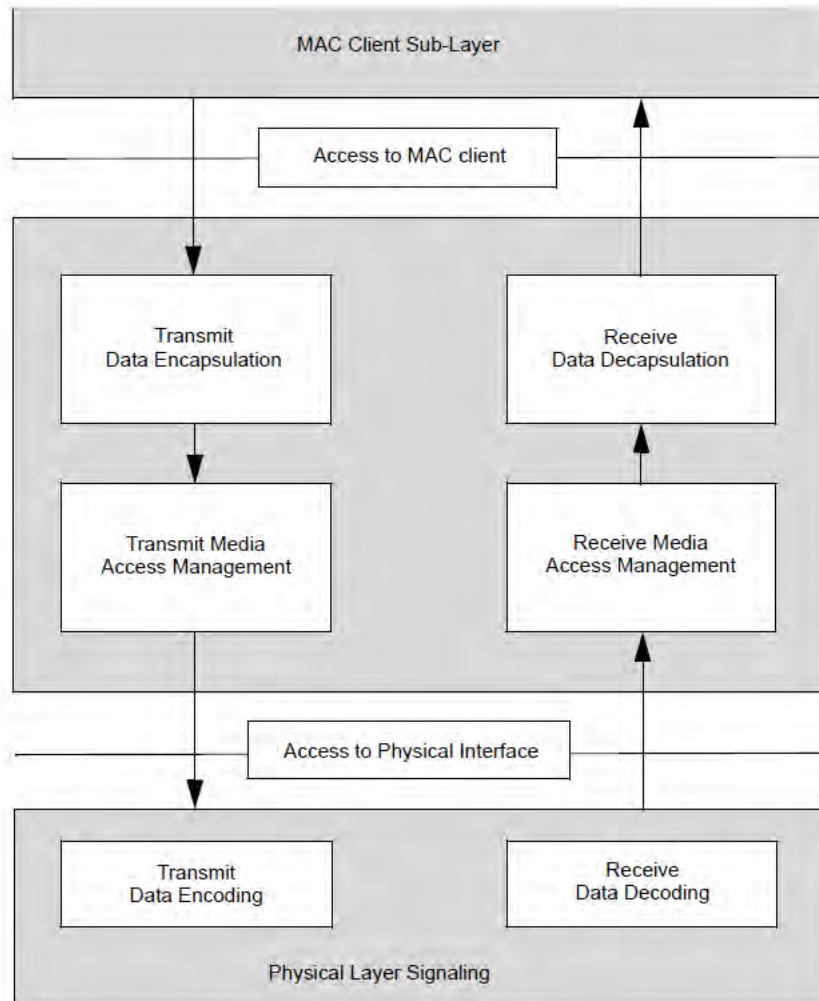
The Ethernet Controller is enabled using [PLIB_ETH_Enable](#) and is disabled using [PLIB_ETH_Disable](#). The Ethernet Controller is disabled by default after any Reset. If the Ethernet Controller is disabled, all of the I/O pins used for the MII/RMII and MIIM interfaces operate as port pins and are under the control of the respective PORTx latch bit and TRISx bit.

Disabling the controller resets the internal DMA state machines and all transmit and receive operations are aborted. The Special Function Registers (SFRs) are still accessible and their values are preserved.

Clearing the ON bit while the Ethernet Controller is active will abort all pending operations and reset the peripheral as previously defined.

Re-enabling the ON bit will restart the Ethernet Controller in its clean reset state while preserving the SFR values.

Ethernet Controller Section Block Diagram



MAC Overview

The MAC sub-layer is part of the functionality described in the OSI model for the Data Link Layer. It defines a medium-independent facility, built on the medium-dependent physical facility provided by the Physical Layer, and under the access-layer-independent LLC sub-layer or other MAC client. It is applicable to a general class of local area broadcast media suitable for use with the Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

The CSMA/CD MAC sub-layer provides services to the MAC client required for the transmission and reception of frames. The CSMA/CD MAC sub-layer makes a best effort to acquire the medium and transfer a serial stream of bits to the Physical Layer. Although certain errors are reported to the client, error recovery is not provided by the MAC.

The following is a summary of the functional capabilities of the CSMA/CD MAC sub-layer as shown in the previous diagram.

- For Frame Transmission:
 - Accepts data from the MAC client and constructs a frame
 - Presents a bit-serial data stream to the Physical Layer for transmission on the medium
 - In half-duplex mode, defers transmission of a bit-serial stream whenever the physical medium is busy
 - It can append proper Frame Check Sequence (FCS) value to outgoing frames and verifies full octet boundary alignment

- Delays transmission of frame bit stream for specified inter-frame gap period
- In half-duplex mode, halts transmission when collision is detected
- In half-duplex mode, schedules retransmission after a collision until a specified retry limit is reached
- In half-duplex mode, enforces collision to ensure propagation throughout network by sending jam message
- Prepends preamble and Start Frame Delimiter and appends FCS to all frame Appends PAD field for frames whose data length is less than a minimum value
- For Frame Reception:
 - Receives a bit-serial data stream from the Physical Layer
 - Presents the received frames to the MAC client (broadcast, multicast, unicast frames, etc.)
 - Checks incoming frames for transmission errors by way of FCS and verifies octet boundary alignment
 - Removes preamble, Start Frame Delimiter and PAD field (if necessary) from received frames
 - Implements the MII Management block that provides control/status connection to the external MII PHY

Note: Not all modes are available on all devices, please refer to the specific device data sheet to determine the set of modes supported for your device. In addition, refer to the **"Ethernet"** section in the family reference manual or the Ethernet Driver Library for more information.

5.6.8.4.2.1 Ethernet Frame Overview

Ethernet Frame Overview

IEEE 802.3-compliant Ethernet frames (packets) are between 64 and 1518 bytes long (Preamble and Start -of-Frame Delimiter not included). Frames containing less than 64 bytes are known as "runt" frames, while frames containing more than 1518 bytes are known as "huge" frames.

An Ethernet frame is made up of the following fields:

- Start-of-Stream/Preamble
- Start-of-Frame Delimiter (SFD)
- Destination MAC address (DA)
- Source MAC address (SA)
- Type/Length field
- Data Payload
- Optional Padding field
- Frame Check Sequence (FCS)
- Traffic on the actual physical cable

Please refer to the IEEE 802.3 Specification for detailed information about the Ethernet protocol.

5.6.8.4.2.2 Ethernet Controller Operation

Basic Ethernet Controller Operation

The Ethernet Controller is enabled using [PLIB_ETH_Enable](#) and is disabled using [PLIB_ETH_Disable](#). The Ethernet Controller is disabled by default after any Reset. If the Ethernet Controller is disabled, all of the I/O pins used for the MII/RMII and MIIM interfaces operate as port pins and are under the control of the respective PORT latch bit and TRIS bit.

Disabling the controller resets the internal DMA state machines and all transmit and receive operations are aborted. The SFRs are still accessible and their values preserved.

Clearing the ON bit while the Ethernet Controller is active will abort all pending operations and reset the peripheral as previously defined.

Re-enabling the ON bit will restart the Ethernet Controller in its clean reset state while preserving the SFR values.

5.6.8.4.2.3 Media Independent Interface (MII)

This section describes the Media Independent Interface (MII).

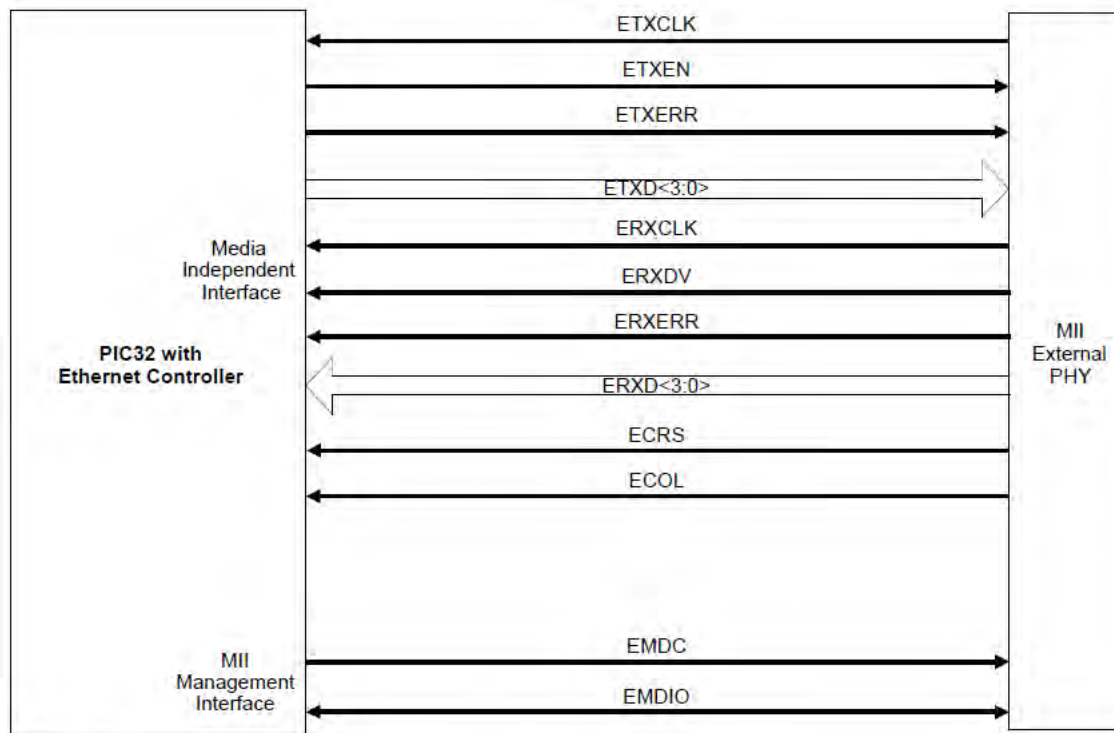
Description

Media Independent Interface (MII)

The Media Independent Interface (MII) is a standard interconnection between the MAC and the PHY for communicating TX and RX frame data. This MII has the following important characteristics:

- Capable of supporting 10/100 Mbps rates for data transfer, and offers support for management functions
- Provides independent four bit wide transmit and receive data paths
- Uses TTL signal levels, compatible with common digital CMOS processes
- Provides full-duplex operation

In 10 Mbps mode, the MII runs at 2.5 MHz; in 100 Mbps mode it runs at 25 MHz. PHYs that provide MII are not required to support both data rates, and may support either one or both.



5.6.8.4.2.4 Reduced Media Independent Interface (RMII)

This section describes the Reduced Media Independent Interface (RMII).

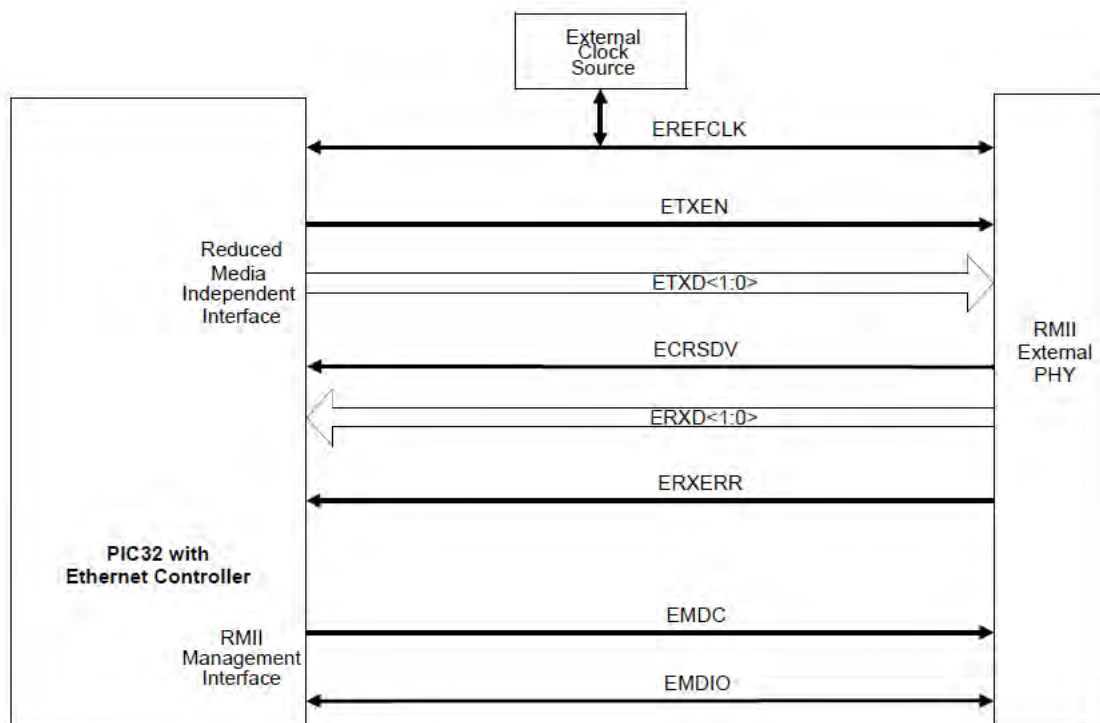
Description

Reduced Media Independent Interface (RMII)

The management interface (MDIO/MDC) is assumed to be identical to that defined in MII.

The RMII has the following characteristics:

- Capable of supporting 10 Mbps and 100 Mbps data rates
- Single clock reference for both MAC and the PHY (can be sourced from the MAC or from an external source)
- Provides independent 2 bit wide transmit and receive data paths
- Uses TTL signal levels, compatible with common digital CMOS processes
- Provides full-duplex operation
- The interface runs at 50 MHz



5.6.8.4.2.5 Flow Control Overview

Flow Control Overview

Ethernet flow control consists of the ability to **send** and receive PAUSE frames, which cause the receiving node to stop transmitting for a specific amount of time.

On the transmit side, the Flow Control (FC) block handles the hardware handshaking between the MAC and the CPU when the transmit flow control is enabled. Flow control for the received packets is part of the MAC functionality.

The PIC32 MAC supports both Symmetric PAUSE and Asymmetric PAUSE as described in Clause 28, Table 28B-2, and Clause 31 and Annex 31B of the IEEE 802.3 Specification.

The FC block supports two modes of operation: manual and automatic. In addition, the mode of transmission (Full-Duplex or

Half-Duplex) programmed into the MAC registers, is used by the FC block.

5.6.8.4.2.6 Receive Filtering Overview

Receive Filtering Overview

The Receive Filter (RXF) block examines all incoming receive packets and accepts or rejects the packet, based on user-selectable filters. The following RX filters are supported:

- CRC Error Acceptance Filter
- Runt Error Acceptance Filter
- CRC Check Rejection Filter
- Runt Rejection Filter
- Unicast Acceptance Filter
- Not Me Unicast Acceptance Filter
- Multicast Acceptance Filter
- Broadcast Acceptance Filter
- Hash Table Acceptance Filter
- Magic Packet Acceptance Filter
- Pattern Match Acceptance Filter with logical inversion

5.6.8.4.2.7 Ethernet DMA and Buffer Management Engine

Ethernet DMA and Buffer Management Engines

To reduce the overhead on the CPU to move the packet data between data memory and the Ethernet controller, internal receive (RX) and transmit (TX) DMA engines are integrated into the module. The DMA engines are responsible for transferring data from system memory to the MAC for transmit packets and for transferring data from the MAC to system memory for receive packets. The DMA engines each have access to the system memory by acting as two different bus masters, one bus master for transmit and one for receive.

The DMA engines use separate Ethernet Descriptor Tables (EDTs) for TX and RX operations to determine where the TX/RX packet buffer resides in the system memory.

Both transmit and receive descriptors, called Ethernet Descriptors (EDs), used by the DMA engines have a similar format, with only the status word formats being different.

Refer to the "**Ethernet**" section in the family reference manual for more information.

5.6.8.4.3 Sample Code

This section provides information for initialization, transmit, and receive using code samples.

Description

Sample Code

The basic functionality of the Ethernet Peripheral Library is grouped into these sections:

- Initialization
- Transmit
- Receive

For a properly functioning Ethernet port, development of a data queue and use of interrupts will be necessary.

For this additional functionality, see the **"Ethernet"** section in the family reference manual or the Ethernet Driver Library for more information.

Initialization Sequence

The Ethernet Controller is enabled using [PLIB_ETH_Enable](#) and is disabled using [PLIB_ETH_Disable](#). The Ethernet Controller is disabled by default after any Reset.

After being Enabled, disabling the controller resets the internal DMA state machines and all transmit and receive operations are aborted. All registers remain accessible and their values are preserved. It is important to reset the PHY and MII Management registers to ensure the PHY is in a known initialized state.

Before enabling the module for transmitting or receiving packets, there are several registers that must be initialized. To initialize the Ethernet Controller to receive and transmit, Microchip recommends the following sequence:

1. Begin with the module.
 - Disable the CPU Ethernet interrupt
 - Turn the Ethernet Controller off
 - Disable receive
 - Disable transmit
 - Wait until the Ethernet module is no longer busy
 - Disable all Ethernet Interrupts
 - Clear the Ethernet Interrupt Pending Flag(s)
 - Clear the transmit and receive start addresses
2. MAC initialize.
 - Reset the MAC
 - A configuration fuse (FETHIO) setting shows which pins will be used for connection to the PHY
 - A configuration fuse (FMIIEN) settings shows whether MII or RMII will be used
3. PHY Initialize is PHY dependent. Common procedures (which may vary by PHY vender) include the following:
 - Reset PHY (using Control Register 0)
 - Set the MII/RMII operation mode
 - Set the normal, swapped, or auto (preferred)MDIX
 - Check the PY capabilities (using Status Register 1)
 - Preferably the auto-negotiation should be selected if the PHY supports it.
 - Expose the supported capabilities (extended Register 4)
 - Start the negotiation (using Control Register 0)
 - wait for the negotiation to complete and get the link partner capabilities (extended Register 5)
 - negotiation result (vendor-specific register)
 - If auto-negotiation is not supported (or not selected)
 - update the PHY Duplex and speed settings (use Control Register 0 or vender-specific register)
4. MAC Configuration
 - Enable MAC receive and pause both transmit and receive direction control frames.
 - Select the desired auto-padding, duplex, huge frame, and CRC capabilities.

- Program back-to-back inter-packet gap and non-back-to-back interpacket gap.
- Program collision window and maximum retransmissions.
- Set Maximum frame length.
- Optionally set the station MAC address(es) (or let them default to the factory value).

5. Continue Ethernet Controller initialization:

- If planning to turn on flow control, update the Pause Timer Value
- If using auto flow control, setup the full/empty watermarks and enable it
- Set the receive filters
- Set receive buffer size - using packets that are too small impacts performance
- Prepare the list/ring of transmit descriptors - refer to the device data sheet for more information
- Prepare the list of receive descriptors populated with valid buffers for messages to be received - refer to the device data sheet for more information
- Update the transmit buffer start address and the receive buffer start address
- Enable the Ethernet Controller and wait until the module is ready
- Enable Ethernet Receive.

Receive Loop

1. Wait until the receive buffer count has been incremented.
2. Inspect the receive descriptor to see whether it is still owned by the Ethernet Module.
3. If the receive descriptor is still owned by the Ethernet module, nothing has been received. Loop to 1.
4. Otherwise, a message was received and software has control of the memory where it is stored.
5. Extract a message and do something with it.
6. Reinitialize buffer descriptor and return receive buffer to list and decrement the receive packet buffer count.
7. Loop.

Transmit Loop

1. Software writes the packet data to packet data memory. A pointer to the packet data is inserted into the transmit buffer descriptor.
2. Update the necessary number of transmit descriptors, starting from the head of the list - setting the data buffer address (large packet fragmentation will impact performance).
3. Update the byte count for each descriptor with the number of bytes contained in each buffer.
4. Set the flag in each transmit descriptor that belongs to the packet to give the descriptor to the Ethernet module for transmission.
5. Enable the transmission.
6. Wait until the Ethernet module returns the transmit buffer descriptor ownership to software.
7. Inspect the results of the transmission.
8. Loop.

The packet transmit process is as follows:

1. Software writes the packet data to a packet buffer in data memory and programs a buffer descriptor entry to point to the packet data buffer. The buffer descriptor is used to define the Transmission Packet data buffer location and length, and its address is written to the Ethernet Controller.
2. Software then calls the TransmitRequestToSendEnable function, which starts the TX DMA and TXBM logic.

3. After the TX DMA engine reads the DATA_BUFFER_ADDRESS value, the engine will begin reading the packet data from the location read from the descriptor.
4. The TXBM indicates the start-of-frame to the MAC and transmits the entire frame data from the transmit buffer, until the end address is reached. The TXBM simply transmits from the start address until the specified number of bytes has been transmitted to the MAC transmit interface.
5. The MAC can retry a transmission due to an early collision.
6. The MAC can abort a transmission due to a late collision, excessive collisions or excessive defers. This condition is signaled by TXABORT bit (ETHIRQ<2>).
7. Once transmission has completed, the TX DMA engine stores the relevant bits of the TSV into the first descriptor entry of the packet.
8. After the packet has been transmitted, all the descriptors used for the transmission are released to the software via the EOWN bits being cleared.
9. If more valid TX descriptors are available (EOWN = 1), the DMA engine will go back to step 3 to begin the next packet's transmission. Otherwise, if the next descriptor is still owned by hardware (EOWN = 0), transmission will halt and wait for the software to set the TXRTS bit again.

Note that any collision that occurs within the CWINDOW bit (EMAC1CLRT<8:14>) boundary is an early collision and results in a Retry operation. A collision that happens beyond the CWINDOW boundary will be treated as a late collision and will cause an abort. The CWINDOW bit in the EMAC1CLRT register is typically set to 64 bytes. An abort condition can also result from reaching the maximum collision count RETX bit (EMAC1CLRT<0:3>) for a packet trying to be sent.

See the "Ethernet" section in the family reference manual or the Ethernet Driver Library for more information.

5.6.8.4.3.1 Initialization

```
#include "plib_eth.h"
```

```

/*****
*/
void my_ETH_Initialize(ETH_MODULE_ID id)
{
    //Disable System (CPU) Interrupt associated with Ethernet Peripheral
    // using the Interrupt Peripheral Library
    // Example: PLIB_INT_SourceDisable(PLIB_INT_SOURCE_ETH_1);

    //Disable Ethernet Peripheral in case it was previously enabled
    PLIB_ETH_Disable(ETH_MODULE_ID1);

    //Disable any Ethernet Controller interrupt generation:
    PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID1, ETH_ALL_INTERRUPT_SOURCES);

    PLIB_ETH_RxDisable(ETH_MODULE_ID1);
    PLIB_ETH_TxRTSDisable(ETH_MODULE_ID1);

    while (PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID1)) { /*wait*/ }

    //Clear the System Interrupt Flag associated with Ethernet Peripheral
    // using the Interrupt Peripheral Library
    // Example: PLIB_INT_SourceFlagClear(PLIB_INT_SOURCE_ETH_1);

    //////////////////////////////////////
    //MAC Initialize
    PLIB_ETH_MIIResetEnable(ETH_MODULE_ID1);
    PLIB_ETH_MIIResetDisable(ETH_MODULE_ID1);

    //A configuration fuse (FETHIO) setting shows which digital pins need to be configured
    if (CONFIGURATION_FUSE_PRIMARY_ETH_PINS==1)
    {
        ConfigurePrimaryEthernetPinsAsDigital();
    }
}

```

```

}
else //(CONFIGURATION_FUSE_PRIMARY_ETH_PINS==0)
{
    ConfigureAlternateEthernetPinsAsDigital();
}

//A configuration fuse (FMIEN) settings shows which digital pins need to be configured
if (CONFIGURATION_FUSE_MII_STYLE==RMII)
{
    PLIB_ETH_RMIIResetEnable(ETH_MODULE_ID1);
    PLIB_ETH_RMIIResetDisable(ETH_MODULE_ID1);
    PLIB_ETH_RMIIISpeedSet(ETH_MODULE_ID1, ETH_RMII_100Mbps);
}
PLIB_ETH_MIIResetEnable(ETH_MODULE_ID1);
PLIB_ETH_MIIResetDisable(ETH_MODULE_ID1);
PLIB_ETH_MIIMClockSet(ETH_MODULE_ID1, ETH_MII_SYSCCLK_DIV_BY_10);

////////////////////
//PHY Initialize is PHY dependent.
// Common procedures (which may vary by PHY vender) include the following:
//
//Reset PHY (using Control Register 0)
//Set the MII/RMII operation mode
//Set the normal, swapped, or auto (preferred)MDIX
//Check the PY capabilities (using Status Register 1)
//Preferably the auto-negotiation should be selected if the PHY supports it.
//Expose the supported capabilities (extended Register 4)
//Start the negotiation (using Control Register 0)
//wait for the negotiation to complete and get the link partner capabilities (extended
Register 5)
//negotiation result (vendor-specific register)
//If auto-negotiation is not supported (or not selected)
//update the PHY Duplex and speed settings (use Control Register 0 or vender-specific
register)

////////////////////
//MAC Configuration
PLIB_ETH_RxEnable(ETH_MODULE_ID1);
PLIB_ETH_TxPauseEnable(ETH_MODULE_ID1);
PLIB_ETH_RxPauseEnable(ETH_MODULE_ID1);

//Select the desired auto-padding, duplex, huge frame, and CRC capabilities
PLIB_ETH_AutoDetecPadSet(ETH_MODULE_ID1, ETH_AUTOPAD_BY_PKT_TYPE_ADD_CRC );
PLIB_ETH_FullDuplexEnable(ETH_MODULE_ID1); //or disable for half-duplex
PLIB_ETH_CRCEnable(ETH_MODULE_ID1); //or disable if EMAC does not add CRC when no padding
is necessary
PLIB_ETH_HugeFrameDisable(ETH_MODULE_ID1); //or enable to receive huge frames

//Program back-to-back inter-packet gap and non-back-to-back interpacket gap
PLIB_ETH_BackToBackIPGpSet(ETH_MODULE_ID1, 0x15 /*backToBackGapValue*/);
PLIB_ETH_NonBackToBackIPG1Set(ETH_MODULE_ID1, 0x0C /*nonBackToBackGap1Value*/);
PLIB_ETH_NonBackToBackIPG2Set(ETH_MODULE_ID1, 0x12 /*nonBackToBackGap2Value*/);

//Set Maximum frame length
PLIB_ETH_MaxFrameLengthSet(ETH_MODULE_ID1, 1527 /*maxFrameLenValue*/);

//Optionally set the station MAC address(es) (or let them default to the factory value)
PLIB_ETH_StationAddressSet(ETH_MODULE_ID1, 1, 0x13 /*stationAddr1Value*/);

////////////////////
//Continue Ethernet Controller Initialization

//Set receive buffer size - using packets that are too small impacts performance
PLIB_ETH_ReceiveBufferSizeSet(ETH_MODULE_ID1, 200 /*receiveBufSizeValue*/);

```

```

    //Prepare the list/ring of transmit descriptors - refer to the device data sheet for more
    information
    //Prepare the list of receive descriptors populated with valid buffers for messages to be
    received - refer
    //to the device data sheet for more information
    //Update the transmit buffer start address and the receive buffer start address
    PLIB_ETH_TxPacketDescAddrSet(ETH_MODULE_ID1, (uint8_t
*)0xadd1add0/*txPktDescStartAddrValue*/);
    PLIB_ETH_RxPacketDescAddrSet(ETH_MODULE_ID1, (uint8_t
*)0xadd3add2/*rxPktDescStartAddrValue*/);

    //If using auto flow control, setup the full/empty watermarks and enable it
    PLIB_ETH_PauseTimerSet(ETH_MODULE_ID1, 0x64 /*pauseTimerValue*/);
    PLIB_ETH_RxEmptyWmarkSet(ETH_MODULE_ID1, 0x3F/*emptyRxWatermarkValue*/);
    PLIB_ETH_RxFullWmarkSet(ETH_MODULE_ID1, 0x10 /*fullRxWatermarkValue*/);
    PLIB_ETH_AutoFlowControlEnable(ETH_MODULE_ID1);

    PLIB_ETH_ReceiveFilterEnable(ETH_MODULE_ID1,
                                ETH_MULTICAST_FILT |    // enable to accept multicast address
packets
                                ETH_BROADCAST_FILTER );// enable to accept all broadcast
address packets

    PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID1, ETH_ALL_INTERRUPT_SOURCES);

    //Enable the Ethernet Controller and wait until the module is ready
    PLIB_ETH_Enable(ETH_MODULE_ID1);
    while (PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID1))
    {
        //Wait
    }

    PLIB_ETH_RxEnable(ETH_MODULE_ID1);

    return (EXIT_SUCCESS);
}

```

5.6.8.4.3.2 Descriptor Table Example

```

typedef volatile struct
{
    union
    {
        //Common to both Receive and Transmit
        struct
        {
            unsigned: 7;
            unsigned EOWN: 1; //0=Software owns, 1=Ethernet Controller owns
            unsigned NPV: 1; //0=Next Desc follows this, 1=Next Desc pointed to by NEXT_ED
            unsigned: 7;
            unsigned bCount: 11;
            unsigned: 3;
            unsigned EOP: 1;
            unsigned SOP: 1;
        };
        uint32_t w;
    };
    // descriptor header

    //Common to both Receive and Transmit
    uint8_t* pEDBuff;    // pointer to data buffer

    union
    {
        //This portion differs from Receive to Transmit
        uint64_t RX_STATUS;    //Receive Status
        struct
        {

```

```

    unsigned RX_PKT_CHECKSUM: 16;
    unsigned: 8;
    unsigned RXF_RSV: 8;
    unsigned RSV: 32;
};

uint64_t TX_STATUS;    // Transmit Status
struct //Transmit Struct
{
    unsigned TSV_TRANSMIT_BYTE_COUNT: 16;
    unsigned TSV_TX_COLLISION_COUNT: 4;
    unsigned TSV_TX_CRC_ERROR: 1;
    unsigned TSV_TX_LENGTH_CHECK_ERROR: 1;
    unsigned TSV_TX_DONE: 1;
    unsigned TSV_TX_MULTICAST: 1;
    unsigned TSV_TX_BROADCAST: 1;
    unsigned TSV_TX_PACKET_DEFER: 1;
    unsigned TSV_TX_EXCESSIVE_DEFER: 1;
    unsigned TSV_TX_MAXIMUM_COLLISION: 1;
    unsigned TSV_TX_LATE_COLLISION: 1;
    unsigned TSV_TX_GIANT: 1;
    unsigned TSV_TX_UNDER_RUN: 1;
    unsigned TSV_TX_STATUS_VECTOR: 1;

    unsigned TSV_TOTAL_BYTES_TX_ON_WIRE: 16;
    unsigned TSV_TX_CONTROL_FRAME: 1;
    unsigned TSV_TX_PAUSE_CONTROL_FRAME: 1;
    unsigned TSV_TX_BACKPRESSURE_APPLIED: 1;
    unsigned TSV_TX_VLAN_TAGGED_FRAME: 1;
    unsigned TSV_IGNORED: 4;
    unsigned TSV_USER_DEFINED: 8;
};

}; // descriptor header

//Common to both Receive and Transmit
uint8_t* next_ed; // next descriptor (NPV==1);

}__attribute__((__packed__)) eth_buffer_descriptor_t; // hardware RX descriptor (linked).

```

For more information on this structure, refer to the **"Ethernet"** section in the family reference manual.

5.6.8.4.3.3 Transmit

C

Description

```
#include "plib_eth.h"
```

```

/*****
//
*/

```

```

void my_ETH_Transmit(ETH_MODULE_ID id, eth_buffer_descriptor_t *pArrDcpt, uint8_t *pArrBuff,
uint8_t nArrayItems, uint16_t *pArrSize)
{
    //Input Parameters:
    // *pArrDcpt : pointer to an array that holds free descriptors
    // *pArrBuff: pointer to an array that holds buffers to be transmitted
    // *pArrSize: pointer to an array that holds the sizes of the buffers to be transmitted
    // nArrayItems: number of buffers to be transmitted
    //
    // Before attempting to transmit, the data must be placed into a packet
    // buffer (list) and the point(s) must be placed into transmit descriptor(s)

```

```

// Refer to the DRV_ETH or the device family reference manual for more information

extern void* VA_TO_PA(char* pBuff); // extern function that returns the physical
// address of the input virtual address parameter

uint16_t ix; // loop index
eth_buffer_descriptor_t *pEDcpt; // current Ethernet descriptor
eth_buffer_descriptor_t *tailDcpt; // last Ethernet descriptor
uint8_t *pBuff; // current data buffer to be transmitted
uint16_t *pSize; // current data buffer size

pEDcpt=pArrDcpt;
pBuff=pArrBuff;
pSize=pArrSize;
tailDcpt=0;

for(ix=0; ix< nArrayItems; ix++, pEDcpt++, pBuff++, pSize++)
{
    // pass the descriptor to hw, use linked descriptors, set proper size
    pEDcpt->pEDBuff = (uint8_t *)VA_TO_PA(pBuff); // set buffer
    pEDcpt->w=0; // clear all the fields
    pEDcpt->NPV=1; // set next pointer valid
    pEDcpt->EOWN=1; // set hw ownership
    pEDcpt->bCount = *pSize; // set proper size
    if(tailDcpt)
    {
        tailDcpt->next_ed = VA_TO_PA(&pEDcpt);
    }
    tailDcpt=pEDcpt;
}
// at this moment pEDcpt is an extra descriptor we use to end the descriptors list
pEDcpt->w=0; // software ownership
tailDcpt->next_ed= VA_TO_PA(&pEDcpt);
pArrDcpt[0].SOP=1; // start of packet
pArrDcpt[nArrayItems-1].EOP=1; // end of packet

PLIB_ETH_TransmitPacketDescStartAddrSet(id, VA_TO_PA(pArrDcpt)); // set the transmit
address
PLIB_ETH_TransmitRequestToSendIsEnabled(id); // set the TXRTS

// the ETHC will transmit the buffers we just programmed
/* do something else in between */
while (PLIB_ETH_TxRTSIsEnabled(id))
{
    //Wait until transmit is not busy
}

// check the ETHSTAT register to see the transfer result
}

```

5.6.8.4.3.4 Receive

```

#include "plib_eth.h"

/*****
//
*/
void my_ETH_Receive(ETH_MODULE_ID id, eth_buffer_descriptor_t *pArrDcpt, uint8_t *pArrBuff,
uint8_t nArrayItems, uint16_t rxBuffSize)
{
    extern void* VA_TO_PA(char* pBuff); // extern function that returns the physical
// address of the input virtual address parameter

    int ix; // index
    eth_buffer_descriptor_t *pEDcpt; // current Ethernet descriptor
    eth_buffer_descriptor_t *tailDcpt; // last Ethernet descriptor
    char* pBuff; // current data buffer to be transmitted

```

```

pEDcpt=pArrDcpt;
pBuff=pArrBuff;
tailDcpt=0;

//Prepare to receive
PLIB_ETH_ReceiveBufferSizeSet(id, (rxBuffSize/16));

for(ix=0; ix< nArrayItems; ix++, pEDcpt++, pBuff++)
{
    // pass the descriptor to hw, use linked descriptors, set proper size
    pEDcpt->pEDBuff=(unsigned char*)VA_TO_PA(pBuff); // set buffer
    pEDcpt->w=0; // clear all the fields
    pEDcpt->NPV= 1; // set next pointer valid
    pEDcpt->EOWN= 1; // set hw ownership

    if(tailDcpt)
    {
        tailDcpt->next_ed=VA_TO_PA(&pEDcpt);
    }
    tailDcpt=pEDcpt;
}
// at this moment pEDcpt is an extra descriptor we use to end the descriptors list
pEDcpt->w=0; // software ownership
tailDcpt->next_ed= VA_TO_PA(&pEDcpt);

PLIB_ETH_RxPacketDescStartAddrSet(id, VA_TO_PA(pArrDcpt));
// set the address of the first RX descriptor
PLIB_ETH_ReceiveEnable(id);
// the Ethernet Controller will receive frames and place them in the receive buffers
// we just programmed
/* do something else in between */

while(PLIB_ETH_RxPacketCountGet(id)==0)
{
    //Wait until something is received.
    // Packets are received as set by the receive filter
}

//Receive the packet
//doSomethingWithReceivedPacket()
// After the received packet has been processed, restore EOWN to 1 and
PLIB_ETH_RxBufferCountDecrement(id);
}

```

5.6.8.4.4 Support for Legacy "Ethernet Controller Library"

These routines (PLIB_ETH_Legacy*) provide MPLAB Harmony support for applications that use the "Ethernet Controller Library for Microchip PIC32MX Microcontrollers" that is installed with XC32 compilers. (See <Microchip\xc32\v1.20\docs\pic32-lib-help\hlpETH.chm> for information on this legacy support.)

An additional argument, an index value from the ETH_MODULE_ID enumeration, has been added to allow these functions to work from within the context of this MPLAB Harmony Peripheral Library.

Legacy Controller Library	MPLAB Harmony
EthClose	DRV_ETHMAC_LegacyClose
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthEventsClr	PLIB_ETH_EventsClr
EthEventsEnableClr	PLIB_ETH_EventsEnableClr
EthEventsEnableGet	PLIB_ETH_EventsEnableGet
EthEventsEnableSet	PLIB_ETH_EventsEnableSet
EthEventsEnableWrite	PLIB_ETH_EventsEnableWrite
EthEventsGet	PLIB_ETH_EventsGet
EthInit	DRV_ETHMAC_LegacyInit
EthMACGetAddress	PLIB_ETH_MACGetAddress
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthMACSetAddress	PLIB_ETH_MACSetAddress
EthMACSetMaxFrame	PLIB_ETH_MACSetMaxFrame
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxFiltersClr	PLIB_ETH_RxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_RxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_RxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_RxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_RxFiltersSet
EthRxFiltersWrite	PLIB_ETH_RxFiltersWrite
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthRxSetBufferSize	PLIB_ETH_RxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_StatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_StatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_StatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_StatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_StatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_StatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_StatTxSColCnt
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

Ethernet Peripheral Library	Legacy Controller Library
PLIB_ETH_EventsClr	EthEventsClr
PLIB_ETH_EventsEnableClr	EthEventsEnableClr
PLIB_ETH_EventsEnableGet	EthEventsEnableGet
PLIB_ETH_EventsEnableSet	EthEventsEnableSet
PLIB_ETH_EventsEnableWrite	EthEventsEnableWrite
PLIB_ETH_EventsGet	EthEventsGet
PLIB_ETH_MACGetAddress	EthMACGetAddress
PLIB_ETH_MACSetAddress	EthMACSetAddress
PLIB_ETH_MACSetMaxFrame	EthMACSetMaxFrame
PLIB_ETH_RxFiltersClr	EthRxFiltersClr
PLIB_ETH_RxFiltersHTSet	EthRxFiltersHTSet
PLIB_ETH_RxFiltersPMClr	EthRxFiltersPMClr
PLIB_ETH_RxFiltersPMSet	EthRxFiltersPMSet
PLIB_ETH_RxFiltersSet	EthRxFiltersSet
PLIB_ETH_RxFiltersWrite	EthRxFiltersWrite
PLIB_ETH_RxSetBufferSize	EthRxSetBufferSize
PLIB_ETH_StatRxAlignErrCnt	EthStatRxAlignErrCnt
PLIB_ETH_StatRxFcsErrCnt	EthStatRxFcsErrCnt
PLIB_ETH_StatRxOkCnt	EthStatRxOkCnt
PLIB_ETH_StatRxOvflCnt	EthStatRxOvflCnt
PLIB_ETH_StatTxMColCnt	EthStatTxMColCnt
PLIB_ETH_StatTxOkCnt	EthStatTxOkCnt
PLIB_ETH_StatTxSColCnt	EthStatTxSColCnt

5.6.8.5 Library Interface

Data Types and Constants

	Name	Description
	<u>ETH_AUTOPAD_OPTION</u>	Lists the possible automatic padding configurations.
	<u>ETH_INTERRUPT_SOURCES</u>	Lists the possible values of ETH_INTERRUPT_SOURCES.
	<u>ETH_MIIM_CLK</u>	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).
	<u>ETH_PATTERN_MATCH_MODE</u>	Lists the possible pattern match values.
	<u>ETH_RECEIVE_FILTER</u>	Lists the possible values of the receive filter.
	<u>ETH_RMII_SPEED</u>	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

Control Functions

	Name	Description
	<u>PLIB_ETH_MCSRxResetDisable</u>	Disables the reset EMAC Control Sub-layer/Receive domain logic.
	<u>PLIB_ETH_MCSRxResetEnable</u>	Enables the reset EMAC Control Sub-layer/Receive domain logic.
	<u>PLIB_ETH_MCSRxResetIsEnabled</u>	Gets the EMAC Control Sub-layer/Receive domain logic reset status.
	<u>PLIB_ETH_MCSTxResetDisable</u>	Disables the reset EMAC Control Sub-layer/Transmit domain logic.
	<u>PLIB_ETH_MCSTxResetEnable</u>	Enables the reset of EMAC Control Sub-layer/Transmit domain logic.

	PLIB_ETH_MCSTxResetIsEnabled	Gets the EMAC Control Sub-layer/Transmit domain logic reset status.
	PLIB_ETH_MIIMReadDataGet	Gets EMAC MIIM management read data after a MII read cycle has completed.
	PLIB_ETH_MIIMReadStart	Initiates an MII management read command.
	PLIB_ETH_MIIMResetDisable	Disables EMAC Reset MII Management.
	PLIB_ETH_MIIMResetEnable	Enables EMAC Reset Media Independent Interface (MII) Management.
	PLIB_ETH_MIIMResetIsEnabled	Gets the EMAC Reset MII Management enable status.
	PLIB_ETH_MIIMScanIncrEnable	Enables EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrDisable	Disables the EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrIsEnabled	Gets the EMAC MIIM scan increment enable status.
	PLIB_ETH_MIIMScanModeDisable	Disables MIIM scan mode.
	PLIB_ETH_MIIMScanModeEnable	Enables MIIM scan mode.
	PLIB_ETH_MIIMScanModeIsEnabled	Gets the MII management scan enable status.
	PLIB_ETH_MIIMWriteDataSet	Sets the EMAC MIIM write data before initiating an MII write cycle.
	PLIB_ETH_MIIMWriteStart	Initiates an MII management write command.
	PLIB_ETH_MIIRResetDisable	Disables the EMAC Soft reset.
	PLIB_ETH_MIIRResetEnable	Enables the EMAC MIIM Soft reset.
	PLIB_ETH_MIIRResetIsEnabled	Gets EMAC MIIM Soft Reset enable status.
	PLIB_ETH_PHYAddressGet	Gets the EMAC MIIM management PHY address.
	PLIB_ETH_PHYAddressSet	Sets the EMAC MIIM PHY address.
	PLIB_ETH_RegisterAddressGet	Gets the EMAC MIIM management register address.
	PLIB_ETH_RegisterAddressSet	Sets EMAC MIIM register address.
	PLIB_ETH_RMIIResetDisable	Disables EMAC Reset RMII.
	PLIB_ETH_RMIIResetEnable	Enables EMAC Reset RMII.
	PLIB_ETH_RMIIResetIsEnabled	Gets the EMAC Reset RMII enable status.
	PLIB_ETH_RxBufferCountDecrement	Causes the Receive Descriptor Buffer Counter to decrement by 1.
	PLIB_ETH_RxDisable	Disables the Ethernet module receive logic.
	PLIB_ETH_RxEnable	Enables the Ethernet receive logic.
	PLIB_ETH_RxFuncResetDisable	Disables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetEnable	Enables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetIsEnabled	Gets the EMAC reset receive function status.
	PLIB_ETH_RxIsEnabled	Gets the Ethernet module receive enable status.
	PLIB_ETH_RxPacketDescAddrGet	Gets the address of the next receive descriptor.
	PLIB_ETH_RxPacketDescAddrSet	Sets the Ethernet module receive packet descriptor start address.
	PLIB_ETH_TestPauseDisable	Disables EMAC test pause.
	PLIB_ETH_TestPauseEnable	Enables EMAC test pause.
	PLIB_ETH_TestPauseIsEnabled	Gets the EMAC test pause enable status.
	PLIB_ETH_TxFuncResetDisable	Disables the EMAC Transmit function reset.
	PLIB_ETH_TxFuncResetEnable	Enables the EMAC transmit function reset.
	PLIB_ETH_TxFuncResetIsEnabled	Gets the EMAC Transmit function reset status.
	PLIB_ETH_TxPacketDescAddrGet	Gets the address of the next descriptor to be transmitted.
	PLIB_ETH_TxPacketDescAddrSet	Sets the Ethernet module transmit packet descriptor start address.
	PLIB_ETH_TxRTSDisable	Aborts an Ethernet module transmission.
	PLIB_ETH_TxRTSEnable	Enables the Ethernet transmit request to send .
	PLIB_ETH_TxRTSIsEnabled	Gets the Ethernet module transmit request to send status.

	PLIB_ETH_CRCDisable	Disables EMAC CRC.
	PLIB_ETH_CRCEnable	Enables EMAC CRC.
	PLIB_ETH_CRCIsEnabled	Gets the EMAC CRC enable status.

Existence Functions

	Name	Description
	PLIB_ETH_ExistsAlignmentErrorCount	Identifies whether the AlignmentErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsAutoFlowControl	Identifies whether the AutoFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionCounts	Identifies whether the CollisionCounts feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionWindow	Identifies whether the CollisionWindow feature exists on the Ethernet module.
	PLIB_ETH_ExistsEnable	Identifies whether the Enable feature exists on the Ethernet module.
	PLIB_ETH_ExistsEthernetControllerStatus	Identifies whether the EthernetControllerStatus feature exists on the Ethernet module.
	PLIB_ETH_ExistsFCSErrorCount	Identifies whether the FCSErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsFramesTransmittedOK	Identifies whether the FramesTransmittedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsFrameReceivedOK	Identifies whether the FrameReceivedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsHashTable	Identifies whether the HashTable feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterPacketGaps	Identifies whether the InterPacketGaps feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterrupt	Identifies whether the Interrupt feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Configuration	Identifies whether the MAC_Configuration feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Resets	Identifies whether the MAC_Resets feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Testing	Identifies whether the MAC_Testing feature exists on the Ethernet module.
	PLIB_ETH_ExistsManualFlowControl	Identifies whether the ManualFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsMaxFrameLength	Identifies whether the MaxFrameLength feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Config	Identifies whether the MIIM_Config feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Indicators	Identifies whether the MIIM_Indicators feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMAddresses	Identifies whether the MIIMAddresses feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMReadWrite	Identifies whether the MIIMReadWrite feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMScanMode	Identifies whether the MIIMScanMode feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMWriteReadData	Identifies whether the MIIMWriteReadData feature exists on the Ethernet module.

	PLIB_ETH_ExistsPatternMatch	Identifies whether the PatternMatch feature exists on the Ethernet module.
	PLIB_ETH_ExistsPauseTimer	Identifies whether the PauseTimer feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveBufferSize	Identifies whether the ReceiveBufferSize feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveFilters	Identifies whether the ReceiveFilters feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveOverflowCount	Identifies whether the ReceiveOverflowCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveWmarks	Identifies whether the ReceiveWmarks feature exists on the Ethernet module.
	PLIB_ETH_ExistsRetransmissionMaximum	Identifies whether the RetransmissionMaximum feature exists on the Ethernet module.
	PLIB_ETH_ExistsRMII_Support	Identifies whether the RMII_Support feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxBufferCountDecrement	Identifies whether the RxBufferCountDecrement feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxEnable	Identifies whether the ReceiveEnable feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxPacketDescriptorAddress	Identifies whether the RxPacketDescriptorAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStationAddress	Identifies whether the StationAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Ethernet module.
	PLIB_ETH_ExistsTransmitRTS	Identifies whether the TransmitRTS feature exists on the Ethernet module.
	PLIB_ETH_ExistsTxPacketDescriptorAddress	Identifies whether the TxPacketDescriptorAddress feature exists on the Ethernet module.

Filtering Functions

	Name	Description
	PLIB_ETH_HashTableGet	Gets the value of the Hash table.
	PLIB_ETH_HashTableSet	Sets the Ethernet module Hash table with the new value.
	PLIB_ETH_PassAllDisable	Disables the EMAC PassAll.
	PLIB_ETH_PassAllEnable	Enables the EMAC PassAll.
	PLIB_ETH_PassAllIsEnabled	Gets the EMAC PassAll enable status.
	PLIB_ETH_PatternMatchChecksumGet	Gets the value of the pattern match checksum register.
	PLIB_ETH_PatternMatchChecksumSet	Sets the Ethernet module pattern match checksum register with the new value.
	PLIB_ETH_PatternMatchGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeSet	Sets the Ethernet module pattern match mode.
	PLIB_ETH_PatternMatchOffsetGet	Gets the value of the patter match offset register.
	PLIB_ETH_PatternMatchOffsetSet	Sets the Ethernet module patter match offset register with the new value.
	PLIB_ETH_PatternMatchSet	Sets the Ethernet module pattern match mask register with the new value.
	PLIB_ETH_PurePreambleDisable	Disables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleEnable	Enables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleIsEnabled	Gets EMAC pure preamble enforcement enable status.

	PLIB_ETH_ReceiveFilterDisable	Disables the specified receive filter.
	PLIB_ETH_ReceiveFilterEnable	Enables the specified receive filter.
	PLIB_ETH_ReceiveFilterIsEnable	Disables the specified receive filter.
	PLIB_ETH_StationAddressGet	Gets the selected EMAC station address.
	PLIB_ETH_StationAddressSet	Sets the selected EMAC Station Address.

Flow Control Functions

	Name	Description
	PLIB_ETH_AutoFlowControlDisable	Disables the Ethernet module Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlEnable	Enables the Ethernet Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlsEnabled	Gets the Ethernet module Automatic Flow Control status.
	PLIB_ETH_BackPresNoBackoffDisable	Disables EMAC back pressure/no backoff.
	PLIB_ETH_BackPresNoBackoffEnable	Enables EMAC back pressure/no back-off.
	PLIB_ETH_BackPresNoBackoffIsEnabled	Gets the EMAC back pressure/no back-off enable status.
	PLIB_ETH_ManualFlowControlDisable	Disable Ethernet module Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlEnable	Enables the Ethernet Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlsEnabled	Gets the Ethernet module Manual Flow Control enable status.
	PLIB_ETH_NoBackoffDisable	Disables EMAC no back-offs.
	PLIB_ETH_NoBackoffEnable	Enables EMAC no backoff.
	PLIB_ETH_NoBackoffIsEnabled	Gets the EMAC no back-off enable status.
	PLIB_ETH_RxEmptyWmarkGet	Gets the Ethernet module receive empty watermark.
	PLIB_ETH_RxEmptyWmarkSet	Sets the Ethernet module receive empty water mark.
	PLIB_ETH_RxFullWmarkGet	Gets the Ethernet module receive full watermark.
	PLIB_ETH_RxFullWmarkSet	Sets the Ethernet module receive full water mark.
	PLIB_ETH_RxPauseDisable	Disables the EMAC receive flow control.
	PLIB_ETH_RxPauseEnable	Enables the EMAC receive flow control.
	PLIB_ETH_RxPauseIsEnabled	Gets the EMAC receive flow pause enable status.
	PLIB_ETH_ShortcutQuantaDisable	Disables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaEnable	Enables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaIsEnabled	Gets EMAC shortcut pause quanta enable status.
	PLIB_ETH_SimResetDisable	Disables the EMAC simulation reset.
	PLIB_ETH_SimResetEnable	Enables the EMAC simulation reset.
	PLIB_ETH_SimResetIsEnabled	Gets the EMAC simulation reset status.
	PLIB_ETH_TestBackPressDisable	Disables EMAC Test backpressure.
	PLIB_ETH_TestBackPressEnable	Enables EMAC Test backpressure.
	PLIB_ETH_TestBackPressIsEnabled	Gets the EMAC test backpressure enable status.
	PLIB_ETH_TxPauseDisable	Disables the transmission of Pause frames.
	PLIB_ETH_TxPauseEnable	Enables the the tansmission Pause frames.
	PLIB_ETH_TxPauseIsEnabled	Gets the Ethernet module enable status.

Initialization Functions

	Name	Description
	PLIB_ETH_AutoDetectPadClear	Clears the EMAC Auto-Pad option.
	PLIB_ETH_AutoDetectPadGet	Gets the current EMAC Auto-Pad setting.
	PLIB_ETH_AutoDetectPadSet	Sets the EMAC Auto-Pad option.
	PLIB_ETH_BackToBackIPGGet	Gets the EMAC back-to-back interpacket gap.

	PLIB_ETH_BackToBackIPGSet	Sets the EMAC back-to-back interpacket gap.
	PLIB_ETH_CollisionWindowGet	Gets the EMAC collision window.
	PLIB_ETH_CollisionWindowSet	Sets the EMAC collision window.
	PLIB_ETH_DelayedCRCDisable	Disables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCEnable	Enables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCIsEnabled	Gets the EMAC delayed CRC enable status.
	PLIB_ETH_Disable	Disables the Ethernet module.
	PLIB_ETH_Enable	Enables the Ethernet module.
	PLIB_ETH_ExcessDeferDisable	Disables EMAC excess defer.
	PLIB_ETH_ExcessDeferEnable	Enables EMAC excess defer.
	PLIB_ETH_ExcessDeferIsEnabled	Gets the EMAC excess defer enable status.
	PLIB_ETH_FrameLengthCheckDisable	Disables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckEnable	Enables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckIsEnabled	Gets the EMAC frame length check status.
	PLIB_ETH_FullDuplexDisable	Disables the EMAC full duplex operatio.
	PLIB_ETH_FullDuplexEnable	Enables the EMAC full deuplex operation.
	PLIB_ETH_FullDuplexIsEnabled	Gets the EMAC full duplex enable status.
	PLIB_ETH_HugeFrameDisable	Disables the EMAC from receiving huge frames.
	PLIB_ETH_HugeFrameEnable	Enables the EMAC to receive huge frames.
	PLIB_ETH_HugeFrameIsEnabled	Gets the EMAC huge frame enable status.
	PLIB_ETH_IsEnabled	Gets the Ethernet module enable status.
	PLIB_ETH_LongPreambleDisable	Disables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleEnable	Enables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleIsEnabled	Gets the EMAC long preamble enforcement enable status.
	PLIB_ETH_LoopbackDisable	Disables the EMAC loopback logic.
	PLIB_ETH_LoopbackEnable	Enables the EMAC loopback logic.
	PLIB_ETH_LoopbackIsEnabled	Gets the EMAC Loopback interface enable status.
	PLIB_ETH_MaxFrameLengthGet	Gets the EMAC maximum frame length.
	PLIB_ETH_MaxFrameLengthSet	Sets the EMAC maximum frame length.
	PLIB_ETH_MIIMClockGet	Gets the current EMAC MIIM clock selection.
	PLIB_ETH_MIIMClockSet	Sets the EMAC MIM clock selection.
	PLIB_ETH_MIIMNoPreDisable	Disables EMAC No Preamble (allows preamble).
	PLIB_ETH_MIIMNoPreEnable	Enables EMAC MIIM No Preamble (suppresses preamble).
	PLIB_ETH_MIIMNoPreIsEnabled	Gets the EMAC MIIM No Preamble enable status.
	PLIB_ETH_NonBackToBackIPG1Get	Gets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG1Set	Sets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG2Get	Gets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_NonBackToBackIPG2Set	Sets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_PauseTimerGet	Gets the Pause Timer value used for flow control.
	PLIB_ETH_PauseTimerSet	Sets the Pause Timer value used for flow control.
	PLIB_ETH_ReceiveBufferSizeGet	Gets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveBufferSizeSet	Sets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveDisable	Disables the EMAC from receiving frames.
	PLIB_ETH_ReceiveEnable	Enables the EMAC to receive the frames.
	PLIB_ETH_ReceiveIsEnabled	Gets the Ethernet EMAC receive enable status.

	PLIB_ETH_ReTxMaxGet	Gets the EMAC maximum retransmissions.
	PLIB_ETH_ReTxMaxSet	Sets the EMAC maximum retransmissions.
	PLIB_ETH_RMII SpeedGet	Gets the current EMAC RMII speed.
	PLIB_ETH_RMII SpeedSet	Sets EMAC RMII Speed.
	PLIB_ETH_StopInIdleDisable	Ethernet module operation will continue when the device enters Idle mode.
	PLIB_ETH_StopInIdleEnable	Ethernet module operation will stop when the device enters Idle mode.
	PLIB_ETH_StopInIdleIsEnabled	Gets the Ethernet module Stop in Idle mode enable status.

Interrupt Functions

	Name	Description
	PLIB_ETH_InterruptClear	Clears the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptSet	Sets the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptsGet	Gets the Ethernet module Interrupt Flag register as a group.
	PLIB_ETH_InterruptSourceDisable	Clears the Ethernet module Interrupt Enable register as a group, using a mask.
	PLIB_ETH_InterruptSourceEnable	Sets the Ethernet module Interrupt Enable register in a group, using a mask.
	PLIB_ETH_InterruptSourceIsEnabled	Gets the Ethernet module Interrupt Enable register singularly or as a group.
	PLIB_ETH_InterruptSourcesGet	Returns the entire interrupt enable register.
	PLIB_ETH_InterruptStatusGet	Gets the Ethernet module Interrupt Flag register as a group, using a mask.

Statistics Functions

	Name	Description
	PLIB_ETH_AlignErrorCountClear	Sets the count of Ethernet alignment errors to zero.
	PLIB_ETH_AlignErrorCountGet	Gets the count of Ethernet alignment errors.
	PLIB_ETH_FCSErrorCountClear	Sets the value of the Ethernet frame check sequence error to zero.
	PLIB_ETH_FCSErrorCountGet	Gets the count of the frame check sequence error.
	PLIB_ETH_FramesRxdOkCountClear	Sets the value of the Ethernet received frames 'OK' count to zero.
	PLIB_ETH_FramesRxdOkCountGet	Gets the count of the frames frames received successfully.
	PLIB_ETH_FramesTxdOkCountClear	Sets the count of Ethernet Control frames transmitted to zero.
	PLIB_ETH_FramesTxdOkCountGet	Gets the count of Ethernet Control Frames transmitted successfully.
	PLIB_ETH_MultipleCollisionCountClear	Sets the value of the Ethernet multiple collision frame count to zero.
	PLIB_ETH_MultipleCollisionCountGet	Gets the count of the frames transmitted successfully after there was more than one collision.
	PLIB_ETH_RxOverflowCountClear	Sets the value of the Ethernet receive overflow count to zero.
	PLIB_ETH_RxOverflowCountGet	Gets the count of the dropped Ethernet receive frames.
	PLIB_ETH_RxPacketCountGet	Gets the value of the receive packet buffer count.
	PLIB_ETH_SingleCollisionCountClear	Sets the value of the Ethernet single collision frame count to zero.
	PLIB_ETH_SingleCollisionCountGet	Gets the count of the frames transmitted successfully on a second attempt.

Status Functions

	Name	Description
	PLIB_ETH_DataNotValid	Gets the MII management read data not valid status.
	PLIB_ETH_EthernetIsBusy	Gets the status value of the Ethernet logic busy.
	PLIB_ETH_LinkHasFailed	Gets the MII management link fail status.
	PLIB_ETH_MIIMIsBusy	Gets the MII management busy status.
	PLIB_ETH_MIIMIsScanning	Gets the MII Management scanning status.

	PLIB_ETH_ReceiveIsBusy	Gets the the Ethernet receive logic busy status.
	PLIB_ETH_TransmitIsBusy	Gets the status value of the Ethernet transmit logic busy status

Description

This section describes the functions, data types, and constants provided in the Ethernet Peripheral Library. Refer to each section for detailed descriptions.

- Initialization Functions
- Status Functions
- Control Functions
- Filtering Functions
- Flow Control Functions
- Statistics Functions
- Interrupt Functions

5.6.8.5.1 Initialization Functions

5.6.8.5.1.1 PLIB_ETH_AutoDetectPadClear Function

C

```
void PLIB_ETH_AutoDetectPadClear(
    ETH_MODULE_ID index,
    ETH_AUTOPAD_OPTION option
);
```

Description

This function sets the EMAC Auto-Pad option. If any auto-pad option other than ETH_AUTOPAD_DISABLED_CHECK_CRC is selected, the CRC must be enabled by [PLIB_ETH_CRCEnable](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_AutoDetectPadClear(MY_ETH_INSTANCE, PAD_OPTION);
```

5.6.8.5.1.2 PLIB_ETH_AutoDetectPadGet Function

C

```
ETH_AUTOPAD_OPTION PLIB_ETH_AutoDetectPadGet(
    ETH_MODULE_ID index
);
```

Description

This function gets the current EMAC Auto-Pad setting.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

[ETH_AUTOPAD_OPTION](#)

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
autoPadOption = PLIB_ETH_AutoDetectPadGet(MY_ETH_INSTANCE);
```

5.6.8.5.1.3 PLIB_ETH_AutoDetectPadSet Function**C**

```
void PLIB_ETH_AutoDetectPadSet(  
    ETH_MODULE_ID index,  
    ETH_AUTOPAD_OPTION option  
);
```

Description

This function sets the EMAC Auto-Pad option. If any auto-pad option other than [ETH_AUTOPAD_DISABLED_CHECK_CRC](#) is selected, the CRC must be enabled by [PLIB_ETH_CRCEnable](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_AutoDetectPadSet(MY_ETH_INSTANCE, PAD_OPTION);
```

5.6.8.5.1.4 PLIB_ETH_BackToBackIPGGet Function**C**

```
uint8_t PLIB_ETH_BackToBackIPGGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC back-to-back interpacket gap.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

backToBackIPGValue - Back-to-back inter-packet gap (7-bits)

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_BackToBackIPGGet(MY_ETH_INSTANCE);
```

5.6.8.5.1.5 PLIB_ETH_BackToBackIPGSet Function**C**

```
void PLIB_ETH_BackToBackIPGSet(  
    ETH_MODULE_ID index,  
    uint8_t backToBackIPGValue  
);
```

Description

This function sets the EMAC back-to-back interpacket gap.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
backToBackIPGValue	Back-to-back inter-packet gap

Returns

None.

Remarks

In Full-Duplex mode, the register value should be the desired period in nibble times minus 3. In Full-Duplex, the recommended setting is 0x15.

In Half-Duplex mode, the register value should be the desired period in nibble times minus 6. In Half-Duplex, the recommended setting is 0x12.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_BackToBackIPGSet(MY_ETH_INSTANCE, backToBackIPGValue);
```

5.6.8.5.1.6 PLIB_ETH_CollisionWindowGet Function

C

```
uint8_t PLIB_ETH_CollisionWindowGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC collision window.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

collisionWindowValue - A uint8_t (6-bit) value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_CollisionWindowGet(MY_ETH_INSTANCE);
```

5.6.8.5.1.7 PLIB_ETH_CollisionWindowSet Function

C

```
void PLIB_ETH_CollisionWindowSet(  
    ETH_MODULE_ID index,  
    uint8_t collisionWindowValue  
);
```

Description

This function sets the EMAC collision window.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
collisionWindowValue	Collision window

Returns

None.

Remarks

This is a programmable field representing the slot time or collision window during which collisions occur in properly configured networks. Since the collision window starts at the beginning of transmission, the preamble and SFD is included. Its default of 0x37 (55d) corresponds to the count of frame bytes at the end of the window.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_CollisionWindowSet(MY_ETH_INSTANCE, collisionWindowValue);
```

5.6.8.5.1.8 PLIB_ETH_DelayedCRCDisable Function**C**

```
void PLIB_ETH_DelayedCRCDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC delayed CRC. No proprietary header exists.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_DelayedCRCDisable(MY_ETH_INSTANCE);
```

5.6.8.5.1.9 PLIB_ETH_DelayedCRCEnable Function**C**

```
void PLIB_ETH_DelayedCRCEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC delayed CRC. A proprietary packet header exists. Four bytes of the packet header are ignored by the CRC function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_DelayedCRCEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.10 PLIB_ETH_DelayedCRCIsEnabled Function

C

```
bool PLIB_ETH_DelayedCRCIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the EMAC delayed CRC enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - A proprietary header exists. Four bytes of packet header are ignored by the CRC function.
- false - No proprietary header

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_DelayedCRCIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.1.11 PLIB_ETH_Disable Function

C

```
void PLIB_ETH_Disable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the Ethernet module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_Disable(MY_ETH_INSTANCE);
```

5.6.8.5.1.12 PLIB_ETH_Enable Function

C

```
void PLIB_ETH_Enable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the Ethernet module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_Enable(MY_ETH_INSTANCE);
```

5.6.8.5.1.13 PLIB_ETH_ExcessDeferDisable Function

C

```
void PLIB_ETH_ExcessDeferDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC excess defer. The EMAC will abort when the excessive deferral limit is reached.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ExcessDeferDisable(MY_ETH_INSTANCE);
```

5.6.8.5.1.14 PLIB_ETH_ExcessDeferEnable Function

C

```
void PLIB_ETH_ExcessDeferEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC excess defer. The EMAC will defer to the carrier indefinitely.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ExcessDeferEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.15 PLIB_ETH_ExcessDeferIsEnabled Function

C

```
bool PLIB_ETH_ExcessDeferIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC excess defer enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC will defer to carrier indefinitely as per the Standard
- false - The EMAC will abort when the excessive deferral limit is reached

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_ExcessDeferIsEnabled(MY_ETH_INSTANCE);
```


5.6.8.5.1.16 PLIB_ETH_FrameLengthCheckDisable Function

C

```
void PLIB_ETH_FrameLengthCheckDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC frame length check.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_FrameLengthCheckDisable(MY_ETH_INSTANCE);
```

5.6.8.5.1.17 PLIB_ETH_FrameLengthCheckEnable Function

C

```
void PLIB_ETH_FrameLengthCheckEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC frame length check. Both transmit and receive frame lengths are compared to the Length/Type field. If the Length/Type field represents a length, the check is performed. Mismatches are reported on the transmit/receive statistics vector.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_FrameLengthCheckEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.18 PLIB_ETH_FrameLengthCheckIsEnabled Function

C

```
bool PLIB_ETH_FrameLengthCheckIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the EMAC frame length check status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Both transmit and receive frame lengths are compared to the Length/Type field. If the Length/Type field represents a length then the check is performed. Mismatches are reported on the transmit/receive statistics vector.
- false - Length/Type field check is not performed

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_FrameLengthCheckIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.1.19 PLIB_ETH_FullDuplexDisable Function

C

```
void PLIB_ETH_FullDuplexDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC full-duplex operation, which results in the EMAC operating in half-duplex.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_FullDuplexDisable(MY_ETH_INSTANCE);
```

5.6.8.5.1.20 PLIB_ETH_FullDuplexEnable Function

C

```
void PLIB_ETH_FullDuplexEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC full duplex operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_FullDuplexEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.21 PLIB_ETH_FullDuplexIsEnabled Function

C

```
bool PLIB_ETH_FullDuplexIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the EMAC full duplex enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC operates in Full-Duplex mode
- false - The EMAC operates in Half-Duplex mode

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_FullDuplexIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.1.22 PLIB_ETH_HugeFrameDisable Function

C

```
void PLIB_ETH_HugeFrameDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC from receiving huge frames.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_HugeFrameDisable(MY_ETH_INSTANCE);
```

5.6.8.5.1.23 PLIB_ETH_HugeFrameEnable Function

C

```
void PLIB_ETH_HugeFrameEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC receiving the frames of any length.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_HugeFrameEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.24 PLIB_ETH_HugeFrameIsEnabled Function

C

```
bool PLIB_ETH_HugeFrameIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the EMAC huge frame enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Frames of any length are transmitted and received
- false - Huge frames are not allowed for receive or transmit

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_HugeFrameIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.1.25 PLIB_ETH_IsEnabled Function

C

```
bool PLIB_ETH_IsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the Ethernet module enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Ethernet module is enabled
- false - Ethernet module is disabled

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_IsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.1.26 PLIB_ETH_LongPreambleDisable Function

C

```
void PLIB_ETH_LongPreambleDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC long preamble enforcement. The EMAC allows any length preamble.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_LongPreambleDisable(MY_ETH_INSTANCE);
```

5.6.8.5.1.27 PLIB_ETH_LongPreambleEnable Function

C

```
void PLIB_ETH_LongPreambleEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC long preamble enforcement. The EMAC only allows receive packets which contain preamble fields less than 12 bytes in length.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_LongPreambleEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.28 PLIB_ETH_LongPreambleIsEnabled Function

C

```
bool PLIB_ETH_LongPreambleIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC long preamble enforcement enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC only allows receive packets, which contain preamble fields less than 12 bytes in length
- false - The EMAC allows any length preamble

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_LongPreambleIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.1.29 PLIB_ETH_LoopbackDisable Function

C

```
void PLIB_ETH_LoopbackDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC loopback. EMAC resumes normal operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_LoopbackDisable(MY_ETH_INSTANCE);
```

5.6.8.5.1.30 PLIB_ETH_LoopbackEnable Function

C

```
void PLIB_ETH_LoopbackEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC loopback logic. The EMAC transmit interface is looped back to the EMAC receive interface.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_LoopbackEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.31 PLIB_ETH_LoopbackIsEnabled Function

C

```
bool PLIB_ETH_LoopbackIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC Loopback interface enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - EMAC transmit interface is looped back to the EMAC receive interface
- false - EMAC normal operation

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_LoopbackIsEnabled(MY_ETH_INSTANCE);
```


5.6.8.5.1.32 PLIB_ETH_MaxFrameLengthGet Function

C

```
uint16_t PLIB_ETH_MaxFrameLengthGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC maximum frame length.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

MaxFrameLength - Maximum frame length

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_MaxFrameLengthGet(MY_ETH_INSTANCE);
```

5.6.8.5.1.33 PLIB_ETH_MaxFrameLengthSet Function

C

```
void PLIB_ETH_MaxFrameLengthSet(  
    ETH_MODULE_ID index,  
    uint16_t MaxFrameLength  
);
```

Description

This function sets the EMAC maximum frame length.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
MaxFrameLength	Maximum frame length

Returns

None.

Remarks

This field resets to 0x05EE, which represents a maximum receive frame of 1518 bytes. An untagged maximum size Ethernet frame is 1518 bytes. A tagged frame adds four octets for a total of 1522 bytes. If a shorter/longer maximum length restriction is desired, program this 16-bit field.

If a proprietary header is allowed, this field should be adjusted accordingly. For example, if 4-byte headers are prepended to

frames, MACMAXF could be set to 1527 bytes. This would allow the maximum VLAN tagged frame plus the 4-byte header.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MaxFrameLengthSet(MY_ETH_INSTANCE, MaxFrameLength);
```

5.6.8.5.1.34 PLIB_ETH_MIIMClockGet Function**C**

```
ETH_MIIM_CLK PLIB_ETH_MIIMClockGet(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the current EMAC MIIM clock selection.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

MIIMClock - of type PLIB_ETH_MIIM_CLK

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
clkstat = PLIB_ETH_MIIMClockGet(MY_ETH_INSTANCE);
```

5.6.8.5.1.35 PLIB_ETH_MIIMClockSet Function**C**

```
void PLIB_ETH_MIIMClockSet(  
    ETH_MODULE_ID index,  
    ETH_MIIM_CLK MIIMClock  
);
```

Description

This function sets the EMAC MIIM clock selection.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
MIIMClock	of type ETH_MIIM_CLK - the system clock divisor for MII

Returns

None.

Remarks

This field is used by the clock divide logic in creating the MII Management Clock (MDC), which the IEEE 802.3 Specification defines to be no faster than 2.5 MHz. Some PHYs support clock rates up to 12.5 MHz.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMClockSet(MY_ETH_INSTANCE, MIIMClock);
```

5.6.8.5.1.36 PLIB_ETH_MIIMNoPreDisable Function**C**

```
void PLIB_ETH_MIIMNoPreDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC No preamble (allows preamble). Normal read/write cycles are performed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMNoPreDisable(MY_ETH_INSTANCE);
```

5.6.8.5.1.37 PLIB_ETH_MIIMNoPreEnable Function**C**

```
void PLIB_ETH_MIIMNoPreEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC MIIM No Preamble (suppresses preamble). The MII Management will perform read/write cycles without the 32-bit preamble field. Some PHYs support suppressed preamble.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMNoPreEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.38 PLIB_ETH_MIIMNoPrelsEnabled Function**C**

```
bool PLIB_ETH_MIIMNoPreIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC MIIM No Preamble enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MII Management will perform read/write cycles without the 32-bit preamble field. Some PHYs support suppressed preamble.
- false - Normal read/write cycles are performed

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_MIIMNoPreIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.1.39 PLIB_ETH_NonBackToBackIPG1Get Function**C**

```
uint8_t PLIB_ETH_NonBackToBackIPG1Get(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC non-back-to-back interpacket gap register 1.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

nonBackToBackIPGValue - Non-back-to-back inter-packet gap (7-bits)

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_NonBackToBackIPG1Get(MY_ETH_INSTANCE);
```

5.6.8.5.1.40 PLIB_ETH_NonBackToBackIPG1Set Function**C**

```
void PLIB_ETH_NonBackToBackIPG1Set(  
    ETH_MODULE_ID index,  
    uint8_t nonBackToBackIPGValue  
);
```

Description

This function sets the EMAC non-back-to-back interpacket gap register 1. A value of 0x0C is recommended.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
nonBackToBackIPGValue	Non-back-to-back inter-packet gap

Returns

None.

Remarks

This is a programmable field representing the optional carrierSense window referenced in the IEEE 802.3 Specification. If a carrier is detected during the timing of IPGR1, the MAC defers to the carrier. If, however, the carrier comes after IPGR1, the MAC continues timing IPGR2 and transmits, knowingly causing a collision, thereby ensuring fair access to the medium. Its range of values is 0x0 to IPGR2. Its recommended value is 0xC (12d).

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_NonBackToBackIPG1Set(MY_ETH_INSTANCE, nonBackToBackIPGValue);
```

5.6.8.5.1.41 PLIB_ETH_NonBackToBackIPG2Get Function**C**

```
uint8_t PLIB_ETH_NonBackToBackIPG2Get(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC non-back-to-back interpacket gap register 2.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

nonBackToBackIPGValue - Non-back-to-back inter-packet gap (7-bits)

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_NonBackToBackIPG2Get(MY_ETH_INSTANCE);
```

5.6.8.5.1.42 PLIB_ETH_NonBackToBackIPG2Set Function**C**

```
void PLIB_ETH_NonBackToBackIPG2Set(  
    ETH_MODULE_ID index,  
    uint8_t nonBackToBackIPGValue  
);
```

Description

This function sets the EMAC non-back-to-back interpacket gap register 2. A value of 0x12 is recommended.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
nonBackToBackIPGValue	Non-back-to-back inter-packet gap

Returns

None.

Remarks

This is a programmable field representing the non-back-to-back Inter-Packet-Gap. Its recommended value is 0x12 (18d), which represents the minimum IPG of 0.96 us (in 100 Mbps) or 9.6 us (in 10 Mbps).

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_NonBackToBackIPG2Set(MY_ETH_INSTANCE, nonBackToBackIPGValue);
```

5.6.8.5.1.43 PLIB_ETH_PauseTimerGet Function**C**

```
uint16_t PLIB_ETH_PauseTimerGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the Pause Timer value used for flow control.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

PauseTimerValue - Pause Timer Value

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PauseTimerValue = PLIB_ETH_PauseTimerGet(MY_ETH_INSTANCE);
```

5.6.8.5.1.44 PLIB_ETH_PauseTimerSet Function**C**

```
void PLIB_ETH_PauseTimerSet(  
    ETH_MODULE_ID index,  
    uint16_t PauseTimerValue  
);
```

Description

This function sets the Pause Timer value used for flow control.

Preconditions

Write to the Pause Timer register before enabling the receiver. Call this function before calling [PLIB_ETH_ReceiveEnable\(\)](#)

Parameters

Parameters	Description
index	Identifier for the device instance
PauseTimerValue	used for Flow Control

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_PauseTimerSet(MY_ETH_INSTANCE, PauseTimerValue);
```

5.6.8.5.1.45 PLIB_ETH_ReceiveBufferSizeGet Function**C**

```
uint8_t PLIB_ETH_ReceiveBufferSizeGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the Ethernet receive buffer size. The buffer size is set from 16 bytes up to 2032 bytes in increments of 16 bytes each.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

ReceiveBufferSize - receive buffer size divided by 16

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
size = PLIB_ETH_ReceiveBufferSizeGet(MY_ETH_INSTANCE);
```

5.6.8.5.1.46 PLIB_ETH_ReceiveBufferSizeSet Function**C**

```
void PLIB_ETH_ReceiveBufferSizeSet(  
    ETH_MODULE_ID index,  
    uint8_t ReceiveBufferSize  
);
```

Description

This function sets the Ethernet receive buffer size. The buffer size is set from 16 bytes up to 2032 bytes in increments of 16 bytes each.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ReceiveBufferSize	In units of 16 bytes, must be a value of 0x01 to 0x7F, 0x00 is invalid

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ReceiveBufferSizeSet(MY_ETH_INSTANCE, ReceiveBufferSize/16 );
```

5.6.8.5.1.47 PLIB_ETH_ReceiveDisable Function**C**

```
void PLIB_ETH_ReceiveDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC from receiving frames.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ReceiveDisable(MY_ETH_INSTANCE);
```

5.6.8.5.1.48 PLIB_ETH_ReceiveEnable Function**C**

```
void PLIB_ETH_ReceiveEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC to receive the frames.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ReceiveEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.49 PLIB_ETH_ReceiveIsEnabled Function**C**

```
bool PLIB_ETH_ReceiveIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the Ethernet EMAC receive enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Enable the EMAC to receive frames
- false - Disable the EMAC to receive frames

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_ReceiveIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.1.50 PLIB_ETH_ReTxMaxGet Function**C**

```
uint8_t PLIB_ETH_ReTxMaxGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC maximum retransmissions.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

retransmitMax - Maximum number of retransmissions

Remarks

The maximum number of attempts is limited to 0x0F.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_ReTxMaxGet(MY_ETH_INSTANCE);
```

5.6.8.5.1.51 PLIB_ETH_ReTxMaxSet Function**C**

```
void PLIB_ETH_ReTxMaxSet(  
    ETH_MODULE_ID index,  
    uint16_t retransmitMax  
);
```

Description

This function sets the EMAC maximum retransmissions.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
retransmitMax	Maximum number of retransmissions

Returns

None.

Remarks

This is a programmable field specifying the number of retransmission attempts following a collision before aborting the packet due to excessive collisions. The IEEE 802.3 Specification standard specifies the maximum number of attempts (attemptLimit) to be 0xF (15d). Its default is 000.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ReTxMaxSet(retransmitMax);
```

5.6.8.5.1.52 PLIB_ETH_RMII SpeedGet Function**C**

```
ETH_RMII_SPEED PLIB_ETH_RMII SpeedGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the current EMAC RMII speed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

RMII_100Mbps - RMII running at 100Mbps RMII_10Mbps - RMII running at 10Mbps

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RMII SpeedGet(MY_ETH_INSTANCE);
```

5.6.8.5.1.53 PLIB_ETH_RMII SpeedSet Function**C**

```
void PLIB_ETH_RMII SpeedSet(  
    ETH_MODULE_ID index,  
    ETH_RMII_SPEED RMII Speed  
);
```

Description

This function sets EMAC RMII speed. RMII speed can be either RMII_100Mbps or RMII_10Mbps.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
RMII Speed	RMII_100Mbps or RMII_10Mbps of type ETH_RMII_SPEED

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RMII SpeedSet (MY_ETH_INSTANCE, ETH_RMII_SPEED RMII Speed);
```

5.6.8.5.1.54 PLIB_ETH_StopInIdleDisable Function**C**

```
void PLIB_ETH_StopInIdleDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function directs the Ethernet module to continue operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_StopInIdleDisable (MY_ETH_INSTANCE);
```

5.6.8.5.1.55 PLIB_ETH_StopInIdleEnable Function**C**

```
void PLIB_ETH_StopInIdleEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function directs the Ethernet module to stop operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_StopInIdleEnable(MY_ETH_INSTANCE);
```

5.6.8.5.1.56 PLIB_ETH_StopInIdleIsEnabled Function

C

```
bool PLIB_ETH_StopInIdleIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the Ethernet module Stop in Idle mode enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Ethernet module transfers stop during Idle mode
- false - Ethernet module transfers continue to run during Idle mode

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_StopInIdleIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2 Control Functions

5.6.8.5.2.1 PLIB_ETH_MCSRxResetDisable Function

C

```
void PLIB_ETH_MCSRxResetDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the reset EMAC Control Sub-layer/Receive domain logic. The MCS/Receive domain logic is released from reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MCSRxResetDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.2 PLIB_ETH_MCSRxResetEnable Function**C**

```
void PLIB_ETH_MCSRxResetEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the reset EMAC Control Sub-layer/Receive domain logic. The MCS/Receive domain logic is held in reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MCSRxResetEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.3 PLIB_ETH_MCSRxResetIsEnabled Function**C**

```
bool PLIB_ETH_MCSRxResetIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC Control Sub-layer/Receive domain logic reset status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - EMAC Control Sub-layer/Receive domain logic is in reset
- false - EMAC Control Sub-layer/Receive domain logic is not in reset

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_MCSRxResetIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.4 PLIB_ETH_MCSTxResetDisable Function**C**

```
void PLIB_ETH_MCSTxResetDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the reset EMAC Control Sub-layer/Transmit domain logic. The MCS/Transmit domain logic is released from reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MCSTxResetDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.5 PLIB_ETH_MCSTxResetEnable Function**C**

```
void PLIB_ETH_MCSTxResetEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function resets the EMAC Control Sub-layer/Transmit domain logic. The MCS/Transmit domain logic is held in reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MCSTxResetEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.6 PLIB_ETH_MCSTxResetIsEnabled Function**C**

```
bool PLIB_ETH_MCSTxResetIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC Control Sub-layer/Transmit domain logic reset status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - EMAC Control Sub-layer/Transmit domain logic is held in reset
- false - EMAC Control Sub-layer/Transmit domain logic is not in reset

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_MCSTxResetIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.7 PLIB_ETH_MIIMReadDataGet Function**C**

```
uint16_t PLIB_ETH_MIIMReadDataGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets EMAC MIIM management read data after a MII read cycle has completed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

readData - MII read data

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
readData = PLIB_ETH_MIIMReadDataGet(MY_ETH_INSTANCE);
```

5.6.8.5.2.8 PLIB_ETH_MIIMReadStart Function**C**

```
void PLIB_ETH_MIIMReadStart(  
    ETH_MODULE_ID index  
);
```

Description

This function initiates an MII read command. The MII Management module will perform a single read cycle. To get data, use PLIB_ETH_MIIReadDataGet.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
while (PLIB_ETH_MIIMIsBusy(MY_ETH_INSTANCE))  
{  
    //Wait until MII Link is not busy  
}  
PLIB_ETH_MIIReadStart(MY_ETH_INSTANCE);  
while (PLIB_ETH_MIIReadIsDataNotValid(MY_ETH_INSTANCE))  
{  
    //Wait until read is complete  
}  
data = PLIB_ETH_MIIReadDataGet(MY_ETH_INSTANCE);
```

5.6.8.5.2.9 PLIB_ETH_MIIMResetDisable Function**C**

```
void PLIB_ETH_MIIMResetDisable(  

```

```
    ETH_MODULE_ID index
);
```

Description

This function disables EMAC Reset MII Management. EMAC will resume normal operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMResetDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.10 PLIB_ETH_MIIMResetEnable Function**C**

```
void PLIB_ETH_MIIMResetEnable(
    ETH_MODULE_ID index
);
```

Description

This function enables EMAC Reset MII Management and holds the MII Management module in reset while enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

MII Management held in Reset after This function is called. Disable ResetMIIanagement to return to normal operation.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMResetEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.11 PLIB_ETH_MIIMResetIsEnabled Function**C**

```
bool PLIB_ETH_MIIMResetIsEnabled(
    ETH_MODULE_ID index
);
```

Description

This function gets the EMAC Reset MII Management enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Reset the MII Management module
- false - Normal operation

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_MIIMResetIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.12 PLIB_ETH_MIIMScanIncrEnable Function**C**

```
void PLIB_ETH_MIIMScanIncrEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC MIIM Scan Increment. The MII Management module will perform read cycles across a range of PHYs. The read cycles will start from address 1 through the value set in the PHY address register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

The read cycles will start at PHY address 1 and continue through the value set for as the PHY address register.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMScanIncrEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.13 PLIB_ETH_MIIMScanIncrDisable Function**C**

```
void PLIB_ETH_MIIMScanIncrDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC MIIM Scan Increment. Allows continuous reads of the same PHY.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMScanIncrDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.14 PLIB_ETH_MIIMScanIncrIsEnabled Function**C**

```
bool PLIB_ETH_MIIMScanIncrIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC MIIM scan increment enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MII Management module will perform read cycles across a range of PHYs. The read cycles will start from address 1 through the value set in the PHY address register.
- false - Continuous reads of the same PHY

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
ScanIncrement = PLIB_ETH_MIIMScanIncrIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.15 PLIB_ETH_MIIMScanModeDisable Function**C**

```
void PLIB_ETH_MIIMScanModeDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables MIIM scan mode. Scan is disabled for Normal operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMScanModeDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.16 PLIB_ETH_MIIMScanModeEnable Function**C**

```
void PLIB_ETH_MIIMScanModeEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables MIIM scan mode. The MII Management module will perform read cycles continuously. (Useful for monitoring the Link Fail.)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMScanModeEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.17 PLIB_ETH_MIIMScanModeIsEnabled Function**C**

```
bool PLIB_ETH_MIIMScanModeIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the MII management scan enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MII Management module will perform read cycles continuously (for example, useful for monitoring the Link Fail)
- false - Normal operation

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_MIIMScanModeIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.18 PLIB_ETH_MIIMWriteDataSet Function**C**

```
void PLIB_ETH_MIIMWriteDataSet(  
    ETH_MODULE_ID index,  
    uint16_t writeData  
);
```

Description

This function sets the EMAC MIIM write data before initiating write cycle.

Preconditions

Prior to a call to this routine, the PHY and Register addresses should be set using PLIB_ETH_MIIPHYAddressSet and PLIB_ETH_MIIMRegisterAddressSet.

Parameters

Parameters	Description
index	Identifier for the device instance
writeData	MII write data

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIMWriteDataSet(MY_ETH_INSTANCE, WriteData);
```

5.6.8.5.2.19 PLIB_ETH_MIIMWriteStart Function**C**

```
void PLIB_ETH_MIIMWriteStart(  
    ETH_MODULE_ID index  
);
```

Description

This function initiates an MII management read command. The MII Management module will perform a write cycle.

Preconditions

The PHY address and MII register address must be configured before a write using `PLIB_ETH_MIIPHYAddressSet(MY_ETH_INSTANCE, phyAddr)` and `PLIB_ETH_MIIRegisterAddressSet(MY_ETH_INSTANCE, regAddr)`

Data to be written must be first loaded into the MII write register using `PLIB_ETH_MIIWriteDataSet(MY_ETH_INSTANCE, writeData)`

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIWriteDataSet(MY_ETH_INSTANCE, dataToWrite);  
PLIB_ETH_MIIWriteStart(MY_ETH_INSTANCE);
```

5.6.8.5.2.20 PLIB_ETH_MIIResetDisable Function**C**

```
void PLIB_ETH_MIIResetDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC MIIM Soft reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIResetDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.21 PLIB_ETH_MIIResetEnable Function

C

```
void PLIB_ETH_MIIResetEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC MIIM soft reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MIIResetEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.22 PLIB_ETH_MIIResetIsEnabled Function

C

```
bool PLIB_ETH_MIIResetIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets EMAC MIIM Soft reset enable status. By default this bit is set to '1'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - MAC MII is in reset
- false - MAC MII is not in reset

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_MIIResetIsEnabled(MY_ETH_INSTANCE);
```


5.6.8.5.2.23 PLIB_ETH_PHYAddressGet Function

C

```
uint8_t PLIB_ETH_PHYAddressGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC MIIM management PHY address.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

phyAddr - A 5-bit address of the PHY

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
phyAddr = PLIB_ETH_PHYAddressGet(MY_ETH_INSTANCE);
```

5.6.8.5.2.24 PLIB_ETH_PHYAddressSet Function

C

```
void PLIB_ETH_PHYAddressSet(  
    ETH_MODULE_ID index,  
    uint8_t phyAddr  
);
```

Description

This function sets the EMAC MIIM PHY address. This field represents the 5-bit PHY Address field of Management cycles. Up to 31 PHYs can be addressed (0 is reserved).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
phyAddr	A 5-bit address of the PHY

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_PHYAddressSet(MY_ETH_INSTANCE, PhyAddr);
```

5.6.8.5.2.25 PLIB_ETH_RegisterAddressGet Function

C

```
uint8_t PLIB_ETH_RegisterAddressGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC MIIM management register address.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

regAddr - The (5-bit) address of the MII Registers

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
regAddr = PLIB_ETH_RegisterAddressGet(MY_ETH_INSTANCE);
```

5.6.8.5.2.26 PLIB_ETH_RegisterAddressSet Function

C

```
void PLIB_ETH_RegisterAddressSet(  
    ETH_MODULE_ID index,  
    uint8_t regAddr  
);
```

Description

This function sets the EMAC MIIM register address. Up to 32 registers may be accessed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
regAddr	The (5-bit) address of the MII Registers.

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RegisterAddressSet(MY_ETH_INSTANCE, regAddr);
```

5.6.8.5.2.27 PLIB_ETH_RMIIResetDisable Function

C

```
void PLIB_ETH_RMIIResetDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC Reset RMII for normal operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RMIIResetDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.28 PLIB_ETH_RMIIResetEnable Function

C

```
void PLIB_ETH_RMIIResetEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC Reset RMII.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RMIIResetEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.29 PLIB_ETH_RMIIResetIsEnabled Function

C

```
bool PLIB_ETH_RMIIResetIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC Reset RMII enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Reset the EMAC RMII module
- false - Normal operation

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_RMIIResetIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.30 PLIB_ETH_RxBufferCountDecrement Function

C

```
void PLIB_ETH_RxBufferCountDecrement(  
    ETH_MODULE_ID index  
);
```

Description

This function causes the Receive Descriptor Buffer Counter to decrement by 1.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

Hardware increments the receive buffer counter and software decrements it. If the receive buffer counter is incremented by the receive logic at the same time, the receive buffer counter will appear unchanged.

Always reads as '0', so there is no get value routine.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxBufferCountDecrement(MY_ETH_INSTANCE);
```

5.6.8.5.2.31 PLIB_ETH_RxDisable Function**C**

```
void PLIB_ETH_RxDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the Ethernet receive logic.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

Disabling Ethernet receive is not recommended for making changes to any receive-related registers. After the receiver has been enabled, the Ethernet module must be reinitialized to implement changes.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.32 PLIB_ETH_RxEnable Function**C**

```
void PLIB_ETH_RxEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the Ethernet receive logic. Packets are received and stored in the receive buffer as controlled by the filter configuration.

Preconditions

All receive registers must be configured before calling this function.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.33 PLIB_ETH_RxFuncResetDisable Function**C**

```
void PLIB_ETH_RxFuncResetDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC reset receive function logic. The reset receive function logic is released from reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxFuncResetDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.34 PLIB_ETH_RxFuncResetEnable Function**C**

```
void PLIB_ETH_RxFuncResetEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC reset receive function logic. The receive function logic is held in reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxFuncResetEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.35 PLIB_ETH_RxFuncResetIsEnabled Function

C

```
bool PLIB_ETH_RxFuncResetIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC reset receive function status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - EMAC Receive function logic is held in reset
- false - EMAC Receive function logic is not in reset

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_RxFuncResetIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.36 PLIB_ETH_RxIsEnabled Function

C

```
bool PLIB_ETH_RxIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the Ethernet module receive enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Ethernet module receive is enabled
- false - Ethernet module receive is disabled

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_ReceiveIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.37 PLIB_ETH_RxPacketDescAddrGet Function

C

```
uint8_t * PLIB_ETH_RxPacketDescAddrGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the address of the next receive descriptor to be used.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

ReceivePacketDescriptorAddress - receive packet descriptor address

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
RxPacketDescAddr = PLIB_ETH_RxPacketDescAddrGet(MY_ETH_INSTANCE);
```

5.6.8.5.2.38 PLIB_ETH_RxPacketDescAddrSet Function

C

```
void PLIB_ETH_RxPacketDescAddrSet(  
    ETH_MODULE_ID index,  
    uint8_t * rxPacketDescStartAddr  
);
```

Description

This function sets the Ethernet receive packet descriptor start address.

Preconditions

No transmit, receive, or DMA operations should be in progress when this function is called. Call this function before enabling transmit and receive.

Parameters

Parameters	Description
index	Identifier for the device instance
rxPacketDescStartAddr	This address must be 4-byte aligned. (The least significant 2 bits must be '00'.)

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxPacketDescAddrSet(MY_ETH_INSTANCE, rxPacketDescStartAddr)
```


5.6.8.5.2.39 PLIB_ETH_TestPauseDisable Function

C

```
void PLIB_ETH_TestPauseDisable(  
    ETH_MODULE_ID index  
)  
;
```

Description

This function disables EMAC Test Pause. The EMAC will resume normal operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TestPauseDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.40 PLIB_ETH_TestPauseEnable Function

C

```
void PLIB_ETH_TestPauseEnable(  
    ETH_MODULE_ID index  
)  
;
```

Description

This function enables EMAC test pause. The EMAC Control sub-layer will inhibit transmissions, just as if a Pause Receive Control frame with a non-zero pause time parameter was received.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TestPauseEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.41 PLIB_ETH_TestPauseIsEnabled Function**C**

```
bool PLIB_ETH_TestPauseIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the EMAC test pause enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC Control sub-layer will inhibit transmissions, just as if a Pause Receive Control frame with a non-zero pause time parameter was received
- false - Normal operation

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_TestPauseIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.42 PLIB_ETH_TxFuncResetDisable Function**C**

```
void PLIB_ETH_TxFuncResetDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC Transmit function reset. The EMAC transmit function is released from reset for normal operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TxFuncResetDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.43 PLIB_ETH_TxFuncResetEnable Function**C**

```
void PLIB_ETH_TxFuncResetEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC transmit function reset. The transmit function is held in reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TxFuncResetEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.44 PLIB_ETH_TxFuncResetIsEnabled Function**C**

```
bool PLIB_ETH_TxFuncResetIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC Transmit function reset status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - EMAC transmit function logic is held in reset
- false - EMAC transmit function logic is not in reset

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TxFuncResetIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.45 PLIB_ETH_TxPacketDescAddrGet Function**C**

```
uint8_t * PLIB_ETH_TxPacketDescAddrGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the address of the next transmit descriptor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Transmit Packet Descriptor Start Address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
txPacketDescAddr = PLIB_ETH_TxPacketDescAddrGet(MY_ETH_INSTANCE);
```

5.6.8.5.2.46 PLIB_ETH_TxPacketDescAddrSet Function**C**

```
void PLIB_ETH_TxPacketDescAddrSet(  
    ETH_MODULE_ID index,  
    uint8_t * txPacketDescStartAddr  
);
```

Description

This function sets the Ethernet transmit packet descriptor start address.

Preconditions

No transmit, receive, or DMA operations should be in progress when this function is called. Call this function before enabling transmit and receive.

Parameters

Parameters	Description
index	Identifier for the device instance
txPacketDescStartAddr	This address must be 4-byte aligned. (The least significant 2 bits must be '00'.)

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TxPacketDescAddrSet(MY_ETH_INSTANCE, txPacketDescStartAddr)
```

5.6.8.5.2.47 PLIB_ETH_TxRTSDisable Function**C**

```
void PLIB_ETH_TxRTSDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function aborts an Ethernet module transmission and disables the transmitter after the current packet has completed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

When disabled by software, transmission stops after the current packet has been completed.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TxRTSDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.48 PLIB_ETH_TxRTSEnable Function**C**

```
void PLIB_ETH_TxRTSEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the Ethernet request to [send](#). Transmit logic is activated and packet(s) defined in the Ethernet descriptor table are transmitted.

Preconditions

The TX descriptor list and TX DMA must be initialized using PLIB_ETH_TransmitPacketDescStartAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This status is cleared by hardware when the transmission is complete.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TxRTSEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.49 PLIB_ETH_TxRTSIsEnabled Function**C**

```
bool PLIB_ETH_TxRTSIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the Ethernet module transmit request to [send](#) status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Ethernet module transmit is active
- false - Ethernet module transmission has stopped or has completed

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_TransmitRTSIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.2.50 PLIB_ETH_CRCDisable Function**C**

```
void PLIB_ETH_CRCDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC CRC.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

The frames presented to the EMAC have a valid CRC

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_CRCDisable(MY_ETH_INSTANCE);
```

5.6.8.5.2.51 PLIB_ETH_CRCEnable Function**C**

```
void PLIB_ETH_CRCEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC CRC. The EMAC will append CRC whether padding is required or not. This must be enabled if auto-padding is enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_CRCEnable(MY_ETH_INSTANCE);
```

5.6.8.5.2.52 PLIB_ETH_CRCIsEnabled Function**C**

```
bool PLIB_ETH_CRCIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the EMAC CRC enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC will append a CRC to every frame whether or not padding was required. Must be set if auto-padding is enabled.

- false - The frames presented to the EMAC have a valid CRC

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_CRCIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.3 Status Functions

5.6.8.5.3.1 PLIB_ETH_DataNotValid Function

C

```
bool PLIB_ETH_DataNotValid(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the MII management read data not valid status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MII Management read cycle has not completed and the read data is not yet valid
- false - The MII Management read cycle is complete and the read data is valid

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_DataNotValid(MY_ETH_INSTANCE);
```

Also see the example for function: PLIB_ETH_MIIReadStart().

5.6.8.5.3.2 PLIB_ETH_EthernetIsBusy Function

C

```
bool PLIB_ETH_EthernetIsBusy(  
    ETH_MODULE_ID index  
);
```

Description

This function sets the value of the Ethernet logic busy. A request indicates that the module has just been turned ON or is completing a transaction after being turned OFF.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Ethernet logic has been turned ON or is completing a transaction
- false - Ethernet logic is idle

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_EthernetIsBusy(MY_ETH_INSTANCE);
```

5.6.8.5.3.3 PLIB_ETH_LinkHasFailed Function**C**

```
bool PLIB_ETH_LinkHasFailed(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the MII management link fail status. This value reflects the last read from the PHY status register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MII Management module link fail has occurred
- false - The MII Management module link fail has not occurred

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_LinkHasFailed(MY_ETH_INSTANCE);
```

5.6.8.5.3.4 PLIB_ETH_MIIMIsBusy Function**C**

```
bool PLIB_ETH_MIIMIsBusy(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the MII management busy status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MII Management module is currently performing an MII Management read or write cycle
- false - The MII Management is free

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_MIIMIsBusy(MY_ETH_INSTANCE);
```

5.6.8.5.3.5 PLIB_ETH_MIIMIsScanning Function**C**

```
bool PLIB_ETH_MIIMIsScanning(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the MII Management scanning status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MII Management module scan operation (continuous MII Management Read cycles) is in progress
- false - The MII Management scan operation is not in progress

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_MIIMIsScanning(MY_ETH_INSTANCE);
```

5.6.8.5.3.6 PLIB_ETH_ReceiveIsBusy Function**C**

```
bool PLIB_ETH_ReceiveIsBusy(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the Ethernet receive logic busy status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Receive logic is receiving data
- false - Receive logic is idle

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_ReceiveIsBusy(MY_ETH_INSTANCE);
```

5.6.8.5.3.7 PLIB_ETH_TransmitIsBusy Function**C**

```
bool PLIB_ETH_TransmitIsBusy(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the value of the Ethernet transmit logic busy status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Transmit logic is sending data
- false - Transmit logic is idle

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_TransmitIsBusy(MY_ETH_INSTANCE);
```

5.6.8.5.4 Filtering Functions**5.6.8.5.4.1 PLIB_ETH_HashTableGet Function****C**

```
uint32_t PLIB_ETH_HashTableGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the value of the Hash table.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

hashTable - Hash table value (64-bits)

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_HashTableGet(MY_ETH_INSTANCE);
```

5.6.8.5.4.2 PLIB_ETH_HashTableSet Function**C**

```
void PLIB_ETH_HashTableSet(  
    ETH_MODULE_ID index,  
    uint64_t hashTableValue  
);
```

Description

This function sets the Hash table with the new value.

Preconditions

Call `PLIB_ETH_HashTableFilterDisable` to disable the Hash table filter before changing the hash table, or set the Hash Table prior to calling [PLIB_ETH_ReceiveEnable](#).

Parameters

Parameters	Description
index	Identifier for the device instance
hashTableValue	hash table value (64-bits)

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_HashTableSet(MY_ETH_INSTANCE, hashTableValue);
```

5.6.8.5.4.3 PLIB_ETH_PassAllDisable Function**C**

```
void PLIB_ETH_PassAllDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC PassAll. Control frames are ignored.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_PassAllDisable(MY_ETH_INSTANCE);
```

5.6.8.5.4.4 PLIB_ETH_PassAllEnable Function**C**

```
void PLIB_ETH_PassAllEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC PassAll, which enables both the normal and the control frames to be passed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_PassAllEnable(MY_ETH_INSTANCE);
```

5.6.8.5.4.5 PLIB_ETH_PassAllIsEnabled Function**C**

```
bool PLIB_ETH_PassAllIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC PassAll enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC will **accept** all frames regardless of type (normal vs. Control)
- false - The received Control frames are ignored

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_PassAllIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.4.6 PLIB_ETH_PatternMatchChecksumGet Function**C**

```
uint16_t PLIB_ETH_PatternMatchChecksumGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the value of the patter match checksum register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

The pattern match checksum.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_PatternMatchChecksumGet(MY_ETH_INSTANCE);
```

5.6.8.5.4.7 PLIB_ETH_PatternMatchChecksumSet Function**C**

```
void PLIB_ETH_PatternMatchChecksumSet(  
    ETH_MODULE_ID index,  
    uint16_t PatternMatchChecksumValue  
);
```

Description

This function sets the pattern match checksum register with the new value.

Preconditions

Call **PLIB_ETH_PatternMatchModeSet**(ETH_PATTERN_MATCH_DISABLED) to disable PatternMatch before changing the pattern match mask, or set the value before calling **PLIB_ETH_ReceiveEnable**.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
PatternMatchChecksumValue	Pattern match checksum value

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_PatternMatchChecksumSet(MY_ETH_INSTANCE, PatternMatchChecksumValue);
```

5.6.8.5.4.8 PLIB_ETH_PatternMatchGet Function**C**

```
uint64_t PLIB_ETH_PatternMatchGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the selected value of the patter match mask register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

patternMatchMaskValue - Pattern Match Mask Values

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_PatternMatchGet(MY_ETH_INSTANCE);
```

5.6.8.5.4.9 PLIB_ETH_PatternMatchModeGet Function**C**

```
ETH_PATTERN_MATCH_MODE PLIB_ETH_PatternMatchModeGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the selected value of the patter match mask register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

modeSel - The method of pattern matching from ETH_PATTERN_MATCH_DISABLED

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_PatternMatchModeGet(MY_ETH_INSTANCE);
```

5.6.8.5.4.10 PLIB_ETH_PatternMatchModeSet Function**C**

```
void PLIB_ETH_PatternMatchModeSet(  
    ETH_MODULE_ID index,  
    ETH_PATTERN_MATCH_MODE modeSel  
);
```

Description

This function sets the pattern match mode.

Preconditions

Set the value before calling [PLIB_ETH_ReceiveEnable\(\)](#).

Parameters

Parameters	Description
index	Identifier for the device instance
modeSel	The method of pattern matching from ETH_PATTERN_MATCH_DISABLED

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_PatternMatchModeSet(MY_ETH_INSTANCE, ETH_PATTERN_MATCH_DISABLED);
```

5.6.8.5.4.11 PLIB_ETH_PatternMatchOffsetGet Function**C**

```
uint16_t PLIB_ETH_PatternMatchOffsetGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the value of the patter match offset register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

PatternMatchOffsetValue - Pattern match offset value

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
value = PLIB_ETH_PatternMatchOffsetGet(MY_ETH_INSTANCE);
```

5.6.8.5.4.12 PLIB_ETH_PatternMatchOffsetSet Function**C**

```
void PLIB_ETH_PatternMatchOffsetSet(  
    ETH_MODULE_ID index,  
    uint16_t PatternMatchOffsetValue  
);
```

Description

This function sets the pattern match offset register with the new value.

Preconditions

Call [PLIB_ETH_PatternMatchModeSet](#)(ETH_PATTERN_MATCH_DISABLED) to disable PatternMatch before changing the pattern match mask, or set the value before calling [PLIB_ETH_ReceiveEnable](#)().

Parameters

Parameters	Description
index	Identifier for the device instance
PatternMatchOffsetValue	Pattern match offset value

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_PatternMatchOffsetSet(MY_ETH_INSTANCE, PatternMatchOffsetValue);
```

5.6.8.5.4.13 PLIB_ETH_PatternMatchSet Function**C**

```
void PLIB_ETH_PatternMatchSet(  
    ETH_MODULE_ID index,  
    uint64_t patternMatchMaskValue  
);
```

Description

This function sets the pattern match mask register with the new value.

Preconditions

Call [PLIB_ETH_PatternMatchModeSet](#)(index,ETH_PATTERN_MATCH_DISABLED) to disable PatternMatch before changing

the pattern match mask, or set the value before calling `PLIB_ETH_ReceiveEnable`.

Parameters

Parameters	Description
index	Identifier for the device instance
patternMatchMaskValue	Pattern Match Mask Values (64-bits)

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_PatternMatchSet(MY_ETH_INSTANCE, patternMatchMaskValue);
```

5.6.8.5.4.14 PLIB_ETH_PurePreambleDisable Function

C

```
void PLIB_ETH_PurePreambleDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC pure preamble enforcement. The EMAC does not perform any preamble checking.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Example

```
PLIB_ETH_PurePreambleDisable(MY_ETH_INSTANCE);
```

Remarks: This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

5.6.8.5.4.15 PLIB_ETH_PurePreambleEnable Function

C

```
void PLIB_ETH_PurePreambleEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC pure preamble enforcement. The EMAC will verify the contents of the preamble and discard packets with errors in the preamble.

sPrecondition: None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

The EMAC will verify the content of the preamble to ensure it contains 0x55 and is error-free. A packet with errors in its preamble is discarded.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_PurePreambleEnable(MY_ETH_INSTANCE);
```

5.6.8.5.4.16 PLIB_ETH_PurePreambleIsEnabled Function**C**

```
bool PLIB_ETH_PurePreambleIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC pure preamble enforcement enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC will verify the content of the preamble to ensure it contains 0x55 and is error-free. A packet with errors in its preamble is discarded.
- false - The EMAC does not perform any preamble checking

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_PurePreambleIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.4.17 PLIB_ETH_ReceiveFilterDisable Function**C**

```
void PLIB_ETH_ReceiveFilterDisable(  
    ETH_MODULE_ID index,  
    ETH_RECEIVE_FILTER filter  
);
```

Description

This function disables the specified receive filter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
filter	The selection of receive filters to be disabled from the enumerated selection ETH_RECEIVE_FILTER

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ReceiveFilterDisable(index,  
                              ETH_CRC_OK_FILTER    |  
                              ETH_RUNT_ENABLE_FILTER |  
                              ETH_UNICAST_FILTER    );
```

5.6.8.5.4.18 PLIB_ETH_ReceiveFilterEnable Function**C**

```
void PLIB_ETH_ReceiveFilterEnable(  
    ETH_MODULE_ID index,  
    ETH_RECEIVE_FILTER filter  
);
```

Description

This function enables the specified receive filter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
filter	The selection of receive filters to be enabled from the enumerated selection ETH_RECEIVE_FILTER

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ReceiveFilterEnable(index,  
                              ETH_CRC_OK_FILTER    |  
                              ETH_RUNT_ENABLE_FILTER |  
                              ETH_UNICAST_FILTER    );
```

5.6.8.5.4.19 PLIB_ETH_ReceiveFilterIsEnable Function

C

```
bool PLIB_ETH_ReceiveFilterIsEnable(  
    ETH_MODULE_ID index,  
    ETH_RECEIVE_FILTER filter  
);
```

Description

This function disables the specified receive filter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
filter	The selection of receive filters to be disabled from the enumerated selection ETH_RECEIVE_FILTER

Returns

- true - If at least one of the specified filters is enabled
- false - If no specified filter is enabled

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
if (PLIB_ETH_ReceiveFilterIsEnable(index, ETH_UNICAST_FILTER))  
{  
    PLIB_ETH_ReceiveFilterDisable(MY_MODULE_ID, ETH_UNICAST_FILTER);  
}
```

5.6.8.5.4.20 PLIB_ETH_StationAddressGet Function

C

```
uint8_t PLIB_ETH_StationAddressGet(  
    ETH_MODULE_ID index,  
    uint8_t which  
);
```

Description

This function gets the selected EMAC station address.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
which	Select station address to change. Valid values are 1,2,3,4,5,6.

Returns

stationAddress1 - Station address

Remarks

On a reset, this register is loaded with the factory preprogrammed station address.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stationAddr1 = PLIB_ETH_StationAddress1Get(MY_ETH_INSTANCE, which);
```

5.6.8.5.4.21 PLIB_ETH_StationAddressSet Function**C**

```
void PLIB_ETH_StationAddressSet(  
    ETH_MODULE_ID index,  
    uint8_t which,  
    uint8_t stationAddress  
);
```

Description

This function sets the selected EMAC Station Address.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
which	Select station address to change. Valid values are 1,2,3,4,5,6.
stationAddress	Station Address.

Returns

None.

Remarks

On a reset, this register is loaded with the factory preprogrammed station address.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_StationAddressSet(MY_ETH_INSTANCE, which, stationAddress);
```

5.6.8.5.5 Flow Control Functions**5.6.8.5.5.1 PLIB_ETH_AutoFlowControlDisable Function****C**

```
void PLIB_ETH_AutoFlowControlDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the Ethernet Automatic Flow Control logic.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_AutoFlowControlDisable(MY_ETH_INSTANCE);
```

5.6.8.5.5.2 PLIB_ETH_AutoFlowControlEnable Function**C**

```
void PLIB_ETH_AutoFlowControlEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables Ethernet Automatic Flow Control logic.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

The full and empty watermarks are used to automatically enable and disable flow control, respectively. When the number of received buffers rises to the full watermark, flow control is automatically enabled. When the receive buffer count falls to the empty watermark, flow control is automatically disabled.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_AutoFlowControlEnable(MY_ETH_INSTANCE);
```

5.6.8.5.5.3 PLIB_ETH_AutoFlowControlIsEnabled Function**C**

```
bool PLIB_ETH_AutoFlowControlIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the Ethernet Automatic Flow Control enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Ethernet module Automatic Flow Control is enabled
- false - Ethernet module Automatic Flow Control is disabled

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_AutoFlowControlIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.5.4 PLIB_ETH_BackPresNoBackoffDisable Function**C**

```
void PLIB_ETH_BackPresNoBackoffDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC back pressure/no back-off. The EMAC will back-off when there is backpressure and collisions occur.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_BackPresNoBackoffDisable(MY_ETH_INSTANCE);
```

5.6.8.5.5.5 PLIB_ETH_BackPresNoBackoffEnable Function**C**

```
void PLIB_ETH_BackPresNoBackoffEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC backpressure/no back-off. The EMAC will not back-off when there is backpressure and collisions occur.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

The EMAC after incidentally causing a collision during backpressure will immediately retransmit without back-off reducing the chance of further collisions and ensuring transmit packets get sent.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_BackPresNoBackoffEnable(MY_ETH_INSTANCE);
```

5.6.8.5.5.6 PLIB_ETH_BackPresNoBackoffIsEnabled Function**C**

```
bool PLIB_ETH_BackPresNoBackoffIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC back pressure/no back-off enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC after incidentally causing a collision during backpressure will immediately retransmit without back-off reducing the chance of further collisions and ensuring transmit packets get sent
- false - The EMAC will not remove the back-off

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_BackPresNoBackoffIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.5.7 PLIB_ETH_ManualFlowControlDisable Function**C**

```
void PLIB_ETH_ManualFlowControlDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the Ethernet Manual Flow Control logic and automatically sends a Pause frame with a 0x0000 Pause Timer value. This function affects both transmit and receive operations.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

Disabling Manual Flow Control will automatically [send](#) a Pause frame with a 0x0000 Pause Timer value to disable flow control.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ManualFlowControlDisable(MY_ETH_INSTANCE);
```

5.6.8.5.5.8 PLIB_ETH_ManualFlowControlEnable Function**C**

```
void PLIB_ETH_ManualFlowControlEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the Ethernet Manual Flow Control logic. Enabling Manual Flow Control will [send](#) a Pause frame using the Pause Timer value. While enabled, a Pause frame is repeated every $128 \times (\text{Pause timer value})/2$ transmit clock cycles.

Affects both transmit and receive operations.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ManualFlowControlEnable(MY_ETH_INSTANCE);
```

5.6.8.5.5.9 PLIB_ETH_ManualFlowControlIsEnabled Function**C**

```
bool PLIB_ETH_ManualFlowControlIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the Ethernet module Manual Flow Control enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Ethernet module Manual Flow Control is enabled
- false - Ethernet module Manual Flow Control is disabled

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_ManualFlowControlIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.5.10 PLIB_ETH_NoBackoffDisable Function**C**

```
void PLIB_ETH_NoBackoffDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC no back-off. The EMAC will back-off when a collision occurs.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_NoBackoffDisable(MY_ETH_INSTANCE);
```

5.6.8.5.5.11 PLIB_ETH_NoBackoffEnable Function**C**

```
void PLIB_ETH_NoBackoffEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC no back-off. The EMAC will not back-off when a collision occurs.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

Following a collision, the EMAC will immediately retransmit rather than using the Binary Exponential Back-off algorithm as specified in the Standard.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_NoBackoffEnable(MY_ETH_INSTANCE);
```

5.6.8.5.5.12 PLIB_ETH_NoBackoffIsEnabled Function**C**

```
bool PLIB_ETH_NoBackoffIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC no back-off enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Following a collision, the EMAC will immediately retransmit
- false - Following a collision, the EMAC will use the Binary Exponential Back-off algorithm

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_NoBackoffIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.5.13 PLIB_ETH_RxEmptyWmarkGet Function**C**

```
uint8_t PLIB_ETH_RxEmptyWmarkGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the Ethernet receive empty watermark.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

receiveEmptyWmark - Empty watermark value

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
watermarkValue = PLIB_ETH_RxEmptyWmarkGet(MY_ETH_INSTANCE);
```

5.6.8.5.5.14 PLIB_ETH_RxEmptyWmarkSet Function**C**

```
void PLIB_ETH_RxEmptyWmarkSet(  
    ETH_MODULE_ID index,  
    uint8_t watermarkValue  
);
```

Description

This function sets the Ethernet receive empty water mark with a new value. The software controlled Receive Buffer Empty Wmark is compared against the receive buffer count to determine the empty watermark condition for the empty watermark interrupt and for disabling flow control if Auto Flow Control is enabled.

The Full Wmark value should always be greater than the Empty Wmark value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
watermarkValue	Empty watermark value

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxEmptyWmarkSet(MY_ETH_INSTANCE, watermarkValue);
```

5.6.8.5.5.15 PLIB_ETH_RxFullWmarkGet Function

C

```
uint8_t PLIB_ETH_RxFullWmarkGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the Ethernet receive full watermark.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

receiveFullWmark - Full watermark value

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
watermarkValue = PLIB_ETH_RxFullWmarkGet(MY_ETH_INSTANCE);
```

5.6.8.5.5.16 PLIB_ETH_RxFullWmarkSet Function

C

```
void PLIB_ETH_RxFullWmarkSet(  
    ETH_MODULE_ID index,  
    uint8_t watermarkValue  
);
```

Description

This function sets the Ethernet receive full water mark with a new value.

The software controlled RX Buffer Full Wmark Pointer is compared against the receive buffer count to determine the full watermark condition for the full watermark interrupt and for enabling flow control if Auto Flow Control is enabled.

The Full Wmark value should always be greater than the Empty Wmark value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
watermarkValue	Full watermark value

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxFullWmarkSet(MY_ETH_INSTANCE, watermarkValue);
```

5.6.8.5.5.17 PLIB_ETH_RxPauseDisable Function**C**

```
void PLIB_ETH_RxPauseDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC receive flow control. The received Pause Flow control frames are ignored.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxPauseDisable(MY_ETH_INSTANCE);
```

5.6.8.5.5.18 PLIB_ETH_RxPauseEnable Function**C**

```
void PLIB_ETH_RxPauseEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC receive flow control. The EMAC will act upon the received Pause frames.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxPauseEnable(MY_ETH_INSTANCE);
```

5.6.8.5.5.19 PLIB_ETH_RxPauseIsEnabled Function

C

```
bool PLIB_ETH_RxPauseIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC receive flow pause enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC acts upon received Pause Flow Control frames
- false - Received Pause Flow Control frames are ignored

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
status = PLIB_ETH_RxPauseIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.5.20 PLIB_ETH_ShortcutQuantaDisable Function

C

```
void PLIB_ETH_ShortcutQuantaDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC shortcut pause quanta. The EMAC will resume normal operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ShortcutQuantaDisable(MY_ETH_INSTANCE);
```


5.6.8.5.5.21 PLIB_ETH_ShortcutQuantaEnable Function

C

```
void PLIB_ETH_ShortcutQuantaEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC shortcut pause quanta. The EMAC reduces the effective pause quanta from 64 byte-times to 1 byte-time.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_ShortcutQuantaEnable(MY_ETH_INSTANCE);
```

5.6.8.5.5.22 PLIB_ETH_ShortcutQuantaIsEnabled Function

C

```
bool PLIB_ETH_ShortcutQuantaIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns EMAC shortcut pause quanta enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC reduces the effective Pause Quanta from 64 byte-times to 1 byte-time
- false - Normal operation

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_ShortcutQuantaIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.5.23 PLIB_ETH_SimResetDisable Function

C

```
void PLIB_ETH_SimResetDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the EMAC simulation reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_SimResetDisable(MY_ETH_INSTANCE);
```

5.6.8.5.5.24 PLIB_ETH_SimResetEnable Function

C

```
void PLIB_ETH_SimResetEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the EMAC simulation reset and resets the random number generator within the transmit (TX) function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_SimResetEnable(MY_ETH_INSTANCE);
```

5.6.8.5.5.25 PLIB_ETH_SimResetIsEnabled Function

C

```
bool PLIB_ETH_SimResetIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function returns the EMAC simulation reset status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Simulation Reset is enabled
- false - Simulation Reset is disabled

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
stat = PLIB_ETH_SimResetIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.5.26 PLIB_ETH_TestBackPressDisable Function

C

```
void PLIB_ETH_TestBackPressDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables EMAC Test backpressure. The EMAC will resume normal operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TestBackPressDisable(MY_ETH_INSTANCE);
```

5.6.8.5.5.27 PLIB_ETH_TestBackPressEnable Function

C

```
void PLIB_ETH_TestBackPressEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables EMAC Test backpressure. The EMAC will assert backpressure on the link. Backpressure causes preamble to be transmitted, raising carrier sense. A transmit packet from the system will be sent during backpressure.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TestBackPressEnable(MY_ETH_INSTANCE);
```

5.6.8.5.5.28 PLIB_ETH_TestBackPressIsEnabled Function

C

```
bool PLIB_ETH_TestBackPressIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the EMAC test backpressure enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EMAC will assert backpressure on the link. Backpressure causes the preamble to be transmitted, raising the carrier sense. A transmit packet from the system will be sent during backpressure.
- false - Normal operation

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TestBackPressIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.5.29 PLIB_ETH_TxPauseDisable Function**C**

```
void PLIB_ETH_TxPauseDisable(  
    ETH_MODULE_ID index  
);
```

Description

This function disables the transmit Pause frames. The Pause frames are blocked from being transmitted.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TxPauseDisable(MY_ETH_INSTANCE);
```

5.6.8.5.5.30 PLIB_ETH_TxPauseEnable Function**C**

```
void PLIB_ETH_TxPauseEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function enables the transmission Pause frames. The Pause frames are allowed for transmission.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_TxPauseEnable(MY_ETH_INSTANCE);
```

5.6.8.5.5.31 PLIB_ETH_TxPauseIsEnabled Function

C

```
bool PLIB_ETH_TxPauseIsEnabled(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the Ethernet module enable status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - Pause Flow Control frames are allowed to be transmitted
- false - Pause Flow Control frames are blocked

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
status = PLIB_ETH_TxPauseIsEnabled(MY_ETH_INSTANCE);
```

5.6.8.5.6 Interrupt Functions

5.6.8.5.6.1 PLIB_ETH_InterruptClear Function

C

```
void PLIB_ETH_InterruptClear(  
    ETH_MODULE_ID index,  
    ETH_INTERRUPT_SOURCES intmask  
);
```

Description

This function clears the Ethernet module interrupt source status using a mask. Logically 'OR' the masks together. Any masks not ORed with the others will be disabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
// To clear several interrupts with one command, execute the following:
PLIB_ETH_InterruptClear(MY_ETH_INSTANCE,
                        ETH_EMPTY_WATERMARK_INTERRUPT |
                        ETH_FULL_WATERMARK_INTERRUPT |
                        ETH_RX_FIFO_OVERFLOW_ERROR_INTERRUPT );
```

5.6.8.5.6.2 PLIB_ETH_InterruptSet Function**C**

```
void PLIB_ETH_InterruptSet(
    ETH_MODULE_ID index,
    ETH_INTERRUPT_SOURCES intmask
);
```

Description

This function sets the Ethernet module interrupt source status using a mask. Logically 'OR' the masks together. Any masks not OR'd with the others will be disabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
// To set several interrupt flags with one command, execute the following:
PLIB_ETH_InterruptSet(MY_ETH_INSTANCE,
                      ETH_EMPTY_WATERMARK_INTERRUPT |
                      ETH_FULL_WATERMARK_INTERRUPT |
                      ETH_RX_FIFO_OVERFLOW_ERROR_INTERRUPT );
```

5.6.8.5.6.3 PLIB_ETH_InterruptsGet Function**C**

```
ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptsGet(
    ETH_MODULE_ID index
);
```

Description

Gets the Ethernet module Interrupt Flag register as a group.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Interrupt bit map, as defined in [ETH_INTERRUPT_SOURCES](#), showing which interrupts have fired.

Remarks

none.

Example**5.6.8.5.6.4 PLIB_ETH_InterruptSourceDisable Function****C**

```
void PLIB_ETH_InterruptSourceDisable(  
    ETH_MODULE_ID index,  
    ETH_INTERRUPT_SOURCES intmask  
);
```

Description

This function disables the Ethernet module interrupts by using a mask to select one or multiple interrupts to disable. Logically 'OR' elements of [ETH_INTERRUPT_SOURCES](#) together.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
// To disable some interrupts with one command, execute the following:  
PLIB_ETH_InterruptSourceDisble(MY_ETH_INSTANCE,  
    ETH_TRANSMIT_DONE_INTERRUPT  
    ETH_TRANSMIT_ABORT_INTERRUPT  
    ETH_RECEIVE_BUFFER_NOT_AVAILABLE_INTERRUPT  
    ETH_RECEIVE_FIFO_OVERFLOW_ERROR_INTERRUPT );  
  
// Or to disable one interrupt with one command, execute the following:  
PLIB_ETH_InterruptSourceDisble(MY_ETH_INSTANCE, ETH_TRANSMIT_DONE_INTERRUPT);  
  
// Or to disable all interrupts with one command, execute the following:  
PLIB_ETH_InterruptSourceDisble(MY_ETH_INSTANCE, ETH_ALL_ETH_INTERRUPTS);
```

5.6.8.5.6.5 PLIB_ETH_InterruptSourceEnable Function**C**

```
void PLIB_ETH_InterruptSourceEnable(  

```



```

    ETH_MODULE_ID index,
    ETH_INTERRUPT_SOURCES intmask
);

```

Description

This function enables the Ethernet module interrupts by using a mask to enable one or multiple interrupt enables. Logically 'OR' elements of [ETH_INTERRUPT_SOURCES](#) together.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```

// To enable some interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceEnable(MY_ETH_INSTANCE,
                                ETH_TRANSMIT_DONE_INTERRUPT
                                ETH_TRANSMIT_ABORT_INTERRUPT
                                ETH_RECEIVE_BUFFER_NOT_AVAILABLE_INTERRUPT
                                ETH_RECEIVE_FIFO_OVERFLOW_ERROR_INTERRUPT);

// Or to enable one interrupt with one command, execute the following:
PLIB_ETH_InterruptSourceEnable(MY_ETH_INSTANCE, ETH_TRANSMIT_DONE_INTERRUPT);

// Or to enable all interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceEnable(MY_ETH_INSTANCE, ETH_ALL_ETH_INTERRUPTS);

```

5.6.8.5.6 PLIB_ETH_InterruptSourceIsEnabled Function

C

```

bool PLIB_ETH_InterruptSourceIsEnabled(
    ETH_MODULE_ID index,
    ETH_INTERRUPT_SOURCES intmask
);

```

Description

This function returns a true if any of the specified interrupt sources are enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
mask	The interrupt mask(s) to be checked

Returns

- true - If any of the specified sources are enabled

- false - If none of the specified sources are enabled

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
if ( PLIB_ETH_InterruptSourceIsEnabled(index, ETH_TRANSMIT_DONE_INTERRUPT))
{
    //If the interrupt is enabled, disable it...
    PLIB_ETH_InterruptSourceDisable(index, ETH_TRANSMIT_DONE_INTERRUPT);
}
```

5.6.8.5.6.7 PLIB_ETH_InterruptSourcesGet Function**C**

```
ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptSourcesGet(
    ETH_MODULE_ID index
);
```

Description

This function returns the entire interrupt enable register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Bit map of interrupt sources, all ORed together, as defined by [ETH_INTERRUPT_SOURCES](#).

Remarks

None.

Example**5.6.8.5.6.8 PLIB_ETH_InterruptStatusGet Function****C**

```
bool PLIB_ETH_InterruptStatusGet(
    ETH_MODULE_ID index,
    ETH_INTERRUPT_SOURCES intmask
);
```

Description

This function gets the Ethernet module Interrupt Flag register using a mask.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Returns

- true - If any of the specified source statuses are enabled
- false - If none of the specified source statuses are enabled

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
if (PLIB_ETH_InterruptStatusGet(index, ETH_TRANSMIT_DONE_INTERRUPT))
{
    PLIB_ETH_InterruptClear(index, ETH_TRANSMIT_DONE_INTERRUPT);
}
```

5.6.8.5.7 Statistics Functions

5.6.8.5.7.1 PLIB_ETH_AlignErrorCountClear Function

C

```
void PLIB_ETH_AlignErrorCountClear(
    ETH_MODULE_ID index
);
```

Description

This function sets the count of Ethernet alignment errors to zero.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	count of alignment errors

Returns

None.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_AlignErrorCountClear(MY_ETH_INSTANCE);
```

5.6.8.5.7.2 PLIB_ETH_AlignErrorCountGet Function

C

```
uint16_t PLIB_ETH_AlignErrorCountGet(
    ETH_MODULE_ID index
);
```

Description

This function gets the count of Ethernet alignment errors.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

value - Count of alignment errors

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
count = PLIB_ETH_AlignErrorCountGet(MY_ETH_INSTANCE);
```

5.6.8.5.7.3 PLIB_ETH_FCSErrorCountClear Function**C**

```
void PLIB_ETH_FCSErrorCountClear(  
    ETH_MODULE_ID index  
);
```

Description

This function sets the value of the Ethernet frame check sequence error to zero.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of FCS Errors

Returns

None.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_FCSErrorCountClear(MY_ETH_INSTANCE);
```

5.6.8.5.7.4 PLIB_ETH_FCSErrorCountGet Function**C**

```
uint16_t PLIB_ETH_FCSErrorCountGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the count of the frame check sequence error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

value - Count of FCS Errors

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
count = PLIB_ETH_FCSErrorCountGet(MY_ETH_INSTANCE);
```

5.6.8.5.7.5 PLIB_ETH_FramesRxdOkCountClear Function**C**

```
void PLIB_ETH_FramesRxdOkCountClear(  
    ETH_MODULE_ID index  
);
```

Description

This function sets the value of the Ethernet frames 'OK' count to zero.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of frames received correctly

Returns

None.

Remarks

Changing the value of this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_FramesRxdOkCountClear(MY_ETH_INSTANCE);
```

5.6.8.5.7.6 PLIB_ETH_FramesRxdOkCountGet Function**C**

```
uint16_t PLIB_ETH_FramesRxdOkCountGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the count of the frames received successfully. Increment count for frames received successfully by the RX Filter. This count will not be incremented if there is a Frame Check Sequence (FCS) or Alignment error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

value - Count of frames received correctly

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
count = PLIB_ETH_FramesRxdOkCountGet(MY_ETH_INSTANCE);
```

5.6.8.5.7.7 PLIB_ETH_FramesTxdOkCountClear Function**C**

```
void PLIB_ETH_FramesTxdOkCountClear(  
    ETH_MODULE_ID index  
);
```

Description

This function sets the count of Ethernet Control frames transmitted to zero.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of control frames transmitted correctly

Returns

None.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_FramesTxdOkCountClear(MY_ETH_INSTANCE);
```

5.6.8.5.7.8 PLIB_ETH_FramesTxdOkCountGet Function**C**

```
uint16_t PLIB_ETH_FramesTxdOkCountGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the count of Ethernet Control Frames transmitted successfully.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

count - count of control frames transmitted correctly

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
count = PLIB_ETH_FramesTxdOkCountGet(MY_ETH_INSTANCE);
```

5.6.8.5.7.9 PLIB_ETH_MultipleCollisionCountClear Function**C**

```
void PLIB_ETH_MultipleCollisionCountClear(  
    ETH_MODULE_ID index  
);
```

Description

This function sets the value of the Ethernet multiple collision frame count to zero.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of multiple collision frames

Returns

None.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_MultipleCollisionCountClear(MY_ETH_INSTANCE);
```

5.6.8.5.7.10 PLIB_ETH_MultipleCollisionCountGet Function**C**

```
uint16_t PLIB_ETH_MultipleCollisionCountGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the count of the frames transmitted successfully after there was more than one collision.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

count - Count of multiple collision frames

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
count = PLIB_ETH_MultipleCollisionCountGet(MY_ETH_INSTANCE);
```

5.6.8.5.7.11 PLIB_ETH_RxOverflowCountClear Function**C**

```
void PLIB_ETH_RxOverflowCountClear(  
    ETH_MODULE_ID index  
);
```

Description

This function sets the value of the Ethernet receive overflow count to zero. Increment counter for frames accepted by the RX filter and subsequently dropped due to internal receive error. This event also sets the receive overflow interrupt flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

None.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_RxOverflowCountClear(MY_ETH_INSTANCE);
```

5.6.8.5.7.12 PLIB_ETH_RxOverflowCountGet Function**C**

```
uint16_t PLIB_ETH_RxOverflowCountGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the count of the dropped Ethernet receive frames.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

count - uint16_t receiver overflow counts

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
count = PLIB_ETH_RxOverflowCountGet(MY_ETH_INSTANCE);
```

5.6.8.5.7.13 PLIB_ETH_RxPacketCountGet Function**C**

```
uint8_t PLIB_ETH_RxPacketCountGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the value of the the receive packet buffer count. When a packet has been successfully received, this value is incremented by hardware. Software decrements the counter after a packet has been read. Call PLIB_ETH_ReceiveBufferCountDecrement(ETH_MODULE_ID index) to decrement.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

RxPacketCount - Number of received packets in memory

Remarks

Receive Packet Buffer Counter cannot be decremented below 0x00 Cleared when the Ethernet module is reset.

The Receive Packet Buffer Count is not set to '0' when the Ethernet module is turned OFF. This allows software to continue to utilize and decrement the count.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
count = PLIB_ETH_RxPacketCountGet(MY_ETH_INSTANCE);
```

5.6.8.5.7.14 PLIB_ETH_SingleCollisionCountClear Function**C**

```
void PLIB_ETH_SingleCollisionCountClear(  
    ETH_MODULE_ID index  
);
```

Description

This function sets the value of the Ethernet single collision frame count to zero.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of single collision frames

Returns

None.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_ETH_SingleCollisionCountClear(MY_ETH_INSTANCE);
```

5.6.8.5.7.15 PLIB_ETH_SingleCollisionCountGet Function**C**

```
uint16_t PLIB_ETH_SingleCollisionCountGet(  
    ETH_MODULE_ID index  
);
```

Description

This function gets the count of the frames transmitted successfully on a second attempt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

count - Count of frames transmitted successfully on second attempt

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
count = PLIB_ETH_SingleCollisionCountGet(MY_ETH_INSTANCE);
```

5.6.8.5.8 Existence Functions**5.6.8.5.8.1 PLIB_ETH_ExistsAlignmentErrorCount Function****C**

```
bool PLIB_ETH_ExistsAlignmentErrorCount(  
    ETH_MODULE_ID index
```

```
);
```

Description

This function identifies whether the AlignmentErrorCount feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_AlignErrorCountClear](#)
- [PLIB_ETH_AlignErrorCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlignmentErrorCount feature is supported on the device
- false - The AlignmentErrorCount feature is not supported on the device

Remarks

None.

5.6.8.5.8.2 PLIB_ETH_ExistsAutoFlowControl Function

C

```
bool PLIB_ETH_ExistsAutoFlowControl(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the AutoFlowControl feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_AutoFlowControlEnable](#)
- [PLIB_ETH_AutoFlowControlDisable](#)
- [PLIB_ETH_AutoFlowControlIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AutoFlowControl feature is supported on the device
- false - The AutoFlowControl feature is not supported on the device

Remarks

None.

5.6.8.5.8.3 PLIB_ETH_ExistsCollisionCounts Function

C

```
bool PLIB_ETH_ExistsCollisionCounts(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the CollisionCounts feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_SingleCollisionCountClear](#)
- [PLIB_ETH_SingleCollisionCountGet](#)
- [PLIB_ETH_MultipleCollisionCountClear](#)
- [PLIB_ETH_MultipleCollisionCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CollisionCounts feature is supported on the device
- false - The CollisionCounts feature is not supported on the device

Remarks

None.

5.6.8.5.8.4 PLIB_ETH_ExistsCollisionWindow Function

C

```
bool PLIB_ETH_ExistsCollisionWindow(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the CollisionWindow feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_CollisionWindowGet](#)
- [PLIB_ETH_CollisionWindowSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CollisionWindow feature is supported on the device

- false - The CollisionWindow feature is not supported on the device

Remarks

None.

5.6.8.5.8.5 PLIB_ETH_ExistsEnable Function

C

```
bool PLIB_ETH_ExistsEnable(  
    ETH_MODULE_ID index  
)
```

Description

This function identifies whether the Enable feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_Enable](#)
- [PLIB_ETH_Disable](#)
- [PLIB_ETH_IsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Enable feature is supported on the device
- false - The Enable feature is not supported on the device

Remarks

None.

5.6.8.5.8.6 PLIB_ETH_ExistsEthernetControllerStatus Function

C

```
bool PLIB_ETH_ExistsEthernetControllerStatus(  
    ETH_MODULE_ID index  
)
```

Description

This function identifies whether the EthernetControllerStatus feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_RxPacketCountGet](#)
- [PLIB_ETH_EthernetIsBus](#)
- [PLIB_ETH_TransmitsBusy](#)
- [PLIB_ETH_ReceiveIsBusy](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EthernetControllerStatus feature is supported on the device
- false - The EthernetControllerStatus feature is not supported on the device

Remarks

None.

5.6.8.5.8.7 PLIB_ETH_ExistsFCSErrorCount Function**C**

```
bool PLIB_ETH_ExistsFCSErrorCount(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the FCSErrorCount feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_FCSErrorCountClear](#)
- [PLIB_ETH_FCSErrorCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FCSErrorCount feature is supported on the device
- false - The FCSErrorCount feature is not supported on the device

Remarks

None.

5.6.8.5.8.8 PLIB_ETH_ExistsFramesTransmittedOK Function**C**

```
bool PLIB_ETH_ExistsFramesTransmittedOK(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the FramesTransmittedOK feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_FramesTxdOkCountClear](#)
- [PLIB_ETH_FramesTxdOkCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FramesTransmittedOK feature is supported on the device
- false - The FramesTransmittedOK feature is not supported on the device

Remarks

None.

5.6.8.5.8.9 PLIB_ETH_ExistsFramexReceivedOK Function**C**

```
bool PLIB_ETH_ExistsFramexReceivedOK(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the FramexReceivedOK feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_FramesRxdOkCountClear](#)
- [PLIB_ETH_FramesRxdOkCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FramexReceivedOK feature is supported on the device
- false - The FramexReceivedOK feature is not supported on the device

Remarks

None.

5.6.8.5.8.10 PLIB_ETH_ExistsHashTable Function**C**

```
bool PLIB_ETH_ExistsHashTable(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the HashTable feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_HashTableSet](#)
- [PLIB_ETH_HashTableGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The HashTable feature is supported on the device
- false - The HashTable feature is not supported on the device

Remarks

None.

5.6.8.5.8.11 PLIB_ETH_ExistsInterPacketGaps Function**C**

```
bool PLIB_ETH_ExistsInterPacketGaps(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the InterPacketGaps feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_BackToBackIPGGet](#)
- [PLIB_ETH_BackToBackIPGSet](#)
- [PLIB_ETH_NonBackToBackIPG1Get](#)
- [PLIB_ETH_NonBackToBackIPG1Set](#)
- [PLIB_ETH_NonBackToBackIPG2Get](#)
- [PLIB_ETH_NonBackToBackIPG2Set](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InterPacketGaps feature is supported on the device
- false - The InterPacketGaps feature is not supported on the device

Remarks

None.

5.6.8.5.8.12 PLIB_ETH_ExistsInterrupt Function

C

```
bool PLIB_ETH_ExistsInterrupt(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the Interrupt feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_InterruptSourceEnable](#)
- [PLIB_ETH_InterruptSourceDisable](#)
- [PLIB_ETH_InterruptSourceIsEnabled](#)
- [PLIB_ETH_InterruptSet](#)
- [PLIB_ETH_InterruptClear](#)
- [PLIB_ETH_InterruptStatusGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Interrupt feature is supported on the device
- false - The Interrupt feature is not supported on the device

Remarks

None.

5.6.8.5.8.13 PLIB_ETH_ExistsMAC_Configuration Function

C

```
bool PLIB_ETH_ExistsMAC_Configuration(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the MAC_Configuration feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_LoopbackEnable](#)
- [PLIB_ETH_LoopbackDisable](#)
- [PLIB_ETH_LoopbackIsEnabled](#)
- [PLIB_ETH_TxPauseEnable](#)
- [PLIB_ETH_TxPauseDisable](#)
- [PLIB_ETH_TxPauselsEnabled](#)
- [PLIB_ETH_RxPauseEnable](#)

- [PLIB_ETH_RxPauseDisable](#)
- [PLIB_ETH_RxPausesIsEnabled](#)
- [PLIB_ETH_PassAllEnable](#)
- [PLIB_ETH_PassAllDisable](#)
- [PLIB_ETH_PassAllIsEnabled](#)
- [PLIB_ETH_ReceiveEnable](#)
- [PLIB_ETH_ReceiveDisable](#)
- [PLIB_ETH_ReceiveIsEnabled](#)
- [PLIB_ETH_ExcessDeferEnable](#)
- [PLIB_ETH_ExcessDeferDisable](#)
- [PLIB_ETH_ExcessDeferIsEnabled](#)
- [PLIB_ETH_BackPresNoBackoffEnable](#)
- [PLIB_ETH_BackPresNoBackoffDisable](#)
- [PLIB_ETH_BackPresNoBackoffIsEnabled](#)
- [PLIB_ETH_NoBackoffEnable](#)
- [PLIB_ETH_NoBackoffDisable](#)
- [PLIB_ETH_NoBackoffIsEnabled](#)
- [PLIB_ETH_LongPreambleEnable](#)
- [PLIB_ETH_LongPreambleDisable](#)
- [PLIB_ETH_LongPreambleIsEnabled](#)
- [PLIB_ETH_PurePreambleEnable](#)
- [PLIB_ETH_PurePreambleDisable](#)
- [PLIB_ETH_PurePreambleIsEnabled](#)
- [PLIB_ETH_AutoDetectPadGet](#)
- [PLIB_ETH_AutoDetectPadSet](#)
- [PLIB_ETH_AutoDetectPadClear](#)
- [PLIB_ETH_CRCEnable](#)
- [PLIB_ETH_CRCDisable](#)
- [PLIB_ETH_CRCIsEnabled](#)
- [PLIB_ETH_DelayedCRCEnable](#)
- [PLIB_ETH_DelayedCRCDisable](#)
- [PLIB_ETH_DelayedCRCIsEnabled](#)
- [PLIB_ETH_HugeFrameEnable](#)
- [PLIB_ETH_HugeFrameDisable](#)
- [PLIB_ETH_HugeFrameIsEnabled](#)
- [PLIB_ETH_FrameLengthCheckEnable](#)
- [PLIB_ETH_FrameLengthCheckDisable](#)
- [PLIB_ETH_FrameLengthCheckIsEnabled](#)
- [PLIB_ETH_FullDuplexEnable](#)

- [PLIB_ETH_FullDuplexDisable](#)
- [PLIB_ETH_FullDuplexIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MAC_Configuration feature is supported on the device
- false - The MAC_Configuration feature is not supported on the device

Remarks

None.

5.6.8.5.8.14 PLIB_ETH_ExistsMAC_Resets Function**C**

```
bool PLIB_ETH_ExistsMAC_Resets(
    ETH_MODULE_ID index
);
```

Description

This function identifies whether the MAC_Resets feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_MIIResetEnable](#)
- [PLIB_ETH_MIIResetDisable](#)
- [PLIB_ETH_MIIResetIsEnabled](#)
- [PLIB_ETH_SimResetEnable](#)
- [PLIB_ETH_SimResetDisable](#)
- [PLIB_ETH_SimResetIsEnabled](#)
- [PLIB_ETH_MCSRxResetEnable](#)
- [PLIB_ETH_MCSRxResetDisable](#)
- [PLIB_ETH_MCSRxResetIsEnabled](#)
- [PLIB_ETH_RxFuncResetEnable](#)
- [PLIB_ETH_RxFuncResetDisable](#)
- [PLIB_ETH_RxFuncResetIsEnabled](#)
- [PLIB_ETH_MCSTxResetEnable](#)
- [PLIB_ETH_MCSTxResetDisable](#)
- [PLIB_ETH_MCSTxResetIsEnabled](#)
- [PLIB_ETH_TxFuncResetEnable](#)
- [PLIB_ETH_TxFuncResetDisable](#)
- [PLIB_ETH_TxFuncResetIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MAC_Resets feature is supported on the device
- false - The MAC_Resets feature is not supported on the device

Remarks

None.

5.6.8.5.8.15 PLIB_ETH_ExistsMAC_Testing Function**C**

```
bool PLIB_ETH_ExistsMAC_Testing(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the MAC_Testing feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_TestBackPressEnable](#)
- [PLIB_ETH_TestBackPressDisable](#)
- [PLIB_ETH_TestBackPressIsEnabled](#)
- [PLIB_ETH_TestPauseEnable](#)
- [PLIB_ETH_TestPauseDisable](#)
- [PLIB_ETH_TestPauseIsEnabled](#)
- [PLIB_ETH_ShortcutQuantaEnable](#)
- [PLIB_ETH_ShortcutQuantaDisable](#)
- [PLIB_ETH_ShortcutQuantaIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MAC_Testing feature is supported on the device
- false - The MAC_Testing feature is not supported on the device

Remarks

None.

5.6.8.5.8.16 PLIB_ETH_ExistsManualFlowControl Function

C

```
bool PLIB_ETH_ExistsManualFlowControl(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the ManualFlowControl feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_ManualFlowControlEnable](#)
- [PLIB_ETH_ManualFlowControlDisable](#)
- [PLIB_ETH_ManualFlowControlIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ManualFlowControl feature is supported on the device
- false - The ManualFlowControl feature is not supported on the device

Remarks

None.

5.6.8.5.8.17 PLIB_ETH_ExistsMaxFrameLength Function

C

```
bool PLIB_ETH_ExistsMaxFrameLength(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the MaxFrameLength feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_MaxFrameLengthGet](#)
- [PLIB_ETH_MaxFrameLengthSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MaxFrameLength feature is supported on the device
- false - The MaxFrameLength feature is not supported on the device

Remarks

None.

5.6.8.5.8.18 PLIB_ETH_ExistsMIIM_Config Function**C**

```
bool PLIB_ETH_ExistsMIIM_Config(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the MIIM_Config feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_MIIMResetEnable](#)
- [PLIB_ETH_MIIMResetDisable](#)
- [PLIB_ETH_MIIMResetIsEnabled](#)
- [PLIB_ETH_MIIMClockGet](#)
- [PLIB_ETH_MIIMClockSet](#)
- [PLIB_ETH_MIIMNoPreEnable](#)
- [PLIB_ETH_MIIMNoPreDisable](#)
- [PLIB_ETH_MIIMNoPreIsEnabled](#)
- [PLIB_ETH_MIIMScanIncrEnable](#)
- [PLIB_ETH_MIIMScanIncrDisable](#)
- [PLIB_ETH_MIIMScanIncrIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MIIM_Config feature is supported on the device
- false - The MIIM_Config feature is not supported on the device

Remarks

None.

5.6.8.5.8.19 PLIB_ETH_ExistsMIIM_Indicators Function**C**

```
bool PLIB_ETH_ExistsMIIM_Indicators(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the MIIM_Indicators feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_LinkHasFailed](#)
- [PLIB_ETH_DataNotValid](#)
- [PLIB_ETH_MIIMIsScanning](#)
- [PLIB_ETH_MIIMIsBusy](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MIIM_Indicators feature is supported on the device
- false - The MIIM_Indicators feature is not supported on the device

Remarks

None.

5.6.8.5.8.20 PLIB_ETH_ExistsMIIMAddresses Function**C**

```
bool PLIB_ETH_ExistsMIIMAddresses(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the MIIMAddresses feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_PHYAddressGet](#)
- [PLIB_ETH_PHYAddressSet](#)
- [PLIB_ETH_RegisterAddressGet](#)
- [PLIB_ETH_RegisterAddressSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MIIMAddresses feature is supported on the device
- false - The MIIMAddresses feature is not supported on the device

Remarks

None.

5.6.8.5.8.21 PLIB_ETH_ExistsMIIMReadWrite Function

C

```
bool PLIB_ETH_ExistsMIIMReadWrite(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the MIIMReadWrite feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_MIIMReadStart](#)
- [PLIB_ETH_MIIMWriteStart](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MIIMReadWrite feature is supported on the device
- false - The MIIMReadWrite feature is not supported on the device

Remarks

None.

5.6.8.5.8.22 PLIB_ETH_ExistsMIIMScanMode Function

C

```
bool PLIB_ETH_ExistsMIIMScanMode(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the MIIMScanMode feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_MIIMScanModeEnable](#)
- [PLIB_ETH_MIIMScanModeDisable](#)
- [PLIB_ETH_MIIMScanModeIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MIIMScanMode feature is supported on the device
- false - The MIIMScanMode feature is not supported on the device

Remarks

None.

5.6.8.5.8.23 PLIB_ETH_ExistsMIWriteReadData Function**C**

```
bool PLIB_ETH_ExistsMIWriteReadData(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the MIWriteReadData feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_MIIMWriteDataSet](#)
- [PLIB_ETH_MIIMReadDataGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MIWriteReadData feature is supported on the device
- false - The MIWriteReadData feature is not supported on the device

Remarks

None.

5.6.8.5.8.24 PLIB_ETH_ExistsPatternMatch Function**C**

```
bool PLIB_ETH_ExistsPatternMatch(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the PatternMatch feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_PatternMatchSet](#)
- [PLIB_ETH_PatternMatchGet](#)
- [PLIB_ETH_PatternMatchChecksumSet](#)
- [PLIB_ETH_PatternMatchChecksumGet](#)
- [PLIB_ETH_PatternMatchOffsetSet](#)
- [PLIB_ETH_PatternMatchOffsetGet](#)
- [PLIB_ETH_PatternMatchModeSet](#)
- [PLIB_ETH_PatternMatchModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PatternMatch feature is supported on the device
- false - The PatternMatch feature is not supported on the device

Remarks

None.

5.6.8.5.8.25 PLIB_ETH_ExistsPauseTimer Function**C**

```
bool PLIB_ETH_ExistsPauseTimer(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the PauseTimer feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_PauseTimerSet](#)
- [PLIB_ETH_PauseTimerGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PauseTimer feature is supported on the device
- false - The PauseTimer feature is not supported on the device

Remarks

None.

5.6.8.5.8.26 PLIB_ETH_ExistsReceiveBufferSize Function**C**

```
bool PLIB_ETH_ExistsReceiveBufferSize(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveBufferSize feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_ReceiveBufferSizeGet](#)
- [PLIB_ETH_ReceiveBufferSizeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiveBufferSize feature is supported on the device
- false - The ReceiveBufferSize feature is not supported on the device

Remarks

None.

5.6.8.5.8.27 PLIB_ETH_ExistsReceiveFilters Function**C**

```
bool PLIB_ETH_ExistsReceiveFilters(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveFilters feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_ReceiveFilterEnable](#)
- [PLIB_ETH_ReceiveFilterDisable](#)
- [PLIB_ETH_ReceiveFilterIsEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiveFilters feature is supported on the device
- false - The ReceiveFilters feature is not supported on the device

Remarks

None.

5.6.8.5.8.28 PLIB_ETH_ExistsReceiveOverflowCount Function**C**

```
bool PLIB_ETH_ExistsReceiveOverflowCount(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveOverflowCount feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_RxOverflowCountClear](#)
- [PLIB_ETH_RxOverflowCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiveOverflowCount feature is supported on the device
- false - The ReceiveOverflowCount feature is not supported on the device

Remarks

None.

5.6.8.5.8.29 PLIB_ETH_ExistsReceiveWmarks Function**C**

```
bool PLIB_ETH_ExistsReceiveWmarks(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveWmarks feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_RxFullWmarkSet](#)
- [PLIB_ETH_RxFullWmarkGet](#)
- [PLIB_ETH_RxEmptyWmarkSet](#)
- [PLIB_ETH_RxEmptyWmarkGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiveWmarks feature is supported on the device
- false - The ReceiveWmarks feature is not supported on the device

Remarks

None.

5.6.8.5.8.30 PLIB_ETH_ExistsRetransmissionMaximum Function

C

```
bool PLIB_ETH_ExistsRetransmissionMaximum(  
    ETH_MODULE_ID index  
)  
;
```

Description

This function identifies whether the RetransmissionMaximum feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_ReTxMaxGet](#)
- [PLIB_ETH_ReTxMaxSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RetransmissionMaximum feature is supported on the device
- false - The RetransmissionMaximum feature is not supported on the device

Remarks

None.

5.6.8.5.8.31 PLIB_ETH_ExistsRMII_Support Function

C

```
bool PLIB_ETH_ExistsRMII_Support(  
    ETH_MODULE_ID index  
)  
;
```

Description

This function identifies whether the RMII_Support feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_RMIIResetEnable](#)
- [PLIB_ETH_RMIIResetDisable](#)
- [PLIB_ETH_RMIIResetIsEnabled](#)
- [PLIB_ETH_RMIIISpeedGet](#)
- [PLIB_ETH_RMIIISpeedSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RMII_Support feature is supported on the device
- false - The RMII_Support feature is not supported on the device

Remarks

None.

5.6.8.5.8.32 PLIB_ETH_ExistsRxBufferCountDecrement Function**C**

```
bool PLIB_ETH_ExistsRxBufferCountDecrement(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the RxBufferCountDecrement feature is available on the Ethernet module. When this function returns true, this function is supported on the device:

- [PLIB_ETH_RxBufferCountDecrement](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RxBufferCountDecrement feature is supported on the device
- false - The RxBufferCountDecrement feature is not supported on the device

Remarks

None.

5.6.8.5.8.33 PLIB_ETH_ExistsRxEnable Function**C**

```
bool PLIB_ETH_ExistsRxEnable(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveEnable feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_RxEnable](#)
- [PLIB_ETH_RxDisable](#)
- [PLIB_ETH_RxIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiveEnable feature is supported on the device
- false - The ReceiveEnable feature is not supported on the device

Remarks

None.

5.6.8.5.8.34 PLIB_ETH_ExistsRxPacketDescriptorAddress Function**C**

```
bool PLIB_ETH_ExistsRxPacketDescriptorAddress(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the RxPacketDescriptorAddress feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_RxPacketDescAddrSet](#)
- [PLIB_ETH_RxPacketDescAddrGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RxPacketDescriptorAddress feature is supported on the device
- false - The RxPacketDescriptorAddress feature is not supported on the device

Remarks

None.

5.6.8.5.8.35 PLIB_ETH_ExistsStationAddress Function**C**

```
bool PLIB_ETH_ExistsStationAddress(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the StationAddress feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_StationAddressGet](#)
- [PLIB_ETH_StationAddressSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StationAddress feature is supported on the device
- false - The StationAddress feature is not supported on the device

Remarks

None.

5.6.8.5.8.36 PLIB_ETH_ExistsStopInIdle Function**C**

```
bool PLIB_ETH_ExistsStopInIdle(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_StopInIdleEnable](#)
- [PLIB_ETH_StopInIdleDisable](#)
- [PLIB_ETH_StopInIdleIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Remarks

None.

5.6.8.5.8.37 PLIB_ETH_ExistsTransmitRTS Function**C**

```
bool PLIB_ETH_ExistsTransmitRTS(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitRTS feature is available on the Ethernet module. When this function returns true,

these functions are supported on the device:

- [PLIB_ETH_TxRTSEnable](#)
- [PLIB_ETH_TxRTSDisable](#)
- [PLIB_ETH_TxRTSIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitRTS feature is supported on the device
- false - The TransmitRTS feature is not supported on the device

Remarks

None.

5.6.8.5.8.38 PLIB_ETH_ExistsTxPacketDescriptorAddress Function

C

```
bool PLIB_ETH_ExistsTxPacketDescriptorAddress(  
    ETH_MODULE_ID index  
);
```

Description

This function identifies whether the TxPacketDescriptorAddress feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- [PLIB_ETH_TxPacketDescAddrSet](#)
- [PLIB_ETH_TxPacketDescAddrGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TxPacketDescriptorAddress feature is supported on the device
- false - The TxPacketDescriptorAddress feature is not supported on the device

Remarks

None.

5.6.8.5.9 Data Types and Constants

5.6.8.5.9.1 ETH_AUTOPAD_OPTION Enumeration

C

```
typedef enum {  
    ETH_AUTOPAD_DISABLED_CHECK_CRC,  
    ETH_AUTOPAD_TO_60BYTES_ADD_CRC,  
    ETH_AUTOPAD_TO_64BYTES_ADD_CRC,  
    ETH_AUTOPAD_BY_PKT_TYPE_ADD_CRC  
} ETH_AUTOPAD_OPTION;
```

Description

AutoDetectPad Selection

This enumeration lists the possible automatic padding configurations.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.8.5.9.2 ETH_INTERRUPT_SOURCES Enumeration

C

```
typedef enum {  
    ETH_TRANSMIT_BVCI_BUS_ERROR_INTERRUPT,  
    ETH_RECEIVE_BVCI_BUS_ERROR_INTERRUPT,  
    ETH_EMPTY_WATERMARK_INTERRUPT,  
    ETH_FULL_WATERMARK_INTERRUPT,  
    ETH_RECEIVE_DONE_INTERRUPT,  
    ETH_PACKET_PENDING_INTERRUPT,  
    ETH_RECEIVE_ACTIVITY_INTERRUPT,  
    ETH_TRANSMIT_DONE_INTERRUPT,  
    ETH_TRANSMIT_ABORT_INTERRUPT,  
    ETH_RECEIVE_BUFFER_NOT_AVAILABLE_INTERRUPT,  
    ETH_RECEIVE_FIFO_OVERFLOW_ERROR_INTERRUPT,  
    ETH_ALL_INTERRUPT_SOURCES  
} ETH_INTERRUPT_SOURCES;
```

Description

ETH Module Interrupt Mask

This enumeration lists the possible values of ETH_INTERRUPT_SOURCES.

Remarks

See also ETH_LEGACY_EVENTS in plib_eth_legacy.h.

5.6.8.5.9.3 ETH_MIIM_CLK Enumeration

C

```
typedef enum {  
    ETH_MIIM_SYSCLK_DIV_BY_4,  
    ETH_MIIM_SYSCLK_DIV_RSVD,  
    ETH_MIIM_SYSCLK_DIV_BY_6,  
    ETH_MIIM_SYSCLK_DIV_BY_8,  
    ETH_MIIM_SYSCLK_DIV_BY_10,  
    ETH_MIIM_SYSCLK_DIV_BY_14,  
    ETH_MIIM_SYSCLK_DIV_BY_20,  
    ETH_MIIM_SYSCLK_DIV_BY_28,  
    ETH_MIIM_SYSCLK_DIV_BY_40  
} ETH_MIIM_CLK;
```

Description

II Clock Selection

This enumeration lists the possible speed selection for RMII. The body contains only two states: RMII_10Mbps or RMII_100Mbps.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.8.5.9.4 ETH_PATTERN_MATCH_MODE Type**C**

```
typedef enum _ETH_PATTERN_MATCH_MODE_ ETH_PATTERN_MATCH_MODE;
```

Description

Pattern Match Modes

This enumeration lists the pattern match mode values.

Remarks

In all pattern match enabled cases, it is the pattern match inversion XOR pattern match checksum AND .

Some states were omitted from the enumeration since they were duplicates.

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.8.5.9.5 ETH_RECEIVE_FILTER Enumeration**C**

```
typedef enum {
    ETH_HASH_FILTER,
    ETH_MAGIC_PACKET_FILTER,
    ETH_PATTERN_MATCH_INVERSION,
    ETH_CRC_ERROR_FILTER,
    ETH_CRC_OK_FILTER,
    ETH_RUNT_ERROR_FILTER,
    ETH_RUNT_ENABLE_FILTER,
    ETH_UNICAST_FILTER,
    ETH_NOT_ME_UNICAST_FILTER,
    ETH_MULTICAST_FILTER,
    ETH_BROADCAST_FILTER,
    ETH_ENABLE_ALL_FILTER
} ETH_RECEIVE_FILTER;
```

Description

ETH Module Receive Filter Mask

This enumeration lists the possible values of the receive filter.

Members

Members	Description
ETH_HASH_FILTER	Enable: Hash Table Filtering.
ETH_MAGIC_PACKET_FILTER	Enable: Magic Packet Filtering.
ETH_PATTERN_MATCH_INVERSION	Enable: Pattern Match Checksum must not match to be accepted.
ETH_CRC_ERROR_FILTER	Enable: Received packet CRC must be invalid to be accepted.
ETH_CRC_OK_FILTER	Enable: Received packet CRC must be valid to be accepted.

ETH_RUNT_ERROR_FILTER	Enable: Received packet must be runt packet to be accepted.
ETH_RUNT_ENABLE_FILTER	Enable: Received packet must not be a runt packet to be accepted.
ETH_UNICAST_FILTER	Enable: Unicast Packets whose destination address matches the station address are accepted.
ETH_NOT_ME_UNICAST_FILTER	Enable: Unicast Packets whose destination address do NOT match the station address are accepted.
ETH_MULTICAST_FILTER	Enable: Multicast Address Packets are accepted.
ETH_BROADCAST_FILTER	Enable: Broadcast Address Packets are accepted.
ETH_ENABLE_ALL_FILTER	Enable: All of the above packets are accepted.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.8.5.9.6 ETH_RMII_SPEED Enumeration**C**

```
typedef enum {
    ETH_RMII_10Mbps,
    ETH_RMII_100Mbps
} ETH_RMII_SPEED;
```

Description

RMII Speed Selection

This enumeration lists the possible speed selection for RMII. The body contains only two states: RMII_10Mbps or RMII_100Mbps.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.8.6 Files**Files**

Name	Description
plib_eth.h	Defines the Ethernet Peripheral Library Interface.

Description**5.6.8.6.1 plib_eth.h**

Ethernet Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Ethernet Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Ethernet module.

Enumerations

	Name	Description
	ETH_AUTOPAD_OPTION	Lists the possible automatic padding configurations.
	ETH_INTERRUPT_SOURCES	Lists the possible values of ETH_INTERRUPT_SOURCES.

	ETH_MIIM_CLK	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).
	ETH_RECEIVE_FILTER	Lists the possible values of the receive filter.
	ETH_RMII_SPEED	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

Functions

	Name	Description
	PLIB_ETH_AlignErrorCountClear	Sets the count of Ethernet alignment errors to zero.
	PLIB_ETH_AlignErrorCountGet	Gets the count of Ethernet alignment errors.
	PLIB_ETH_AutoDetectPadClear	Clears the EMAC Auto-Pad option.
	PLIB_ETH_AutoDetectPadGet	Gets the current EMAC Auto-Pad setting.
	PLIB_ETH_AutoDetectPadSet	Sets the EMAC Auto-Pad option.
	PLIB_ETH_AutoFlowControlDisable	Disables the Ethernet module Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlEnable	Enables the Ethernet Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlsEnabled	Gets the Ethernet module Automatic Flow Control status.
	PLIB_ETH_BackPresNoBackoffDisable	Disables EMAC back pressure/no backoff.
	PLIB_ETH_BackPresNoBackoffEnable	Enables EMAC back pressure/no back-off.
	PLIB_ETH_BackPresNoBackoffIsEnabled	Gets the EMAC back pressure/no back-off enable status.
	PLIB_ETH_BackToBackIPGGet	Gets the EMAC back-to-back interpacket gap.
	PLIB_ETH_BackToBackIPGSet	Sets the EMAC back-to-back interpacket gap.
	PLIB_ETH_CollisionWindowGet	Gets the EMAC collision window.
	PLIB_ETH_CollisionWindowSet	Sets the EMAC collision window.
	PLIB_ETH_CRCDisable	Disables EMAC CRC.
	PLIB_ETH_CRCEnable	Enables EMAC CRC.
	PLIB_ETH_CRCIsEnabled	Gets the EMAC CRC enable status.
	PLIB_ETH_DataNotValid	Gets the MII management read data not valid status.
	PLIB_ETH_DelayedCRCDisable	Disables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCEnable	Enables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCIsEnabled	Gets the EMAC delayed CRC enable status.
	PLIB_ETH_Disable	Disables the Ethernet module.
	PLIB_ETH_Enable	Enables the Ethernet module.
	PLIB_ETH_EthernetIsBusy	Gets the status value of the Ethernet logic busy.
	PLIB_ETH_ExcessDeferDisable	Disables EMAC excess defer.
	PLIB_ETH_ExcessDeferEnable	Enables EMAC excess defer.
	PLIB_ETH_ExcessDeferIsEnabled	Gets the EMAC excess defer enable status.
	PLIB_ETH_ExistsAlignmentErrorCount	Identifies whether the AlignmentErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsAutoFlowControl	Identifies whether the AutoFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionCounts	Identifies whether the CollisionCounts feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionWindow	Identifies whether the CollisionWindow feature exists on the Ethernet module.
	PLIB_ETH_ExistsEnable	Identifies whether the Enable feature exists on the Ethernet module.
	PLIB_ETH_ExistsEthernetControllerStatus	Identifies whether the EthernetControllerStatus feature exists on the Ethernet module.

	<u>PLIB_ETH_ExistsFCSErrorCount</u>	Identifies whether the FCSErrorCount feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsFramesTransmittedOK</u>	Identifies whether the FramesTransmittedOK feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsFrameReceivedOK</u>	Identifies whether the FrameReceivedOK feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsHashTable</u>	Identifies whether the HashTable feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsInterPacketGaps</u>	Identifies whether the InterPacketGaps feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsInterrupt</u>	Identifies whether the Interrupt feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMAC_Configuration</u>	Identifies whether the MAC_Configuration feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMAC_Resets</u>	Identifies whether the MAC_Resets feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMAC_Testing</u>	Identifies whether the MAC_Testing feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsManualFlowControl</u>	Identifies whether the ManualFlowControl feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMaxFrameLength</u>	Identifies whether the MaxFrameLength feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMIIM_Config</u>	Identifies whether the MIIM_Config feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMIIM_Indicators</u>	Identifies whether the MIIM_Indicators feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMIIMAddresses</u>	Identifies whether the MIIMAddresses feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMIIMReadWrite</u>	Identifies whether the MIIMReadWrite feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMIIMScanMode</u>	Identifies whether the MIIMScanMode feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsMIIMWriteReadData</u>	Identifies whether the MIIMWriteReadData feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsPatternMatch</u>	Identifies whether the PatternMatch feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsPauseTimer</u>	Identifies whether the PauseTimer feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsReceiveBufferSize</u>	Identifies whether the ReceiveBufferSize feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsReceiveFilters</u>	Identifies whether the ReceiveFilters feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsReceiveOverflowCount</u>	Identifies whether the ReceiveOverflowCount feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsReceiveWmarks</u>	Identifies whether the ReceiveWmarks feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsRetransmissionMaximum</u>	Identifies whether the RetransmissionMaximum feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsRMII_Support</u>	Identifies whether the RMII_Support feature exists on the Ethernet module.
	<u>PLIB_ETH_ExistsRxBufferCountDecrement</u>	Identifies whether the RxBufferCountDecrement feature exists on the Ethernet module.

	PLIB_ETH_ExistsRxEnable	Identifies whether the ReceiveEnable feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxPacketDescriptorAddress	Identifies whether the RxPacketDescriptorAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStationAddress	Identifies whether the StationAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Ethernet module.
	PLIB_ETH_ExistsTransmitRTS	Identifies whether the TransmitRTS feature exists on the Ethernet module.
	PLIB_ETH_ExistsTxPacketDescriptorAddress	Identifies whether the TxPacketDescriptorAddress feature exists on the Ethernet module.
	PLIB_ETH_FCSErrorCountClear	Sets the value of the Ethernet frame check sequence error to zero.
	PLIB_ETH_FCSErrorCountGet	Gets the count of the frame check sequence error.
	PLIB_ETH_FrameLengthCheckDisable	Disables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckEnable	Enables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckIsEnabled	Gets the EMAC frame length check status.
	PLIB_ETH_FramesRxdOkCountClear	Sets the value of the Ethernet received frames 'OK' count to zero.
	PLIB_ETH_FramesRxdOkCountGet	Gets the count of the frames frames received successfully.
	PLIB_ETH_FramesTxdOkCountClear	Sets the count of Ethernet Control frames transmitted to zero.
	PLIB_ETH_FramesTxdOkCountGet	Gets the count of Ethernet Control Frames transmitted successfully.
	PLIB_ETH_FullDuplexDisable	Disables the EMAC full duplex operatio.
	PLIB_ETH_FullDuplexEnable	Enables the EMAC full deuplex operation.
	PLIB_ETH_FullDuplexIsEnabled	Gets the EMAC full duplex enable status.
	PLIB_ETH_HashTableGet	Gets the value of the Hash table.
	PLIB_ETH_HashTableSet	Sets the Ethernet module Hash table with the new value.
	PLIB_ETH_HugeFrameDisable	Disables the EMAC from receiving huge frames.
	PLIB_ETH_HugeFrameEnable	Enables the EMAC to receive huge frames.
	PLIB_ETH_HugeFrameIsEnabled	Gets the EMAC huge frame enable status.
	PLIB_ETH_InterruptClear	Clears the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptSet	Sets the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptsGet	Gets the Ethernet module Interrupt Flag register as a group.
	PLIB_ETH_InterruptSourceDisable	Clears the Ethernet module Interrupt Enable register as a group, using a mask.
	PLIB_ETH_InterruptSourceEnable	Sets the Ethernet module Interrupt Enable register in a group, using a mask.
	PLIB_ETH_InterruptSourceIsEnabled	Gets the Ethernet module Interrupt Enable register singularly or as a group.
	PLIB_ETH_InterruptSourcesGet	Returns the entire interrupt enable register.
	PLIB_ETH_InterruptStatusGet	Gets the Ethernet module Interrupt Flag register as a group, using a mask.
	PLIB_ETH_IsEnabled	Gets the Ethernet module enable status.
	PLIB_ETH_LinkHasFailed	Gets the MII management link fail status.
	PLIB_ETH_LongPreambleDisable	Disables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleEnable	Enables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleIsEnabled	Gets the EMAC long preamble enforcement enable status.

	PLIB_ETH_LoopbackDisable	Disables the EMAC loopback logic.
	PLIB_ETH_LoopbackEnable	Enables the EMAC loopback logic.
	PLIB_ETH_LoopbackIsEnabled	Gets the EMAC Loopback interface enable status.
	PLIB_ETH_ManualFlowControlDisable	Disable Ethernet module Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlEnable	Enables the Ethernet Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlIsEnabled	Gets the Ethernet module Manual Flow Control enable status.
	PLIB_ETH_MaxFrameLengthGet	Gets the EMAC maximum frame length.
	PLIB_ETH_MaxFrameLengthSet	Sets the EMAC maximum frame length.
	PLIB_ETH_MCSRxResetDisable	Disables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetEnable	Enables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetIsEnabled	Gets the EMAC Control Sub-layer/Receive domain logic reset status.
	PLIB_ETH_MCSTxResetDisable	Disables the reset EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetEnable	Enables the reset of EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetIsEnabled	Gets the EMAC Control Sub-layer/Transmit domain logic reset status.
	PLIB_ETH_MIIMClockGet	Gets the current EMAC MIIM clock selection.
	PLIB_ETH_MIIMClockSet	Sets the EMAC MIM clock selection.
	PLIB_ETH_MIIMIsBusy	Gets the MII management busy status.
	PLIB_ETH_MIIMIsScanning	Gets the MII Management scanning status.
	PLIB_ETH_MIIMNoPreDisable	Disables EMAC No Preamble (allows preamble).
	PLIB_ETH_MIIMNoPreEnable	Enables EMAC MIIM No Preamble (suppresses preamble).
	PLIB_ETH_MIIMNoPreIsEnabled	Gets the EMAC MIIM No Preamble enable status.
	PLIB_ETH_MIIMReadDataGet	Gets EMAC MIIM management read data after a MII read cycle has completed.
	PLIB_ETH_MIIMReadStart	Initiates an MII management read command.
	PLIB_ETH_MIIMResetDisable	Disables EMAC Reset MII Management.
	PLIB_ETH_MIIMResetEnable	Enables EMAC Reset Media Independent Interface (MII) Management.
	PLIB_ETH_MIIMResetIsEnabled	Gets the EMAC Reset MII Management enable status.
	PLIB_ETH_MIIMScanIncrDisable	Disables the EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrEnable	Enables EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrIsEnabled	Gets the EMAC MIIM scan increment enable status.
	PLIB_ETH_MIIMScanModeDisable	Disables MIIM scan mode.
	PLIB_ETH_MIIMScanModeEnable	Enables MIIM scan mode.
	PLIB_ETH_MIIMScanModeIsEnabled	Gets the MII management scan enable status.
	PLIB_ETH_MIIMWriteDataSet	Sets the EMAC MIIM write data before initiating an MII write cycle.
	PLIB_ETH_MIIMWriteStart	Initiates an MII management write command.
	PLIB_ETH_MIIRResetDisable	Disables the EMAC Soft reset.
	PLIB_ETH_MIIRResetEnable	Enables the EMAC MIIM Soft reset.
	PLIB_ETH_MIIRResetIsEnabled	Gets EMAC MIIM Soft Reset enable status.
	PLIB_ETH_MultipleCollisionCountClear	Sets the value of the Ethernet multiple collision frame count to zero.
	PLIB_ETH_MultipleCollisionCountGet	Gets the count of the frames transmitted successfully after there was more than one collision.
	PLIB_ETH_NoBackoffDisable	Disables EMAC no back-offs.
	PLIB_ETH_NoBackoffEnable	Enables EMAC no backoff.

	PLIB_ETH_NoBackoffIsEnabled	Gets the EMAC no back-off enable status.
	PLIB_ETH_NonBackToBackIPG1Get	Gets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG1Set	Sets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG2Get	Gets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_NonBackToBackIPG2Set	Sets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_PassAllDisable	Disables the EMAC PassAll.
	PLIB_ETH_PassAllEnable	Enables the EMAC PassAll.
	PLIB_ETH_PassAllIsEnabled	Gets the EMAC PassAll enable status.
	PLIB_ETH_PatternMatchChecksumGet	Gets the value of the pattern match checksum register.
	PLIB_ETH_PatternMatchChecksumSet	Sets the Ethernet module pattern match checksum register with the new value.
	PLIB_ETH_PatternMatchGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeSet	Sets the Ethernet module pattern match mode.
	PLIB_ETH_PatternMatchOffsetGet	Gets the value of the patter match offset register.
	PLIB_ETH_PatternMatchOffsetSet	Sets the Ethernet module patter match offset register with the new value.
	PLIB_ETH_PatternMatchSet	Sets the Ethernet module pattern match mask register with the new value.
	PLIB_ETH_PauseTimerGet	Gets the Pause Timer value used for flow control.
	PLIB_ETH_PauseTimerSet	Sets the Pause Timer value used for flow control.
	PLIB_ETH_PHYAddressGet	Gets the EMAC MIIM management PHY address.
	PLIB_ETH_PHYAddressSet	Sets the EMAC MIIM PHY address.
	PLIB_ETH_PurePreambleDisable	Disables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleEnable	Enables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleIsEnabled	Gets EMAC pure preamble enforcement enable status.
	PLIB_ETH_ReceiveBufferSizeGet	Gets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveBufferSizeSet	Sets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveDisable	Disables the EMAC from receiving frames.
	PLIB_ETH_ReceiveEnable	Enables the EMAC to receive the frames.
	PLIB_ETH_ReceiveFilterDisable	Disables the specified receive filter.
	PLIB_ETH_ReceiveFilterEnable	Enables the specified receive filter.
	PLIB_ETH_ReceiveFilterIsEnable	Disables the specified receive filter.
	PLIB_ETH_ReceiveIsBusy	Gets the the Ethernet receive logic busy status.
	PLIB_ETH_ReceiveIsEnabled	Gets the Ethernet EMAC receive enable status.
	PLIB_ETH_RegisterAddressGet	Gets the EMAC MIIM management register address.
	PLIB_ETH_RegisterAddressSet	Sets EMAC MIIM register address.
	PLIB_ETH_ReTxMaxGet	Gets the EMAC maximum retransmissions.
	PLIB_ETH_ReTxMaxSet	Sets the EMAC maximum retransmissions.
	PLIB_ETH_RMIIResetDisable	Disables EMAC Reset RMII.
	PLIB_ETH_RMIIResetEnable	Enables EMAC Reset RMII.
	PLIB_ETH_RMIIResetIsEnabled	Gets the EMAC Reset RMII enable status.
	PLIB_ETH_RMII SpeedGet	Gets the current EMAC RMII speed.
	PLIB_ETH_RMII SpeedSet	Sets EMAC RMII Speed.
	PLIB_ETH_RxBufferCountDecrement	Causes the Receive Descriptor Buffer Counter to decrement by 1.
	PLIB_ETH_RxDisable	Disables the Ethernet module receive logic.

	PLIB_ETH_RxEmptyWmarkGet	Gets the Ethernet module receive empty watermark.
	PLIB_ETH_RxEmptyWmarkSet	Sets the Ethernet module receive empty water mark.
	PLIB_ETH_RxEnable	Enables the Ethernet receive logic.
	PLIB_ETH_RxFullWmarkGet	Gets the Ethernet module receive full watermark.
	PLIB_ETH_RxFullWmarkSet	Sets the Ethernet module receive full water mark.
	PLIB_ETH_RxFuncResetDisable	Disables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetEnable	Enables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetIsEnabled	Gets the EMAC reset receive function status.
	PLIB_ETH_RxIsEnabled	Gets the Ethernet module receive enable status.
	PLIB_ETH_RxOverflowCountClear	Sets the value of the Ethernet receive overflow count to zero.
	PLIB_ETH_RxOverflowCountGet	Gets the count of the dropped Ethernet receive frames.
	PLIB_ETH_RxPacketCountGet	Gets the value of the receive packet buffer count.
	PLIB_ETH_RxPacketDescAddrGet	Gets the address of the next receive descriptor.
	PLIB_ETH_RxPacketDescAddrSet	Sets the Ethernet module receive packet descriptor start address.
	PLIB_ETH_RxPauseDisable	Disables the EMAC receive flow control.
	PLIB_ETH_RxPauseEnable	Enables the EMAC receive flow control.
	PLIB_ETH_RxPauseIsEnabled	Gets the EMAC receive flow pause enable status.
	PLIB_ETH_ShortcutQuantaDisable	Disables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaEnable	Enables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaIsEnabled	Gets EMAC shortcut pause quanta enable status.
	PLIB_ETH_SimResetDisable	Disables the EMAC simulation reset.
	PLIB_ETH_SimResetEnable	Enables the EMAC simulation reset.
	PLIB_ETH_SimResetIsEnabled	Gets the EMAC simulation reset status.
	PLIB_ETH_SingleCollisionCountClear	Sets the value of the Ethernet single collision frame count to zero.
	PLIB_ETH_SingleCollisionCountGet	Gets the count of the frames transmitted successfully on a second attempt.
	PLIB_ETH_StationAddressGet	Gets the selected EMAC station address.
	PLIB_ETH_StationAddressSet	Sets the selected EMAC Station Address.
	PLIB_ETH_StopInIdleDisable	Ethernet module operation will continue when the device enters Idle mode.
	PLIB_ETH_StopInIdleEnable	Ethernet module operation will stop when the device enters Idle mode.
	PLIB_ETH_StopInIdleIsEnabled	Gets the Ethernet module Stop in Idle mode enable status.
	PLIB_ETH_TestBackPressDisable	Disables EMAC Test backpressure.
	PLIB_ETH_TestBackPressEnable	Enables EMAC Test backpressure.
	PLIB_ETH_TestBackPressIsEnabled	Gets the EMAC test backpressure enable status.
	PLIB_ETH_TestPauseDisable	Disables EMAC test pause.
	PLIB_ETH_TestPauseEnable	Enables EMAC test pause.
	PLIB_ETH_TestPauseIsEnabled	Gets the EMAC test pause enable status.
	PLIB_ETH_TransmitIsBusy	Gets the status value of the Ethernet transmit logic busy status
	PLIB_ETH_TxFuncResetDisable	Disables the EMAC Transmit function reset.
	PLIB_ETH_TxFuncResetEnable	Enables the EMAC transmit function reset.
	PLIB_ETH_TxFuncResetIsEnabled	Gets the EMAC Transmit function reset status.
	PLIB_ETH_TxPacketDescAddrGet	Gets the address of the next descriptor to be transmitted.
	PLIB_ETH_TxPacketDescAddrSet	Sets the Ethernet module transmit packet descriptor start address.
	PLIB_ETH_TxPauseDisable	Disables the transmission of Pause frames.

	PLIB_ETH_TxPauseEnable	Enables the the tansmission Pause frames.
	PLIB_ETH_TxPausesIsEnabled	Gets the Ethernet module enable status.
	PLIB_ETH_TxRTSDisable	Aborts an Ethernet module transmission.
	PLIB_ETH_TxRTSEnable	Enables the Ethernet transmit request to send .
	PLIB_ETH_TxRTSIsEnabled	Gets the Ethernet module transmit request to send status.

Types

	Name	Description
	ETH_PATTERN_MATCH_MODE	Lists the possible pattern match values.

File Name

plib_eth.h

Company

Microchip Technology Inc.

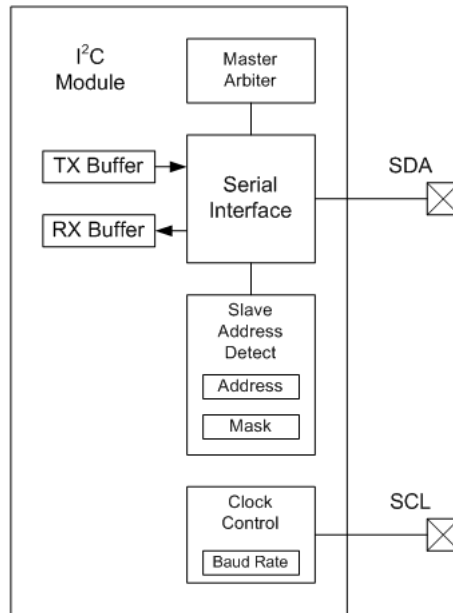
5.6.9 I2C Peripheral Library

5.6.9.1 Introduction

This library provides a low-level abstraction of the Inter-Integrated Circuit (I²C) module on Microchip microcontrollers.

Description

I²C Peripheral Library
for
Microchip Microcontrollers



This library provides a low-level abstraction of the Inter-Integrated Circuit (I^2C) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Please refer to the I^2C -Bus Specification (v2.1), available from Philips Semiconductor (see the **Note** below) for complete details on the operation and requirements for the I^2C bus.

Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software.

5.6.9.2 Release Notes

MPLAB Harmony Version: v0.70b **I^2C Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.9.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO

YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.9.4 How the Library Works

How the Library Works

Before enabling the I²C module the caller must initialize basic configuration, clock frequency, and Slave address recognition features (see [Initializing the I2C](#)).

After the module has been enabled, it can begin monitoring the bus as a slave device waiting to be addressed by an external master (see [Operating as a Slave Receiver](#)). A slave device only transfers data on the bus when it has been addressed by a master (see [Operating as a Slave Transmitter](#)). If the module is used to start a transfer, then it is operating as a master. A master addresses a slave and controls the transfer of data by providing the clock (see [Operating as a Master Transmitter](#) and [Operating as a Master Receiver](#)).

Some operations in the I²C library initiate actions on the I²C bus that require time to complete. Also, some events occur asynchronously on the bus. In each of these cases, the module causes either a "master", "slave", or "error" interrupt. [Interrupt State Machine](#) describes the different states that can cause an interrupt and show what causes the transition from one state to another. [Handling Errors](#) describes the various errors that can occur and how to deal with them.

Data transfers on the I²C bus must follow specific formats defined by the I²C bus protocol. (Refer to [Forming Transfers](#) for details)

Note: The I²C peripheral library does not directly respond to interrupts. The client software (usually a driver, middleware layer, or application) should implement the Interrupt Service Routine (ISR) and call the I²C library's interface routines from that ISR to manage the state of the I²C module.

5.6.9.4.1 Initializing the I2C

To initialize the I²C module, perform the following sequence:

1. Select the desired general configuration options using General Initialization Functions to select the desired operation of the following features:
 - Slave Clock Stretching
 - General Call Address Recognition
 - System Management Bus (SMBus) Support
 - High Frequency Operation
 - Reserved Address Protection
 - Stop-in-Idle Operation
2. Program the Baud Rate Generator to set the desired baud rate using the [PLIB_I2C_BaudRateSet](#) function.
3. If operating in slave mode, set the desired slave address using the [PLIB_I2C_SlaveAddress7BitSet](#) function (for 7-bit mode operation) or [PLIB_I2C_SlaveAddress10BitSet](#) function (for 10-bit mode operation).
4. If running in an interrupt-driven environment, clear any pending interrupts and enable the appropriate (master, slave, and error) I²C interrupts. Refer to the **"Interrupt Controller"** section in the related family reference manual section for information.

5. Enable the module for operation using the [PLIB_I2C_Enable](#) function.

Example

```
#define MY_I2C_ID I2C_ID_1
uint32_t actualClock;

// Configure General I2C Options
PLIB_I2C_SlaveClockStretchingEnable(MY_I2C_ID);
PLIB_I2C_GeneralCallEnable(MY_I2C_ID);
PLIB_I2C_SMBDisable(MY_I2C_ID);
PLIB_I2C_HighFrequencyDisable(MY_I2C_ID);
PLIB_I2C_ReservedAddressProtectEnable(MY_I2C_ID);
PLIB_I2C_StopInIdleEnable(MY_I2C_ID);

// Set Desired baud rate
PLIB_I2C_BaudRateSet ( MY_I2C_ID, MY_PERIPHEAL_CLOCK_FREQ, MY_I2C_BAUD_RATE);

// Set the appropriate slave address (slave only)
PLIB_I2C_SlaveAddress7BitSet(MY_I2C_ID, MY_SLAVE_ADDRESS);

// Optional: Clear and enable interrupts before enabling the I2C module.

// Enable the module
PLIB_I2C_Enable(MY_I2C_ID);
```

Note: Refer to the "Interrupt Controller" section in the related family reference manual section for details on how to clear and enable the I²C module interrupts.

The previous example uses the following constants:

Symbol	Description
I2C_ID_1	Defined as the selected I ² C module value from the I2C Module enumeration . Example: #define MY_I2C_ID I2C_ID_1
MY_I2C_PERIPHERAL_CLOCK	Defined as the peripheral clock frequency (in Hz) of the I ² C module. Example: #define MY_I2C_PERIPHERAL_CLOCK 80000000
MY_I2C_BAUD_RATE	Defined as desired I ² C baud rate. Example: #define MY_I2C_BAUD_RATE 100000
MY_SLAVE_ADDRESS	Defined as the desired 7-bit I ² C slave address. Example #define 0x1A // Slave address 0011010 in binary Note: If 0 is passed in as the mask value, the I ² C module will only respond to single slave address passed in the second parameter.

5.6.9.4.2 Operating as a Slave Receiver

Before enabling the module, the module's "local" slave address and mask should be programmed using the [PLIB_I2C_SlaveAddress7BitSet](#) function or the [PLIB_I2C_SlaveAddress10BitSet](#) function, depending on the desired address width. If slave address masking is supported on the chosen processor the "don't care bits" mask can be set using the [PLIB_I2C_SlaveMask7BitSet](#) function or the [PLIB_I2C_SlaveMask10BitSet](#) function, also depending on the desired address width. Once the I²C module has been enabled, it will begin waiting for an I²C master to address it. When the programmed local slave address is recognized (ignoring any bits in the programmed mask value, if supported), the module will ACK the address and notify software that it has received an address byte. If the R/W bit of the address is 0, the module enters receive mode automatically. As additional bytes are received, the module will automatically ACK them if the previous byte has been read from the RX buffer before the new byte has been completely received and notify software of the newly available byte. Both address and data bytes received must be read from the RX buffer by software.

Preconditions

1. The module must have had it's basic options and clock frequency initialized and have been enabled (see [Initializing the I2C](#)).
2. An external master must have started a transfer.
3. The address address of the transfer must have matched with the address (ignoring any bits in the mask) programmed into the module by software.

Process

Software must follow this sequence:

1. Ensure that a byte has been received either by polling the [PLIB_I2C_ReceivedByteIsAvailable](#) function or by waiting for a slave interrupt (at which time, this function should still be checked).
2. Identify if the byte is a data byte or an address byte using the [PLIB_I2C_SlaveAddressIsDetected](#) function.
3. If the byte is an address byte, software must identify if it is starting a read or write operation using the [PLIB_I2C_SlaveReadIsRequested](#) function.
4. Software must read the byte from the RX buffer using the [PLIB_I2C_ReceivedByteGet](#) function, even if it was an address byte.
5. If clock stretching was enabled during initialization, then software must call the [PLIB_I2C_SlaveClockRelease](#) function to release the SCL line.

Example

```
#define MY_I2C_ID  I2C_ID_1
uint8_t data;

if ( PLIB_I2C_ReceivedByteIsAvailable(MY_I2C_ID) )
{
    // Check to see if the byte is data or an address
    if (PLIB_I2C_SlaveAddressWasDetected(MY_I2C_ID))
    {
        if ( PLIB_I2C_SlaveReadWasRequested(MY_I2C_ID) )
        {
            // Beginning a slave receive sequence
        }
        else
        {
            // Beginning a slave transmit sequence
        }
    }
    else
    {
        // Data Byte Received
    }
    // Read the data or address from the RX buffer
    data = PLIB_I2C_ReceivedByteGet(MY_I2C_ID);

    // Free the SCL line (only if clock stretching was enabled)
    PLIB_I2C_SlaveClockRelease(MY_I2C_ID);
}
```

The above example uses the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I ² C module value from the I2C Module enumeration . Example: #define MY_I2C_ID I2C_ID_1
data	This is the variable used to hold the byte received.

5.6.9.4.3 Operating as a Slave Transmitter

Once an an external master has addressed the the I²C module (see [Operating as a Slave Receiver](#)) with the read-write bit set

(as determined by calling the [PLIB_I2C_SlaveReadIsRequested](#) function), software can begin transmitting data to the master.

Pre-Conditions

1. The module must have had it's basic options and clock frequency initialized and have been enabled(see [Initializing the I2C](#))
2. An external master must have started a transfer.
3. The address address of the transfer must have matched with the address slave programmed into the I²C module (see [Managing Slave Addresses](#)).
4. A slave read must have been requested (as described above).

Note: When the address has been received and matched the programmed local slave address, a slave interrupt will occur if it has been enabled.

Transmitting a Byte

When operating as a slave transmitter on the I²C bus, the software must use the following sequence to transmit data bytes.

1. Ensure that the TX buffer is currently ready to [accept](#) a byte to be transmitted using the [PLIB_I2C_TransmitterIsReady](#) function.
2. Write the byte to be transmitted to the TX buffer using the [PLIB_I2C_TransmitterByteSend](#) function.
3. If clock stretching was enabled (using the [PLIB_I2C_SlaveClockStretchingEnable](#) function) and the previous byte was ACKed, the [PLIB_I2C_SlaveClockRelease](#) function must be called to release the SCL line (it is safe to call this even if the previous byte was NAKed.)

Example

```
#define MY_I2C_ID  I2C_ID_1
uint8_t          data;

// Assign the desired byte value into the "data" variable

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    // Write the byte to the TX buffer
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, data);

    // Free the SCL line (only if clock stretching was enabled)
    PLIB_I2C_SlaveClockRelease(MY_I2C_ID);
}
```

Checking Acknowledgement

After a byte has been transmitted, the module latches the ACK/NAK response from the master so that software may use the following sequence to verify it and causes another slave interrupt (if it has been enabled).

1. Ensure that the byte has been completely transmitted using the [PLIB_I2C_TransmitterByteHasCompleted](#) function (and/or by responding to the slave interrupt).
2. Determine if the byte was ACKed or NAKed using the [PLIB_I2C_TransmitterByteWasAcknowledged](#) function.

Example

```
#define MY_I2C_ID  I2C_ID_1

if (PLIB_I2C_TransmitterByteHasCompleted(MY_I2C_ID))
{
    if (PLIB_I2C_TransmitterByteWasAcknowledged(MY_I2C_ID))
    {
        // Transmission successful
    }
    else
    {
        // Transmission was not successful
    }
}
```



```
}

```

The above examples use the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I ² C module value from the I2C_Module_enumeration . Example: #define MY_I2C_ID I2C_ID_1
data	This is the buffer containing byte of data to transmit.

5.6.9.4.4 Operating as a Master Transmitter

Whenever the I²C module initiates a transfer on the bus, it is acting as master. A transfer can be initiated any time the bus is idle. If two masters attempt to initiate transfers at nearly the same time, one of them will lose arbitration at some point during the transfer and must retry the transfer later when the bus becomes idle. The winner of arbitration can continue, without any data loss.

An I²C bus transfer always begins with a Start condition, followed by one or two properly formatted bytes containing the 7-bit or 10-bit target slave address (address of the slave device that is the target of the transfer). The direction of the remainder of transfers (after the address has been sent) is indicated by bit 0 of the address (0 = write/transmitting, 1 = read/receiving, from the point of view of the master). After that, any amount of data may be transferred and a repeated Start condition (followed by another properly formatted address) can occur to change the direction of the transfer or even the target slave device. See [Forming Transfers](#) for details of the possible transfer formats. When the master is finished, an I²C bus transfer always ends with a Stop condition. These steps are summarized below.

Summary of Steps

1. Send a Start condition.
2. Send a 7-bit or 10-bit address (one or two bytes with a write transfer indication in bit 0).
3. Send data bytes.
4. Send a Stop condition.

Each of these steps making up a transfer will take some time to complete. The completion of each step will cause an I²C master interrupt. The software must always keep track of which state it was in, check the I²C status, and respond to the interrupt by performing the next appropriate step.

Note 1: Arbitration loss can occur after each step has completed. Software should check for arbitration loss by calling the [PLIB_I2C_ArbitrationLossHasOccurred](#) function. If arbitration loss occurs, it must be cleared by calling the [PLIB_I2C_ArbitrationLossClear](#) function.

Note 2: After any step completes, but before sending the Stop condition (see step 4), the master may [send](#) a repeated Start condition and start over at step 2 to change slave targets and/or transfer direction.

Sending a Start Condition

To start a master transfer, the software must do the following:

Preconditions

The module must have had its basic options and clock frequency initialized and have been enabled (see [Initializing the I2C](#)).

Process

1. Verify that the bus is idle using the [PLIB_I2C_BusIsIdle](#) function.
2. Send the start signal using the [PLIB_I2C_MasterStart](#) function.
3. Check for arbitration loss after the start condition has completed using the [PLIB_I2C_ArbitrationLossHasOccurred](#) function.

Example: Sending a Start Condition

```
#define MY_I2C_ID  I2C_ID_1

if (PLIB_I2C_BusIsIdle(MY_I2C_ID))
{
    PLIB_I2C_MasterStart(MY_I2C_ID);
}
```

The Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated. Even if the bus is verified to be idle before sending the start condition, a bus collision can still occur while the Start condition is being transmitted. To verify that arbitration has not been lost during transmission of the Start condition, the software should check the [PLIB_I2C_ArbitrationLossHasOccurred](#) function after the Start condition has completed.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID  I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    // Abort transfer, retry later

    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

Sending a 7-bit Address

To [send](#) a 7-bit address, the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Ensure that the TX buffer is currently ready to [accept](#) a byte to be transmitted using the [PLIB_I2C_TransmitterIsReady](#) function.
2. Format the address byte correctly with the 7-bit address and the read/write bit appropriately set (for a read transfer) or cleared (for a write transfer). This can be done easily using the `PLIB_I2C_ADDRESS_7_BIT_FORMAT` macro and the `I2C_READ` or `I2C_WRITE` constants.
3. Write the address byte to the TX buffer using the [PLIB_I2C_TransmitterByteSend](#) function (the address byte can be easily accessed using the `PLIB_I2C_ADDRESS_7_BIT_BYTE_ACCESS` macro)

Note: A master interrupt will occur once the address byte has been transmitted and ACKed or NAKed by the targeted slave device. (If the slave device doesn't answer, an implicit NAK will occur.)

4. Once the address byte has been ACKed or NAKed by the slave target, software must verify the result and respond appropriately. The Acknowledgement and Arbitration process is described in a subsequent section, just before the description of how to [send](#) a Stop condition.)

Example: Sending a 7-bit Address

```
#define MY_I2C_ID  I2C_ID_1
#define SLAVE_TARGET_7BIT_ADDRESS 23
I2C_7_BIT_ADDRESS slaveTargetAddress7Bit;

// Format the 7-bit address byte with the read or write indication in bit 0
I2C_ADDRESS_7_BIT_FORMAT(slaveTargetAddress7Bit, SLAVE_TARGET_7BIT_ADDRESS,
    I2C_WRITE);

// Send the address byte
if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID,
```

```
I2C_ADDRESS_7_BIT_BYTE_ACCESS(slaveTargetAddress7Bit));
}
```

Note: Bits 7-1 contain the 7-bit target slave address. Bit 0 indicates read or write direction for the remainder of the transfer.

Sending a 10-bit Target Slave Address

To **send** a 10-bit address, the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Format the two address bytes correctly with the 10-bit address and the read/write bit appropriately set (for a read transfer) or cleared (for a write transfer). This can be done using the `I2C_ADDRESS_10_BIT_FORMAT` macro and the `I2C_READ` or `I2C_WRITE` constants.
2. Ensure that the TX buffer is currently ready to **accept** a byte to be transmitted using the `PLIB_I2C_TransmitterIsReady` function.
3. Write the first address byte to the TX buffer using the `PLIB_I2C_TransmitterByteSend` function (the first address byte can be easily accessed using the `PLIB_I2C_ADDRESS_10_BIT_1ST_BYTE_ACCESS` macro)

Note: After the first byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or you can poll the `PLIB_I2C_TransmitterByteHasCompleted` function).

4. Ensure that the first address byte was ACKed by the slave target (to do this, follow the Acknowledgement process described further below)
5. Write the second address byte to the TX buffer using the `PLIB_I2C_TransmitterIsReady` function and the `PLIB_I2C_TransmitterByteSend` function (the second address byte can be easily accessed using the `PLIB_I2C_ADDRESS_10_BIT_2ND_BYTE_ACCESS` macro).

Note: After the second byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or you can poll the `PLIB_I2C_TransmitterByteHasCompleted` function).

6. Ensure that the second address byte was ACKed by the slave target (to do this, follow the process in the Acknowledgement and Arbitration, which is described later in this section).

Example: Formatting the 10-bit Address in Two Bytes

```
#define MY_I2C_ID I2C_ID_1
#define SLAVE_TARGET_10BIT_ADDRESS 0x23

I2C_10_BIT_ADDRESS slaveAddress10Bit;

I2C_ADDRESS_10_BIT_FORMAT(slaveAddress10Bit, SLAVE_TARGET_10BIT_ADDRESS, I2C_WRITE);
```

Note: The `slaveAddress10Bit` variable is used in the following two examples.

Example: Sending the First Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID,
    I2C_ADDRESS_10_BIT_1ST_BYTE_ACCESS(slaveAddress10Bit));
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Example: Sending the Second Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
```

```
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID,
    I2C_ADDRESS_10_BIT_2ND_BYTE_ACCESS(slaveAddress10Bit));
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Sending a Data Byte

The following sequence can be used to [send](#) a data byte and repeated to [send](#) an arbitrary number of data bytes.

Preconditions

- The Start condition must have been sent and completed as previously described
- The target slave address has been fully transmitted (with the write indication in bit 0) and ACKed by the targeted slave device

Process

1. Ensure that the TX buffer is currently ready to [accept](#) a byte to be transmitted using the [PLIB_I2C_TransmitterIsReady Function](#).
2. Write the byte to be transmitted to the TX buffer using the [PLIB_I2C_TransmitterByteSend Function](#).

Note: After the byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or poll the [PLIB_I2C_TransmitterByteHasCompleted](#) function).

3. Check for arbitration loss or an ACK or NAK from the slave as described in the next section, Acknowledgement and Arbitration.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t          data;

// Assign the data value into the "data" variable

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, data);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Acknowledgement and Arbitration

After sending a data or address byte (as described above), the master transmitter must check to see if arbitration was lost and if the byte was ACK'd or NAK'd. If the byte was ACK'd, the slave receiver is ready to receive more bytes. If the byte was NAKed, the slave receiver cannot [accept](#) any more bytes at this time.

Preconditions

- The start condition must have been sent and completed as previously described
- A byte (address or data) must have been transmitted, using one of the the previous processes

Process

1. Ensure that the byte has been completely transmitted using the [PLIB_I2C_TransmitterByteHasCompleted](#) function (and/or by waiting for the master interrupt)
2. Determine if arbitration loss has occurred using the [PLIB_I2C_ArbitrationLossHasOccurred](#) function.
3. Determine if the byte was ACKed or NAKed using the [PLIB_I2C_TransmitterByteWasAcknowledged](#) function.

Example

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterByteHasCompleted(MY_I2C_ID))
```

```

{
    if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
    {
        // Handle arbitration loss

        PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
    }
    else
    {
        if (PLIB_I2C_TransmitterByteWasAcknowledged(MY_I2C_ID))
        {
            // Transmission successful
        }
        else
        {
            // Transmission was not successful
        }
    }
}

```

To handle arbitration loss, the master transmitter must stop the transfer process and retry the entire transfer again at a later time when the bus is not busy.

Sending a Repeated Start Condition

If the master wants to change slave targets or transfer directions, it can [send](#) a repeated Start condition followed by a new address at any time after the previous target slave address has been acknowledged (and before the Stop condition has been sent).

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed
- Zero or more data bytes may also have been sent and ACKed

Process

Send the repeated Start condition using the [PLIB_I2C_MasterStartRepeat](#) function.

Example

```
#define MY_I2C_ID I2C_ID_1
```

```
PLIB_I2C_MasterStartRepeat(MY_I2C_ID);
```

The Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated.

The repeated Start must be followed by another target slave address (either 7-bit or 10-bit, as appropriate).

Sending a Stop Condition

When the master transmitter has finished sending data, it must end the transfer with a stop condition.

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed
- Zero or more data bytes may also have been sent and ACKed

Note: At least one address byte must be transmitted between the Start and Stop conditions. Following a Start condition immediately with a Stop condition is a protocol error and can cause some slave devices to lock up and stop responding to the bus.

Process

Send a stop condition using the [PLIB_I2C_MasterStop](#) function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStop(MY_I2C_ID);
```

The Stop condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Stop condition has been initiated.

After a Stop condition, the bus should be idle unless a master immediately starts a new transfer. Strictly speaking, a bus collision can still occur while the Stop condition is being transmitted. Although the transfer is already completed when the Stop condition has been sent, software does not have to abort the transfer; however, it should still check for and clear the arbitration loss condition if it occurs.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

The previous examples use the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I ² C module value from the I2C Module enumeration . Example: <code>#define MY_I2C_ID I2C_ID_1</code>
data	This is the buffer containing byte of data to transmit.
slaveTargetAddress7Bit	This variable is used to demonstrate how to properly format and access a 7-bit target slave address
SLAVE_TARGET_7BIT_ADDRESS	This constant is used as a placeholder for the actual 7-bit target slave address that must be defined by the client code.
slaveAddress10Bit	This variable is used to demonstrate how to properly format and access a 10-bit slave address.
SLAVE_TARGET_10BIT_ADDRESS	This constant is used as a placeholder for the actual 10-bit slave address that must be defined by the client code.

5.6.9.4.5 Operating as a Master Receiver

Any time the I²C module initiates a transfer on the bus, it is acting as a master. A transfer can be initiated any time the bus is idle. If two masters attempt to initiate transfers at nearly the same time, one of them will lose arbitration and must retry the transfer later when the bus becomes idle. The winner of arbitration can continue, without any data loss.

An I²C bus transfer always begins with a Start condition, followed by a 7-bit or 10-bit (1-byte or 2-byte) target slave address. Therefore, every master transfer starts off transmitting at least one byte. The direction of transfer of subsequent bytes (after the first byte has been sent) is indicated by bit 0 of the first address byte (0 = write/transmitting, 1 = read/receiving, from the point of view of the master). If 10-bit addressing is used or additional data (such as an internal address) is to be transmitted after the first address byte, the first address byte must indicate a write transfer. To change the direction after transmitting more than just a single address byte, a repeated Start condition (followed by another single address byte) must be sent (see [Forming Transfers](#) for details of the possible transfer formats). When the master is finished, an I²C bus transfer always ends with a Stop condition. This results in two basic formats for receiving data as a master, as summarized below.

Summary of Steps: Read Only

1. Send a Start condition.
2. Send a 7-bit address with a read transfer indication in bit 0.
3. Receive data bytes.
4. Send a Stop condition.

Summary of Steps: Combined Read-Write

1. Send a Start condition.
2. Send a 7-bit address (or the first byte of a 10-bit address) with a write transfer indication in bit 0.
3. Send the second byte of a 10-bit address (and/or any additional data to be transmitted).
4. Send a repeated Start condition.
5. Resend the first byte of the 10-bit address (or a 7-bit address) with a read indication in bit 0.
6. Receive data bytes.
7. Send a Stop condition.

Each of these steps making up a transfer will take some time to complete. The completion of each step will cause an I²C master interrupt. The software must always keep track of which state it was in and respond to the interrupt by performing the next appropriate step.

Sending a Start Condition

To start a master transfer, the software must do the following:

Preconditions

The module must have had its basic options and clock frequency initialized and have been enabled (see Initializing the I2C).

Process

1. Verify that the bus is idle using the [PLIB_I2C_BusIsIdle](#) function.
2. Send the start signal using the [PLIB_I2C_MasterStart](#) function.

Example

```
#define MY_I2C_ID  I2C_ID_1
if (PLIB_I2C_BusIsIdle(MY_I2C_ID))
{
    PLIB_I2C_MasterStart(MY_I2C_ID);
}
```

The Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated. Even if the bus is verified to be idle before sending the Start condition, a bus collision can still occur while the Start condition is being transmitted. To verify that arbitration has not been lost during transmission of the Start condition, the software should check the [PLIB_I2C_ArbitrationLossHasOccurred](#) function after the Start condition has completed.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID  I2C_ID_1
if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    // Abort transfer, retry later
    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

Sending a 7-bit Address

To **send** a 7-bit address (or the first byte of a 10-bit address), the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Ensure that the TX buffer is currently ready to **accept** a byte to be transmitted using the [PLIB_I2C_TransmitterIsReady](#) function.
2. Format the address byte correctly with the 7-bit address and the read/write bit appropriately set (for a read transfer) or cleared (for a write transfer). This can be done easily using the `PLIB_I2C_ADDRESS_7_BIT_FORMAT` macro. and the `I2C_READ` or `I2C_WRITE` constants.
3. Write the address byte to the TX buffer using the [PLIB_I2C_TransmitterByteSend](#) function (the address byte can be easily accessed using the `I2C_ADDRESS_7_BIT_BYTE_ACCESS` macro).

Note: A master interrupt will occur once the address byte has been transmitted and ACKed or NAKed by the targeted slave device (or if arbitration was lost). If the slave device does not answer, an implicit NAK will occur.

4. Once the address byte has been ACKed or NAKed by the slave target (or if arbitration was lost), software must verify the result and respond appropriately. The Acknowledgement and Arbitration process is described in a subsequent section, just before the description of how to **send** a Stop condition.)

Example: Sending a 7-bit Address

```
#define MY_I2C_ID    I2C_ID_1
I2C_7_BIT_ADDRESS  slaveTargetAddress7Bit;

// Format the 7-bit address byte with the read or write indication in bit 0
I2C_ADDRESS_7_BIT_FORMAT(slaveTargetAddress7Bit, SLAVE_TARGET_7BIT_ADDRESS, I2C_WRITE);

// Send the address byte
if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID,
    I2C_ADDRESS_7_BIT_BYTE_ACCESS(slaveTargetAddress7Bit));
}
```

Note: Bits 7-1 contain the 7-bit target slave address. Bit 0 indicates read or write direction for the remainder of the transfer.

Sending a 10-bit Target Slave Address

To **send** a 10-bit address, the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Format the two address bytes correctly with the 10-bit address and the read/write bit cleared, indicating a write transfer. This can be done easily using the `PLIB_I2C_ADDRESS_10_BIT_FORMAT` macro and the `I2C_WRITE` constant.
2. Ensure that the TX buffer is currently ready to **accept** a byte to be transmitted using the [PLIB_I2C_TransmitterIsReady](#) function.
3. Write the first address byte to the TX buffer using the [PLIB_I2C_TransmitterByteSend](#) function (the first address byte can be easily accessed using the `I2C_ADDRESS_10_BIT_1ST_BYTE_ACCESS` Macro)

Note: After the first byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or poll the [PLIB_I2C_TransmitterByteHasCompleted](#) function).

4. Ensure that the first address byte was ACKed by the slave target and check for arbitration loss. To do this, use the Acknowledgement and Arbitration process, which described in the next section.
5. Write the second address byte to the TX buffer using the [PLIB_I2C_TransmitterIsReady](#) function and the [PLIB_I2C_TransmitterByteSend](#) function (the second address byte can be easily accessed using the `I2C_ADDRESS_10_BIT_2ND_BYTE_ACCESS` macro).

Note: After the second byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or you can poll the [PLIB_I2C_TransmitterByteHasCompleted](#) function).

6. Ensure that the second address byte was ACKed by the slave target and that arbitration was not lost. To do this, use the Acknowledgement and Arbitration process, which described in the next section.

Example: Formatting the 10-bit Address in Two Bytes

```
#define MY_I2C_ID I2C_ID_1
#define SLAVE_TARGET_10BIT_ADDRESS 0x23

I2C_10_BIT_ADDRESS slaveAddress10Bit;
I2C_ADDRESS_10_BIT_FORMAT(slaveAddress10Bit, SLAVE_TARGET_10BIT_ADDRESS, I2C_WRITE);
```

Note: The slaveAddress10Bit variable is used in the following two examples.

Example: Sending the First Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1
if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID,
    I2C_ADDRESS_10_BIT_1ST_BYTE_ACCESS(slaveAddress10Bit));
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Example: Sending the Second Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID,
    I2C_ADDRESS_10_BIT_2ND_BYTE_ACCESS(slaveAddress10Bit));
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Acknowledgement and Arbitration

After sending a data or address byte (as described above), the master transmitter must check to see if arbitration was lost and if the byte was ACKed or NAKed. If the byte was ACKed, the slave receiver is ready to receive more bytes. If the byte was NAKed, the slave receiver cannot [accept](#) any more bytes at this time.

Preconditions

- The Start condition must have been sent and completed as previously described
- A byte (address or data) must have been transmitted, using one of the the previous processes

Process

1. Ensure that the byte has been completely transmitted using the [PLIB_I2C_TransmitterByteHasCompleted](#) function (and/or by waiting for the master interrupt).
2. Determine if arbitration loss has occurred using the [PLIB_I2C_ArbitrationLossHasOccurred](#) function.
3. Determine if the byte was ACKed or NAKed using the [PLIB_I2C_TransmitterByteWasAcknowledged](#) function.

```
#define MY_I2C_ID I2C_ID_1
if (PLIB_I2C_TransmitterByteHasCompleted(MY_I2C_ID))
{
    if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
    {
        // Handle arbitration loss
        PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
    }
    else
```

```

    {
        if (PLIB_I2C_TransmitterByteWasAcknowledged(MY_I2C_ID))
        {
            // Transmission successful
        }
        else
        {
            // Transmission was not successful
        }
    }
}

```

To handle arbitration loss, the master must stop the transfer process and retry the entire transfer again at a later time when the bus is not busy.

Sending a Repeated Start Condition

If the master wants to change slave targets or transfer directions, it must **send** a repeated Start condition followed by a new address at any time after the previous target slave address has been acknowledged (and before the Stop condition has been sent).

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed.
- Zero or more data bytes may also have been sent or received and ACKed as appropriate.

Process

Send the repeated Start condition using the [PLIB_I2C_MasterStartRepeat](#) function.

Example

```
#define MY_I2C_ID I2C_ID_1
```

```
PLIB_I2C_MasterStartRepeat(MY_I2C_ID);
```

The repeated Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated. It must then be followed by another target slave address (either 7-bit or 10-bit, as appropriate). Strictly speaking, a bus collision can still occur while the repeated start condition is being transmitted. To verify that arbitration has not been lost during transmission of the repeated Start condition, the software should check the [PLIB_I2C_ArbitrationLossHasOccurred](#) function after the repeated Start condition has completed.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID I2C_ID_1
```

```

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    // Abort transfer, retry later

    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}

```

Receiving a Data Byte

The following sequence can be used to receive a data byte and repeated to receive an arbitrary number of data bytes.

Preconditions

- The Start condition must have been sent and completed as previously described
- The target slave address must have been fully transmitted (with the read indication in bit 0) and ACKed by the targeted slave device

Process

1. Start the I²C module cycling the clock line to receive one byte by calling the [PLIB_I2C_MasterReceiverClock1Byte](#) function. The I2C module will generate a master interrupt once the eighth clock cycle has completed, but before the ninth cycle has started.
2. Ensure that a byte was received and is available in the RX buffer using the [PLIB_I2C_ReceivedByteIsAvailable](#) function.
3. If a byte is available, get it from the RX buffer using the [PLIB_I2C_ReceivedByteGet](#) function.
4. ACK or NAK the byte (as determined by software) using the [PLIB_I2C_ReceivedByteAcknowledge](#) function. The master receiver indicates to the slave transmitter that it wishes to terminate the transfer by NAKing the last byte. All other bytes should be ACKed after reception.
5. Ensure that the ACK/NAK signal has completed using the [PLIB_I2C_ReceiverByteAcknowledgeHasCompleted](#) function or by waiting for the master interrupt

Note: Arbitration loss cannot occur during a master reception (only during transmission).

Example: Enabling the Receiver

```
#define MY_I2C_ID I2C_ID_1
PLIB_I2C_MasterReceiverClock1Byte(MY_I2C_ID);
```

Wait for reception to complete before attempting to get the byte from the receiver.

Example: Reading a Byte from the RX Buffer

```
#define MY_I2C_ID I2C_ID_1
#define MY_MAX_COUNT 10
uint8_t data;
int count;

if ( PLIB_I2C_ReceivedByteIsAvailable(MY_I2C_ID) )
{
    data = PLIB_I2C_ReceivedByteGet(MY_I2C_ID);
    count++;

    // Determine if the data should be ACK'd or NAK'd
    if (count < MY_MAX_COUNT)
    {
        PLIB_I2C_ReceivedByteAcknowledge(MY_I2C_ID, true);
    }
    else
    {
        PLIB_I2C_ReceivedByteAcknowledge(MY_I2C_ID, false);
    }
}
```

Wait for the ACK/NAK to complete by waiting for the next master interrupt.

Example: Verifying ACK/NAK Signal is Complete

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(MY_I2C_ID))
{
    // Byte reception sequence complete
}
```

Sending a Stop Condition

When the master transmitter has finished sending data, it must end the transfer with a Stop condition.

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed.

- Zero or more data bytes may also have been sent or received and ACKed as appropriate

Note: At least one address byte must be transmitted between the Start and Stop conditions. Following a Start condition immediately with a Stop condition is a protocol error and can cause some slave devices to lock up and stop responding to the bus.

Process

Send a Stop condition using the [PLIB_I2C_MasterStop](#) function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStop(MY_I2C_ID);
```

The Stop condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Stop condition has been initiated.

After a Stop condition, the bus should be idle unless a master immediately starts a new transfer. Strictly speaking, a bus collision can still occur while the stop condition is being transmitted. Although the transfer is already completed when the Stop condition has been sent, software does not have to abort the transfer; however, it should still check for and clear the arbitration loss condition if it occurs.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

The previous example uses the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I ² C module value from the I2C Module enumeration . Example: <code>#define MY_I2C_ID I2C_ID_1</code>
data	This is the buffer to hold byte of data received.
slaveTargetAddress7Bit	This variable is used to demonstrate how to properly format and access a 7-bit target slave address
SLAVE_TARGET_7BIT_ADDRESS	This constant is used as a place holder for the actual 7-bit target slave address that must be defined by the client code.
slaveAddress10Bit	This variable is used to demonstrate how to properly format and access a 10-bit target slave address
SLAVE_TARGET_10BIT_ADDRESS	This constant is used as a place holder for the actual 10-bit target slave address that must be defined by the client code.

5.6.9.4.6 Interrupt State Machine

The I²C state machine can be used in either a polled or an interrupt-driven manner. However, even when polled, software must check the state of the master, slave, and error interrupt flags to identify when a state transition occurs.

The I²C module can cause three different interrupts:

- Slave Interrupt - A byte was sent or received in Slave mode
- Master Interrupt - A bus action has completed in Master mode

- Error Interrupt - An error has occurred in any mode (see [Handling Errors](#))

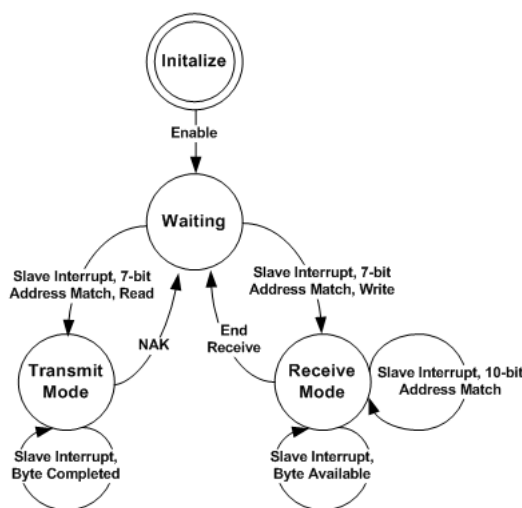
Note: Refer to the "Interrupt Controller" section in the related family reference manual for directions on how to enable, detect, and respond to interrupts.

The I²C module will not start generating interrupts (setting the I²C interrupt flags, even if interrupts are disabled) until it is properly configured and enabled. After that, interrupts are generated as described below. Software has to respond appropriately once the interrupt has occurred (the flag has been set) to allow the state machine to advance to the next state. These actions are described below the state machine diagram to which they refer.

Notes:

1. These state machine diagrams show the normal operations. Error and exception cases are explained in the [Handling Errors](#) section and are not shown in the diagrams.
2. The transition labels sometimes represent hardware interrupts (or the setting of an interrupt flag) and sometimes represent software actions. Refer to the transition tables to understand the transitions.
3. The Wait state is shared between all state machines (i.e., while in the Wait state, the I²C module can start a new master-mode transfer or be addressed by an external master).
4. The Start state is shared between both read and write master transfer state machines (i.e., when starting a new transfer, software can choose start either a read or a write transfer).

Slave Mode State Machine



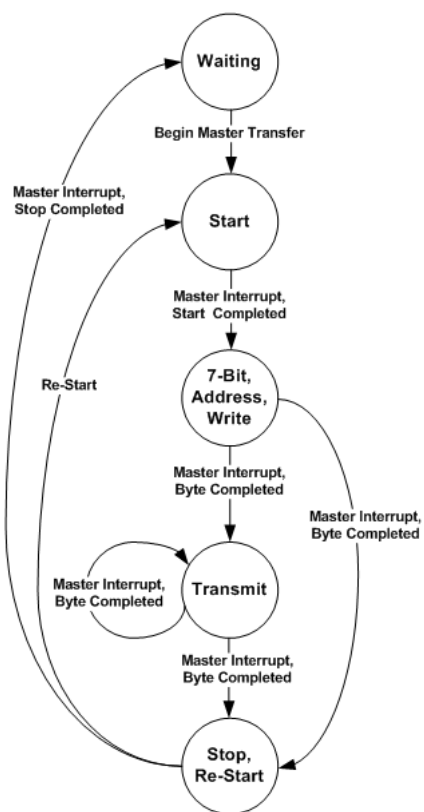
Slave Mode State Transitions

In Slave mode, state transitions occur under hardware control automatically or in response to an interrupt. Software must respond appropriately to ensure that the module continues to operate correctly.

Transition	From State	To State	Software Actions
Enable	Initialize	Wait	Enable the module, using the PLIB_I2C_Enable function
Slave Interrupt, 7-bit Address Match, Write	Wait	Receive mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and the PLIB_I2C_SlaveAddressWasDetected function to verify that a 7-bit address (or the first byte of a 10-bit address) has been received. Then, software must get the address byte from the receiver buffer (the byte is automatically ACKed), using the PLIB_I2C_ReceivedByteGet function. Software must also release the clock by calling PLIB_I2C_SlaveClockRelease function to allow reception of the next byte. Note: This transition occurs on a write transfer with either a 7-bit or 10-bit address (for the first of two address bytes).

Slave Interrupt, Byte Received	Receive mode	Receive mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and, if one has, get the byte from the receiver buffer by calling PLIB_I2C_ReceivedByteGet function. Then, software must release the clock by calling PLIB_I2C_SlaveClockRelease function to allow reception of the next byte.
Slave Interrupt, 10-bit Address Match	Receive mode	Receive mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and (if 10-bit addressing is being used) call the PLIB_I2C_SlaveAddress10BitWasDetected function to verify that the second byte of the 10-bit address matched the low-order 8-bits of the local-slave 10-bit address. Next, software must get the address byte received from the receiver buffer (the byte is automatically ACKed), using the PLIB_I2C_ReceivedByteGet function. Note: This transition occurs on write transfer when using 10-bit (2-byte) addressing when the second address byte is received.
End Receive	Receive mode	Wait	None. Software is not aware of this transition. A receive transfer can end with a stop condition, a repeated start, or if the second-half of a 10-bit slave address does not match the programmed local slave address (and possibly the optional mask).
Slave Interrupt, 7-bit Address Match, Read	Wait	Transmit mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and the PLIB_I2C_SlaveAddressWasDetected function to verify that an address has been received. Then, software must get the address byte from the receiver buffer (the byte is automatically ACKed), using the PLIB_I2C_ReceivedByteGet function. Notes: This transition occurs on a read transfer with a 7-bit (1-byte) address or after a repeated-start with a 10-bit slave address. The second byte of a 10-bit address is not sent after a repeated-start condition, only the first byte. The 10-bit slave device must remember if it was addressed at the beginning of this transfer. After handling the address byte received, software should call the PLIB_I2C_TransmitterIsReady function to make sure that the transmitter buffer is able to accept a byte and, if it is, call the PLIB_I2C_TransmitterByteSend function to send the requested byte.
Slave Interrupt, Byte Completed	Transmit mode	Transmit mode	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function to check to see if the byte was ACKed or NAKed. If the byte was ACKed, software must send the next byte by calling the PLIB_I2C_TransmitterIsReady function to make sure that the transmitter buffer is able to accept a byte and, if it is, call the PLIB_I2C_TransmitterByteSend function to send the next byte.
NAK	Transmit mode	Wait	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function to check to see if the byte was ACKed or NAKed. The master will NAK the last byte it wishes to receive. At that point, the transfer is effectively over and software must not attempt to send any more bytes. The module automatically transitions back to the Wait state when the Stop condition occurs, but software does not receive an interrupt at that point.

Master Mode Write Transfer State Machine

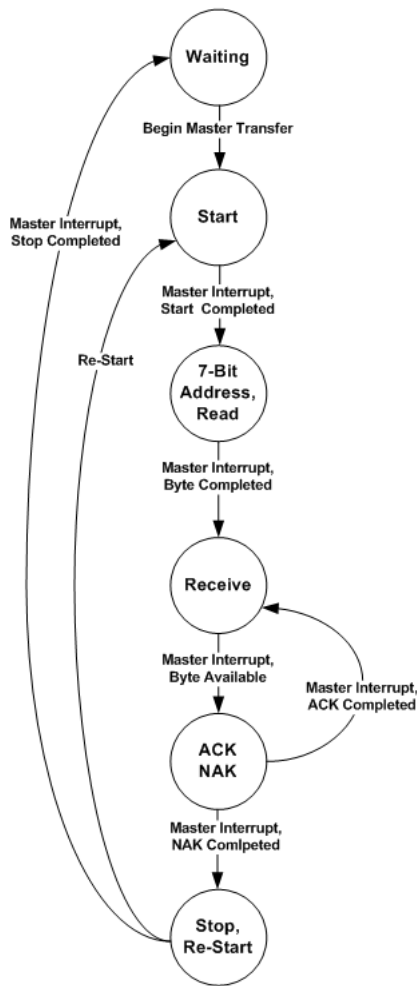


Master-Mode Write-Transfer State Transitions

Transition	From State	To State	Software Actions
Begin Master Transfer	Wait	Start	This state transition occurs when software initiates transmission of the start condition. Software should call the PLIB_I2C_BusIdle function to verify that the bus is not currently in use by another master. Then it must call the PLIB_I2C_MasterStart function (or PLIB_I2C_MasterStartRepeat function) to send the start (or repeated start) condition.
Master Interrupt, Start Completed	Start	7-Bit Address, Write	After the start condition has completed, software should check for a possible collision by calling the PLIB_I2C_ArbitrationLossHasOccurred function. Then, it must send an appropriately formatted 7-bit address byte (or the first byte of the 10-bit address) to the I ² C bus to address a slave target. To do this, it should first call the PLIB_I2C_TransmitterIsReady function to verify that the transmitter buffer is ready to accept a byte and then call the PLIB_I2C_TransmitterByteSend function to send the byte.
Master Interrupt, 7-bit Address, Write Completed	7-bit Address, Write	Transmit	When the master interrupt occurs (or the PLIB_I2C_TransmitterByteHasCompleted function returns true) to indicate that the last byte sent has completed, software must check to ensure that the slave device ACKed the byte by calling the PLIB_I2C_TransmitterByteWasAcknowledged function. Software should also call the PLIB_I2C_ArbitrationLossHasOccurred function to determine if a collision occurred. If all is well, software may then send another byte to the slave using the PLIB_I2C_TransmitterIsReady function and the PLIB_I2C_TransmitterByteSend function. If 10-bit addressing is being used for this transfer, then the start condition and first byte of the 10-bit address must be followed by the second byte of the 10-bit address. Multiple slaves may respond to the first byte of the 10-bit address. It requires the second byte to uniquely identify the 10-bit slave. However, this is sent using the same process as if it were a data byte.

Master Interrupt, Byte Completed	Transmit	Transmit	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function check to ensure that the slave device ACKed the byte and the PLIB_I2C_ArbitrationLossHasOccurred function to determine if a collision occurred. If all is well, software may call the PLIB_I2C_TransmitterIsReady function and PLIB_I2C_TransmitterByteSend function to send another byte to the slave and remain in the Transmit state. If not, software should abort the transfer by sending a Stop condition and transition to the Stop, repeated Start state.
Master Interrupt, Byte Completed	7-bit Address, Write	Stop, repeated Start	If the slave NAKed any byte sent or if software wishes to end or re-start the transfer (perhaps to change direction) it can do so by sending a stop condition (using the PLIB_I2C_MasterStop function) or repeated Start condition (using the PLIB_I2C_MasterStartRepeat function).
Master Interrupt, Byte Completed	Transmit	Stop, repeated Start	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function check to check to see if the slave ACKed or NAKed the byte. If the slave NAKed the byte or if software wishes to end or re-start the transfer (perhaps to change direction) it can do so by sending a Stop condition (using the PLIB_I2C_MasterStop function) or repeated Start condition (using the PLIB_I2C_MasterStartRepeat function).
Restart	Stop, repeated Start	Start	Immediately after initiating the repeated Start condition, software transitions back to the Start state where it must decide what sort of transfer it will start next. Note: Read transfers will transition to the Start state in the Master Mode Read Transfer state machine.
Master Interrupt, Stop Completed	Stop, repeated Start	Wait	When the master interrupt occurs, indicating that the Stop condition has completed, software transitions back to the Wait state (which is actually part of the Slave Mode state machine).

Master-Mode Read-Transfer State Machine



Master Mode Read Transfer State Transitions

In Master mode, transitions are initiated under software control and interrupts occur (the master interrupt flag is set, even when interrupts are disabled) when the action initiated has completed.

If at any point, an address or data byte is NAKed by the slave or an error occurs, usually the safest thing for the master to do is to end the transfer with a Stop condition and release the bus.

Transition	From State	To State	Software Actions
Begin Master Transfer	Wait	Start	This state transition occurs when software initiates transmission of the start condition. Software should call the PLIB_I2C_BusIdle function to verify that the bus is not currently in use by another master. Then it must call the PLIB_I2C_MasterStart function (or PLIB_I2C_MasterStartRepeat function) to send the start (or repeated Start) condition.

Master Interrupt, Start Completed	Start	7-Bit Address, Read	After the start condition has completed, software should check for a possible collision by calling the PLIB_I2C_ArbitrationLossHasOccurred function. Then, it must send an appropriately formatted 7-bit address byte (or the first byte of the 10-bit address) to the I ² C bus to address a slave target. To do this, it should first call the PLIB_I2C_TransmitterIsReady function to verify that the transmitter buffer is ready to accept a byte and then call the PLIB_I2C_TransmitterByteSend function to send the byte. Note: If using 10-bit addressing, this is a repeated transmission of the first byte of the 10-bit address since all 10-bit transfers must start off as write transfers (so both address bytes can be sent) then be re-started to become read transfers.
Master Interrupt, Byte Completed	7-Bit Address, Read	Receive	Software must initiate the "Clock1Byte" operation (using the PLIB_I2C_MasterReceiverClock1Byte function) so that the I ² C module will cycle the SCL line eight times causing the slave to send the requested byte of data.
Master Interrupt, Byte Available	Receive	ACK NAK	When a byte has been received (which software can verify using the PLIB_I2C_ReceivedBytesAvailable function), software must get the received byte from the RX buffer (using the PLIB_I2C_ReceivedByteGet function) and initiate transmission of the ACK/NAK bit (using the PLIB_I2C_ReceivedByteAcknowledge function). If it intends to receive another byte, software must ACK the byte just received. If it does not intend to receive another byte, software must NAK the byte just received.
Master Interrupt, ACK Completed	ACK, NAK	Receive	Software can verify that the ACK signal has completed using the PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function. If software ACKed the previously received byte, it must initiate the "Clock1Byte" operation (using the PLIB_I2C_MasterReceiverClock1Byte function) so that the I ² C module will cycle the SCL line eight times so that the slave can send the requested byte. Otherwise, software must have NAKed the previous byte to end the transfer.
Master Interrupt, NAK Completed	ACK, NAK	Stop, Restart	Software can verify that the NAK signal has completed using the PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function. If software intends to continue the transfer without allowing the bus to go idle, it must initiate a repeated-start (using the PLIB_I2C_MasterStartRepeat function) and transition immediately to the Start state. If software intends to end the transfer and allow the bus to go idle, it must initiate a Stop condition using the PLIB_I2C_MasterStop function.
Restart	Stop, repeated Start	Start	Immediately after initiating the repeated Start condition, software automatically transitions back to the "Start" state where it must decide if it will restart with a read or write transfer. Note: Write transfers will transition to the Start state in the Master Mode Write Transfer state machine.
Master Interrupt, Stop Completed	Stop, repeated Start	Wait	When the master interrupt occurs, indicating that the Stop condition has completed, software transitions back to the waiting state (which is actually part of the Slave Mode state machine).

5.6.9.4.7 Managing Slave Addresses

An I²C address identifies which slave device is being targeted by a master device on a transfer-by-transfer basis. I²C addresses can be either 7-bits or 10-bits, as shown in the following sections, 7-Bit Slave Addresses and 10-Bit Slave Addresses.

Addresses that are 7-bits long are sent in a single byte where bit 0 identifies the direction of the remaining data in the transfer. Addresses that are 10-bits long are sent in two bytes using a reserved range in the 7-bit address space (addresses 0x78 through 0x7C) to encode the two high-order bits (bits A9 and A8) in the first byte. The second byte encodes the eight low-order bites (A7 through A0). These formats are in the following two sections.

7-Bit Slave Addresses

When using 7-bit slave addressing, the address byte is the first byte sent by any I²C master device immediately following a

"Start" condition. The 7-bit address is shifted left by 1 bit to fill bits 7-to-1 in the address byte. Bit 0 in the 7-bit address byte indicates the transfer direction, as follows.

7-Bit Address Format

A6 A5 A4 A3	A2 A1 A0 R/W

Bits	Usage
A6:A0	Bits 7 through 1 of the address byte contain the 7-bit slave address.
R/W	Bit 0 of the address byte is the read/write bit (0 = Read, 1 = Write).

The `I2C_ADDRESS_7_BIT_FORMAT` macro can be used to correctly format a 7-bit slave address. The `I2C_ADDRESS_7_BIT_BYTE_ACCESS` macro can be used to access the byte-format of the address (including the read/write bit). See [Operating as a Master Transmitter](#) and [Operating as a Master Receiver](#) for examples.

10-Bit Slave Addresses

When using 10-bit slave addresses, the first address byte must be sent immediately following a start or repeated Start condition. The second byte must immediately follow the first byte when performing a write transfer. Read transfers must begin as write transfers so that both bytes may be sent. Immediately after sending the second address byte, the master will issue a repeated Start condition and resend the first address byte only (with the read/write bit cleared, indicating a read transfer. The second byte is not retransmitted. Since the transfer has been restarted, all listening 10-bit slave devices know that they will not be address again until the transfer has been ended with a Stop condition. However, devices that respond to 7-bit addresses might respond, so the first byte of the 10-bit address must be repeated to prevent that. The format of the 10-bit address is shown as follows.

10-Bit Address Format

1st Byte

2nd Byte

1 1 1 1	0 A9 A8 R/W	A7 A6 A5 A4	A3 A2 A1 A0

Bits	Usage
11110	Bits 7 through 3 of the first byte are constants. This guarantees that the first byte of the 10-bit address will fall within the reserved range of the 7-bit addresses.
A9:A8	Bits 2 through 1 of the first byte are the two high-order bits of the 10-bit address.
R/W	Bit 0 of the first byte is the Read/Write bit (0 = Read, 1 = Write).
A7:A0	The second byte contains the 8 low-order bits of the 10-bit address.

The `I2C_ADDRESS_10_BIT_FORMAT` macro can be used to correctly format a 10-bit slave address. The `I2C_ADDRESS_10_BIT_1ST_BYTE_ACCESS` macro can be used to access the first byte of the 10-bit address and the `I2C_ADDRESS_10_BIT_2ND_BYTE_ACCESS` macro can be used to access the second byte of the 10-bit address. See [Operating as a Master Transmitter](#) and [Operating as a Master Receiver](#) for examples.

Operating in Slave Mode

In Slave mode, the "slave address" refers to the address (and optional mask) programmed into the I²C module to determine to which address or addresses the module will respond when addressed by an external master. The "raw" (unformatted) address is passed to or returned from the Managing Slave Addresses routines (without being formatted as shown above). Likewise, the mask passed to or returned from the Slave Mask Control Functions is unformatted.

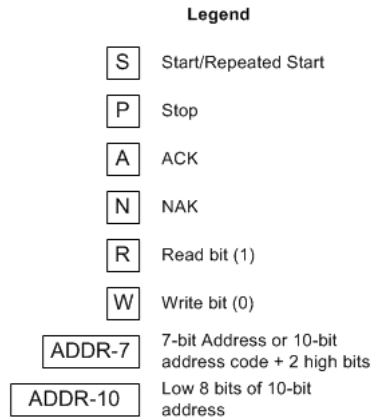
Operating in Master Mode

In Master mode, the "slave address" refers to the address transmitted on the I²C bus to identify the slave target of the transfer. In these cases, the slave address must be correctly formatted (as shown above) and sent on the bus one byte at a time.

5.6.9.4.8 Forming Transfers

Forming Transfers

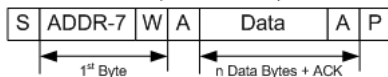
Transfers are formed by chaining the basic "building blocks" shown in the following transfer format legend.



The following are all valid I²C transfer formats:

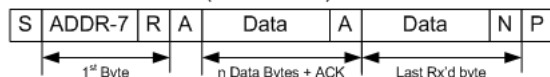
- This format allows a master to transmit n bytes of data to an I²C slave supporting a 7-bit address.

Master-Transmit (7-bit Address)



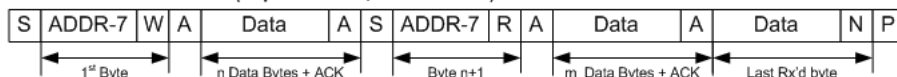
- This format allows a master to receive $n+1$ bytes of data from a "streaming" slave. Such a slave has no internal addressing capability. That is, it has no internal registers or addresses. It can only "stream" data to a master that reads from it.

Master-Receive ONLY (7-bit Address)



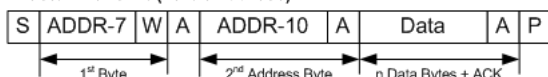
- This format allows a master to write n bytes of data to a slave, followed immediately by reading $n+1$ bytes of data from a slave by sending a repeated Start condition after the first transfer. This format is useful for specifying an internal address to a device, and then reading data from that address. A repeated Start condition can also be used to chain transfers together without letting the bus go idle and potentially losing arbitration.

Master Combined Format (Repeated Start, 7-bit Address)



- The following format allows a master to **send** n bytes of data to a target supporting a 10-bit address.

Master Transmit (10-bit Address)



- This format is used to receive $n+1$ bytes of data from a slave supporting a 10-bit address. The repeated Start condition must be used because the second byte of the 10-bit address must be sent after the first byte of the 10-bit address. Then, the direction of the transfer must be switched to a "read". To do this, after the repeated Start, the master repeats the first byte of

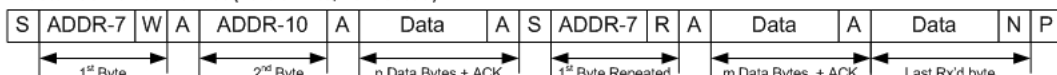
the 10-bit address with a "read" indication in the low-order bit and 10-bit slave devices must "remember" if they were being addressed once the transfer has started.

Master Receive (10-bit Address)



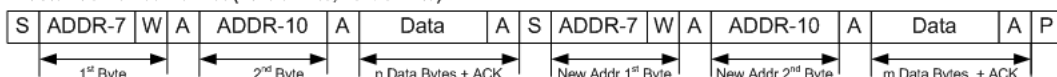
- This format allows a master to **send** n bytes of data to a slave device supporting a 10-bit address and immediately read $n+1$ bytes of data from the same device using the repeated Start and resending the first byte of the 10-bit address technique as shown in the previous format. This format is useful when the master needs to read data from a specific address within a device and it needs to **send** that address before performing the read.

Master Combined Format (10-bit write, 10-bit read)



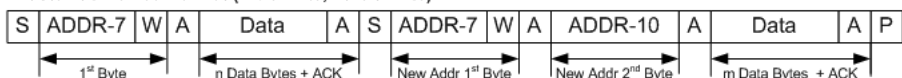
- This format allows a master to chain writes to multiple slaves supporting 10-bit addresses.

Master Combined Format (10-bit write, 10-bit write)



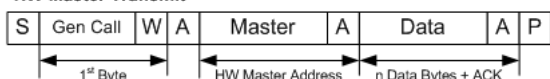
- This format allows a master to chain multiple writes to different slaves. Note that only the first write can be to a slave supporting a 7-bit address. The second (and any subsequent) writes must be to slaves with 10-bit addresses once any 10-bit device has been addressed.

Master Combined Format (7-bit write, 10-bit write)



- This format can be used by a "hardware" master that does not have a programmable address. This allows the hardware master to broadcast to all slaves on the bus using the "General Call" address, followed by the master's address. (The master address is generally the same as the device's slave address if it can also act as a slave device.) This format is not normally used by a microcontroller acting as a master. However, it may be received by a microcontroller if such a hardware master exists on the bus.

HW Master Transmit



5.6.9.4.9 Handling Errors

There are three basic types of errors that can occur during various I²C operations:

- Transmitter Overflow
- Receiver Overflow
- Arbitration Loss

Handling Transmitter Overflow Errors

A transmitter overflow error occurs when the software attempts to write to the TX buffer (by calling the [PLIB_I2C_TransmitterByteSend](#) function) while the transmitter is busy. When this occurs, the write is not allowed and the transmitter overflow status bit is set. This can be identified by calling the [PLIB_I2C_TransmitterOverflowHasOccurred](#) function.

Additional attempts to write to the TX buffer will not be allowed until the transmitter overflow error is cleared by calling the [PLIB_I2C_TransmitterOverflowClear](#) function. This sort of error should be either avoided by checking the [PLIB_I2C_TransmitterIsReady](#) function before attempting to **send** a byte or by checking for the error after attempting to **send** a byte.

Interrupts: The transmitter overflow error does not trigger an interrupt.

Handling Receiver Overflow Errors

A receiver overflow error occurs when an incoming byte has been completely received and is ready to be transferred to the RX buffer, but software has not yet gotten the previous byte from the RX buffer. When this happens the receiver overflow status bit is set and the incoming byte will be lost. This can happen when software does not respond quickly enough to a slave or master interrupt when receiving data. Software should check for this error before calling the [PLIB_I2C_ReceivedByteGet](#) function to get a byte from the RX buffer to determine if any data has been lost. This error can be identified by calling the [PLIB_I2C_ReceiverOverflowHasOccurred](#) function. This error must be cleared by calling the [PLIB_I2C_ReceiverOverflowClear](#) function or no additional data will be received.

Interrupts: The receiver overflow error does not trigger an interrupt (although the incoming data byte does, even though it is lost.)

Handling Arbitration Loss Errors

Strictly speaking, arbitration loss is not an error. Rather it is a normal part of bus operation in a multi-master environment. Arbitration loss occurs when more than one master attempts to transmit on the bus at the same (or nearly the same) time. When this happens, at some point one of the masters will attempt to transmit a binary one ('1') when the other attempts to transmit a binary zero ('0'). At that point, the master transmitting the zero will win the transmission, because the zero transmission will dominate on the bus. The master detecting that the bus signals a zero when it attempted to transmit a one will lose arbitration and must immediately stop transmitting on the bus (which happens automatically).

Arbitration loss can be detected during any of the following actions:

- Transmission of a Start condition
- Transmission of a repeated Start condition
- Transmission of an address or data byte
- Transmission of an acknowledge bit
- Transmission of a Stop condition

Arbitration loss can be identified by calling the [PLIB_I2C_ArbitrationLossHasOccurred](#) function. The arbitration loss condition can be cleared by calling the [PLIB_I2C_ArbitrationLossClear](#) function. Once arbitration loss occurs, the losing master must wait until the bus is once again idle (at which time a master interrupt will occur if arbitration loss has occurred) and retransmit the transfer from the beginning.

Interrupts: The I²C module will trigger an error interrupt when arbitration loss occurs.

5.6.9.4.10 Other Features

Other Features

Note: The following features are not available on all devices. Please refer to the specific device data sheet to determine which features are supported by your device.

General Call Address Support

The I²C General Call address (reserved address 0) allows a master to address all slave devices on the bus. During slave operation the module can be made to respond to or ignore the general call address using the [PLIB_I2C_GeneralCallEnable](#) function or the [PLIB_I2C_GeneralCallDisable](#) function. When a slave address byte is received, software can identify if it was the general call address using the [PLIB_I2C_SlaveAddressIsGeneralCall](#) function (or by checking the value of the address byte

received).

Note: The general call address is always 7-bits.

SMBus Support

The I²C module can support communication on a System Management Bus (SMBus) by calling the [PLIB_I2C_SMBEnable](#) function to adjust the electrical characteristics of the SMBus.

Operation During Sleep Mode

When the device enters Sleep mode, all clock sources supplied to the I²C modules are shut down. If the device enters Sleep mode in the middle of an I²C master transmission or reception operation, the operation is aborted. If the device enters Sleep mode when the module is operating in slave mode, the external master's clock will continue run the slave state machine. If a slave address match occurs, an interrupt will be provided to wake up the device.

Note: Indeterminate results can occur if an error occurs during Sleep mode.

Operation During Idle Mode

When the device enters Idle mode, the system clock sources remain functional and the CPU stops code execution.

I²C operation during Idle mode can be controlled using the [PLIB_I2C_StopInIdleEnable](#) function or the [PLIB_I2C_StopInIdleDisable](#) function.

By default, the I²C module continues to operate in Idle mode and provide full functionality.

Operation with DMA

Software activity must occur between the transfer of each byte, making DMA inappropriate for use with the I²C module.

5.6.9.4.11 Using the Library

Using the Library

This section describes the basic architecture of the I²C Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_i2c.h](#)

The interface to the I²C library is defined in the "[plib_i2c.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the I²C library must include "peripheral.h".

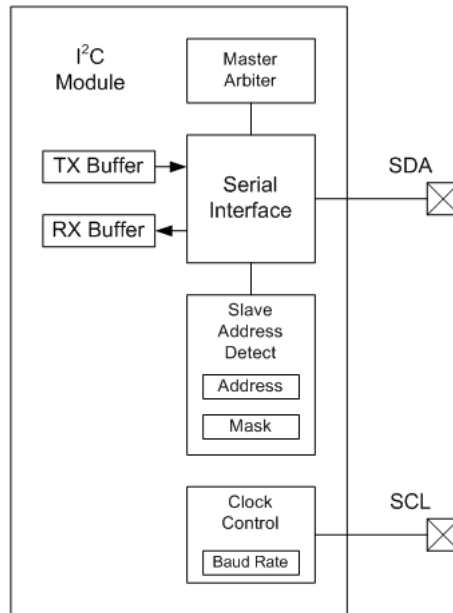
Peripheral Module IDs

Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. To support this, the first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). For the I²C Peripheral Library, the module instance IDs are defined by the [I2C Module enumeration](#). Not all microcontrollers will have all instances of the module listed in this enumeration. Please refer to the specific device data sheet for more information.

5.6.9.4.11.1 Hardware Abstraction Model

This library provides a low-level abstraction of the Inter-Integrated Circuit (I²C) module on Microchip PIC microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Hardware Abstraction Model Diagram



Hardware Abstraction Model Description

The I²C Peripheral Library provides interface routines to interact with the blocks shown in the previous diagram.

The **TX Buffer** and **RX Buffer** are 1-byte (8-bit) buffers for data transmitted (TX) or received (RX) by the **Serial Interface** to the I²C bus over the **SDA** line synchronized with the **SCL** line by the **Clock Control** logic.

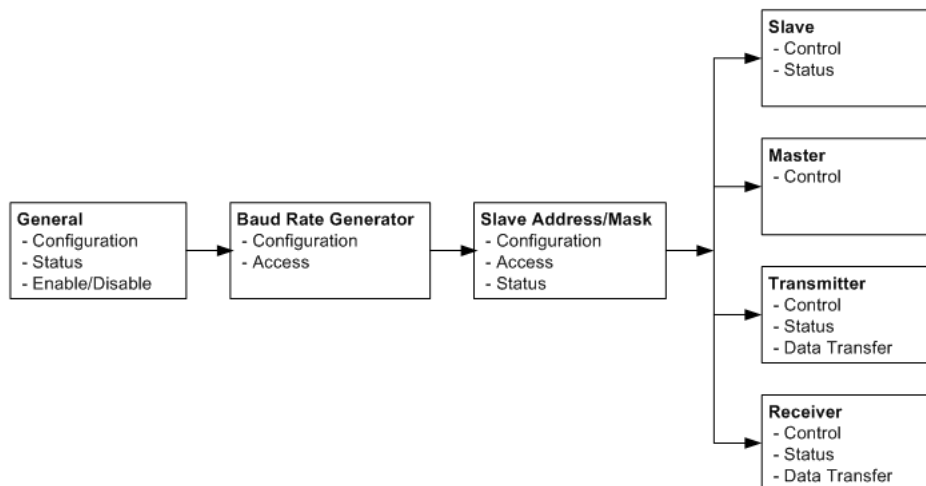
The desired baud rate frequency can be programmed in the **Clock Control** logic (within I²C bus specifications of 0-100 kHz for Low-to-Fast I²C mode and up to 3.4 MHz for High-Speed I²C mode) by setting the **Baud Rate** value. However, the nature of the I²C bus allows slower targets to stretch the clock to slower rates as required, in real time.

The I²C bus is a transfer-based multi-master bus, allowing multiple masters to initiate transfers at any time the bus is not currently busy. The **Master Arbiter** logic monitors transfers and determines if arbitration has been lost. There is no loss of data for the winning master. However, the losing master must retry the entire transfer again later, when the bus is idle. Slave devices are identified by either 7- or-10 bit addresses. The I²C module can act as either a bus master that can initiate transfers or as a slave, responding to transfers initiated by other masters on the bus.

The **Slave Address Detect** logic monitors transfers initiated by other masters to recognize when other masters have addressed this module when it is acting as a slave. The Slave **Address** is programmed into the I²C module by the software along with an (optional) address **Mask**, specifying bits in the address that can be ignored. This ability to "mask off" certain bits in the address allows the I²C module to identify multiple addresses and respond as if it were more than one slave device.

5.6.9.4.11.2 Library Usage Model

Usage Model



The items in the diagram correspond to the groupings in the [Library Interface](#) section.

Library Interface Section	Description

5.6.9.4.12 Configuring the Library

This library is appropriate for microcontrollers with a dedicated I²C module (not an SSP or MSSP module).

5.6.9.5 Library Interface

Data Types and Constants

	Name	Description
	I2C_MODULE_ID	Identifies the supported I2C modules.

Baud Rate Generator Control Functions

	Name	Description
	PLIB_I2C_BaudRateGet	Calculates the I2C module's current SCL clock frequency.
	PLIB_I2C_BaudRateSet	Sets the desired baud rate.

Existence Functions

	Name	Description
	PLIB_I2C_ExistsAcksequenceInProgress	Identifies whether the AcksequenceInProgress feature exists on the I2C module
	PLIB_I2C_ExistsArbitrationLoss	Identifies whether the ArbitrationLoss feature exists on the I2C module
	PLIB_I2C_ExistsBaudRate	Identifies whether the BaudRate feature exists on the I2C module
	PLIB_I2C_ExistsBusIdle	Identifies whether the BusIdle feature exists on the I2C module
	PLIB_I2C_ExistsClockStretching	Identifies whether the ClockStretching feature exists on the I2C module
	PLIB_I2C_ExistsDataLineHoldTime	Identifies whether the DataLineHoldTime feature exists on the I2C module
	PLIB_I2C_ExistsGeneralCall	Identifies whether the GeneralCall feature exists on the I2C module
	PLIB_I2C_ExistsGeneralCallAddressDetect	Identifies whether the GeneralCallAddressDetect feature exists on the I2C module

	PLIB_I2C_ExistsHighFrequency	Identifies whether the HighFrequency feature exists on the I2C module
	PLIB_I2C_ExistsIPMI	Identifies whether the IPMI feature exists on the I2C module
	PLIB_I2C_ExistsMasterReceiverClock1Byte	Identifies whether the MasterReceiverClock1Byte feature exists on the I2C module
	PLIB_I2C_ExistsMasterStart	Identifies whether the MasterStart feature exists on the I2C module
	PLIB_I2C_ExistsMasterStartRepeat	Identifies whether the MasterStartRepeat feature exists on the I2C module
	PLIB_I2C_ExistsMasterStop	Identifies whether the MasterStop feature exists on the I2C module
	PLIB_I2C_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteAcknowledge	Identifies whether the ReceivedByteAcknowledge feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteAvailable	Identifies whether the ReceivedByteAvailable feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteGet	Identifies whether the ReceivedByteGet feature exists on the I2C module
	PLIB_I2C_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the I2C module
	PLIB_I2C_ExistsReservedAddressProtect	Identifies whether the ReservedAddressProtect feature exists on the I2C module
	PLIB_I2C_ExistsSlaveAddress10Bit	Identifies whether the SlaveAddress10Bit feature exists on the I2C module
	PLIB_I2C_ExistsSlaveAddress7Bit	Identifies whether the SlaveAddress7Bit feature exists on the I2C module
	PLIB_I2C_ExistsSlaveAddressDetect	Identifies whether the SlaveAddressDetect feature exists on the I2C module
	PLIB_I2C_ExistsSlaveAddressHoldEnable	Identifies whether the SlaveAddressHoldEnable feature exists on the I2C module
	PLIB_I2C_ExistsSlaveBufferOverwrite	Identifies whether the SlaveBufferOverwrite feature exists on the I2C module
	PLIB_I2C_ExistsSlaveBusCollisionDetect	Identifies whether the SlaveBusCollisionDetect feature exists on the I2C module
	PLIB_I2C_ExistsSlaveClockHold	Identifies whether the SlaveClockHold feature exists on the I2C module
	PLIB_I2C_ExistsSlaveDataDetect	Identifies whether the SlaveDataDetect feature exists on the I2C module
	PLIB_I2C_ExistsSlaveInterruptOnStart	Identifies whether the SlaveInterruptOnStart feature exists on the I2C module
	PLIB_I2C_ExistsSlaveInterruptOnStop	Identifies whether the SlaveInterruptOnStop feature exists on the I2C module
	PLIB_I2C_ExistsSlaveMask	Identifies whether the SlaveMask feature exists on the I2C module
	PLIB_I2C_ExistsSlaveReadRequest	Identifies whether the SlaveReadRequest feature exists on the I2C module
	PLIB_I2C_ExistsSMBus	Identifies whether the SMBus feature exists on the I2C module
	PLIB_I2C_ExistsStartDetect	Identifies whether the StartDetect feature exists on the I2C module
	PLIB_I2C_ExistsStopDetect	Identifies whether the StopDetect feature exists on the I2C module
	PLIB_I2C_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the I2C module
	PLIB_I2C_ExistsTransmitterByteAcknowledge	Identifies whether the TransmitterByteAcknowledge feature exists on the I2C module

	PLIB_I2C_ExistsTransmitterByteComplete	Identifies whether the TransmitterByteComplete feature exists on the I2C module
	PLIB_I2C_ExistsTransmitterByteSend	Identifies whether the TransmitterByteSend feature exists on the I2C module
	PLIB_I2C_ExistsTransmitterIsBusy	Identifies whether the TransmitterBusy feature exists on the I2C module
	PLIB_I2C_ExistsTransmitterOverflow	Identifies whether the TransmitterOverflow feature exists on the I2C module

General Initialization Functions

	Name	Description
	PLIB_I2C_Disable	Disables the specified I2C module.
	PLIB_I2C_Enable	Enables the specified I2C module.
	PLIB_I2C_GeneralCallDisable	Disables the I2C module from recognizing the general call address.
	PLIB_I2C_GeneralCallEnable	Enables the I2C module to recognize the general call address.
	PLIB_I2C_HighFrequencyDisable	Disables the I2C module from using high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_HighFrequencyEnable	Enables the I2C module to use high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_IPMIDisable	Disables the I2C module's support for the IPMI specification
	PLIB_I2C_IPMIEnable	Enables the I2C module to support the Intelligent Platform Management Interface (IPMI) specification (see Remarks).
	PLIB_I2C_ReservedAddressProtectDisable	Disables the I2C module from protecting reserved addresses, allowing it to respond to them.
	PLIB_I2C_ReservedAddressProtectEnable	Enables the I2C module to protect (not respond to) reserved addresses.
	PLIB_I2C_SlaveClockStretchingDisable	Disables the I2C module from stretching the slave clock.
	PLIB_I2C_SlaveClockStretchingEnable	Enables the I2C module to stretch the slave clock.
	PLIB_I2C_SMBDisable	Disable the I2C module support for SMBus electrical signaling levels.
	PLIB_I2C_SMBEnable	Enables the I2C module to support System Management Bus (SMBus) electrical signaling levels.
	PLIB_I2C_StopInIdleDisable	Disables the Stop-in-Idle feature.
	PLIB_I2C_StopInIdleEnable	Enables the I2C module to stop when the processor enters Idle mode

General Status Functions

	Name	Description
	PLIB_I2C_ArbitrationLossClear	Clears the arbitration loss status flag
	PLIB_I2C_ArbitrationLossHasOccurred	Identifies if bus arbitration has been lost.
	PLIB_I2C_BusIsIdle	Determines whether the I2C bus is idle or busy.
	PLIB_I2C_StartClear	Clears the start status flag
	PLIB_I2C_StartWasDetected	Identifies when a Start condition has been detected.
	PLIB_I2C_StopClear	Clears the stop status flag
	PLIB_I2C_StopWasDetected	Identifies when a Stop condition has been detected

Master Control Functions

	Name	Description
	PLIB_I2C_MasterReceiverClock1Byte	Drives the bus clock to receive 1 byte of data from a slave device.
	PLIB_I2C_MasterStart	Sends an I2C Start condition on the I2C bus in Master mode.

	PLIB_I2C_MasterStartRepeat	Sends a repeated Start condition during an ongoing transfer in Master mode.
	PLIB_I2C_MasterStop	Sends an I2C Stop condition to terminate a transfer in Master mode.

Receiver Control Functions

	Name	Description
	PLIB_I2C_ReceivedByteAcknowledge	Allows a receiver to acknowledge a that a byte of data has been received.
	PLIB_I2C_ReceivedByteGet	Gets a byte of data received from the I2C bus interface.
	PLIB_I2C_ReceivedBytesAvailable	Detects whether the receiver has data available.
	PLIB_I2C_ReceiverByteAcknowledgeHasCompleted	Determines if the previous acknowledge has completed.
	PLIB_I2C_ReceiverOverflowClear	Clears the receiver overflow status flag.
	PLIB_I2C_ReceiverOverflowHasOccurred	Identifies if a receiver overflow error has occurred.
	PLIB_I2C_MasterReceiverReadyToAcknowledge	Checks whether the hardware is ready to acknowledge.

Slave Address Control Functions

	Name	Description
	PLIB_I2C_SlaveAddress10BitGet	Identifies the current 10-bit Slave mode address.
	PLIB_I2C_SlaveAddress10BitSet	Selects 10-bit Slave mode addressing and sets the address value.
	PLIB_I2C_SlaveAddress10BitWasDetected	Detects reception of the second byte of a 10-bit slave address.
	PLIB_I2C_SlaveAddress7BitGet	Identifies the current 7-bit Slave mode address.
	PLIB_I2C_SlaveAddress7BitSet	Selects 7-bit Slave mode addressing and sets the slave address value.
	PLIB_I2C_SlaveAddressHoldDisable	Disables Address holding.
	PLIB_I2C_SlaveAddressHoldEnable	Enables address holding.
	PLIB_I2C_SlaveAddressIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveAddressIsGeneralCall	Identifies if the current slave address received is the general call address.
	PLIB_I2C_SlaveAddressModeIs10Bits	Identifies if the current slave address mode is 7-bits or 10-bits.
	PLIB_I2C_SlaveDataIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveReadIsRequested	Detects if the request from the master was a read or write.

Slave Control Functions

	Name	Description
	PLIB_I2C_AcksequenceIsInProgress	Checks whether the acknowledge sequence is in progress or it is completed.
	PLIB_I2C_DataLineHoldTimeSet	Sets the data line hold time.
	PLIB_I2C_SlaveBufferOverwriteDisable	Disables buffer overwrite.
	PLIB_I2C_SlaveBufferOverwriteEnable	Enables buffer overwrite.
	PLIB_I2C_SlaveBusCollisionDetectDisable	Disables bus collision detect interrupts.
	PLIB_I2C_SlaveBusCollisionDetectEnable	Enables slave bus collision interrupts.
	PLIB_I2C_SlaveClockHold	Holds the SCL clock line low after receiving a byte.
	PLIB_I2C_SlaveClockRelease	Releases a previously held SCL clock line.
	PLIB_I2C_SlaveDataHoldDisable	Disables data holding.
	PLIB_I2C_SlaveDataHoldEnable	Enables data holding.
	PLIB_I2C_SlaveInterruptOnStartDisable	Disables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStartEnable	Enables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStopDisable	Disables the feature of generating interrupt on stop condition.

	PLIB_I2C_SlaveInterruptOnStopEnable	Enables the feature of generating interrupt on stop condition.

Slave Mask Control Functions

	Name	Description
	PLIB_I2C_SlaveMask10BitGet	Identifies the current 10-bit Slave mode address mask.
	PLIB_I2C_SlaveMask10BitSet	Sets the 10-bit mask for Slave mode addressing.
	PLIB_I2C_SlaveMask7BitGet	Identifies the current 7-bit Slave mode address mask.
	PLIB_I2C_SlaveMask7BitSet	Sets the 7-bit mask for Slave mode addressing .

Transmitter Control Functions

	Name	Description
	PLIB_I2C_TransmitterByteHasCompleted	Detects whether the module has finished transmitting the most recent byte.
	PLIB_I2C_TransmitterByteSend	Sends a byte of data on the I2C bus.
	PLIB_I2C_TransmitterByteWasAcknowledged	Determines whether the most recently sent byte was acknowledged.
	PLIB_I2C_TransmitterIsBusy	Identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).
	PLIB_I2C_TransmitterIsReady	Detects if the transmitter is ready to accept data to transmit.
	PLIB_I2C_TransmitterOverflowClear	Clears the transmitter overflow status flag.
	PLIB_I2C_TransmitterOverflowHasOccurred	Identifies if a transmitter overflow error has occurred.

Description

The I²C peripheral library consists of a set of interface routines, data types and constants, and macros provided by the [plib_i2c.h](#) header file. This section describes those items.

5.6.9.5.1 General Initialization Functions

5.6.9.5.1.1 PLIB_I2C_Disable Function

C

```
void PLIB_I2C_Disable(
    I2C_MODULE_ID index
);
```

Description

This function disables the specified I2C module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsModuleEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_Disable( MY_I2C_ID );
```

5.6.9.5.1.2 PLIB_I2C_Enable Function

C

```
void PLIB_I2C_Enable(
    I2C_MODULE_ID index
);
```

Description

This function enables the specified I2C module.

Preconditions

The module should be appropriately configured before being enabled.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsModuleEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID          I2C_ID_1
#define MY_CLOCK_FREQUENCY 80000000
#define MY_BAUD_RATE      10000
#define MY_SLAVE_ADDRESS  0x23

PLIB_I2C_SlaveClockStretchingEnable ( MY_I2C_ID );
PLIB_I2C_SMBDisable ( MY_I2C_ID );
PLIB_I2C_HighFrequencyDisable ( MY_I2C_ID );
PLIB_I2C_ReservedAddressProtectEnable ( MY_I2C_ID );
PLIB_I2C_StopInIdleDisable ( MY_I2C_ID );
PLIB_I2C_BaudRateSet ( MY_I2C_ID, MY_CLOCK_FREQUENCY, MY_BAUD_RATE );

PLIB_I2C_SlaveAddress7BitSet( MY_I2C_ID, MY_SLAVE_ADDRESS );
PLIB_I2C_Enable ( MY_I2C_ID );
```

5.6.9.5.1.3 PLIB_I2C_GeneralCallDisable Function

C

```
void PLIB_I2C_GeneralCallDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables the I2C module from recognizing the general call address when operating as a slave receiver.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The General-call feature can be re-enabled by calling the [PLIB_I2C_GeneralCallEnable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsGeneralCall](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_GeneralCallDisable(MY_I2C_ID);
```

5.6.9.5.1.4 PLIB_I2C_GeneralCallEnable Function**C**

```
void PLIB_I2C_GeneralCallEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables the I2C module to recognize the general call address when operating as a slave receiver.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The General-call feature can be disabled by calling the [PLIB_I2C_GeneralCallDisable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsGeneralCall](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_GeneralCallEnable ( MY_I2C_ID );
```

5.6.9.5.1.5 PLIB_I2C_HighFrequencyDisable Function**C**

```
void PLIB_I2C_HighFrequencyDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables the I2C module from using high-frequency signaling, preventing it from using the 400 kHz and 1 MHz signaling rates.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The high-frequency feature can be re-enabled by calling the [PLIB_I2C_HighFrequencyEnable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsHighFrequency](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_HighFrequencyDisable ( MY_I2C_ID );
```

5.6.9.5.1.6 PLIB_I2C_HighFrequencyEnable Function**C**

```
void PLIB_I2C_HighFrequencyEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables the I2C module to use high-frequency signaling, allowing it to use the 400 kHz and 1 MHz signaling rates.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The high-speed feature can be disabled by calling the `PLIB_I2C_HighSpeedEnable` function.

This feature must be enabled if frequencies higher than 100 kHz programmed using the [PLIB_I2C_BaudRateSet](#) function.

The `MY_I2C_ID` macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsHighFrequency](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_HighFrequencyEnable ( MY_I2C_ID );
```

5.6.9.5.1.7 PLIB_I2C_IPMIDisable Function**C**

```
void PLIB_I2C_IPMIDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables the I2C module's support for the IPMI specification.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

Please refer to the IPMI specification for details of the Intelligent Platform Management Interface. The IPMI specification is the property of Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Inc.

The `MY_I2C_ID` macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsIPMI](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_IPMIDisable ( MY_I2C_ID );
```

5.6.9.5.1.8 PLIB_I2C_IPMIEnable Function

C

```
void PLIB_I2C_IPMIEnable(  
    I2C_MODULE_ID index  
);
```

Description

This function enables the I2C module to support the IPMI specification.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

Please refer to the IPMI specification for details of the Intelligent Platform Management Interface. The IPMI specification is the property of Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Inc.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsIPMI](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1  
  
PLIB_I2C_IPMIEnable ( MY_I2C_ID );
```

5.6.9.5.1.9 PLIB_I2C_ReservedAddressProtectDisable Function

C

```
void PLIB_I2C_ReservedAddressProtectDisable(  
    I2C_MODULE_ID index  
);
```

Description

This function disables the I2C module from protecting reserved addresses, allowing it to respond to them when they match the module's slave address and mask.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The reserved-address-protect feature can be re-enabled by calling the [PLIB_I2C_ReservedAddressProtectEnable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsReservedAddressProtect](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_ReservedAddressProtectDisable ( MY_I2C_ID );
```

5.6.9.5.1.10 PLIB_I2C_ReservedAddressProtectEnable Function**C**

```
void PLIB_I2C_ReservedAddressProtectEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables the I2C module to protect reserved addresses by not responding to them, even if they match the module's slave address and mask.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The reserved-address-protect feature can be disabled by calling the [PLIB_I2C_ReservedAddressProtectDisable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsReservedAddressProtect](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_ReservedAddressProtectEnable(MY_I2C_ID);
```

5.6.9.5.1.11 PLIB_I2C_SlaveClockStretchingDisable Function**C**

```
void PLIB_I2C_SlaveClockStretchingDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables the I2C module from stretching the slave clock to allow time for software to respond to bytes received from the master.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The clock stretching feature can be re-enabled by calling the [PLIB_I2C_SlaveClockStretchingEnable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsClockStretching](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockStretchingDisable ( MY_I2C_ID );
```

5.6.9.5.1.12 PLIB_I2C_SlaveClockStretchingEnable Function**C**

```
void PLIB_I2C_SlaveClockStretchingEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables the I2C module to stretch the slave clock to allow time for software to respond to bytes received from the master.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The clock stretching feature can be disabled by calling the [PLIB_I2C_SlaveClockStretchingDisable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsClockStretching](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockStretchingEnable ( MY_I2C_ID );
```

5.6.9.5.1.13 PLIB_I2C_SMBDisable Function**C**

```
void PLIB_I2C_SMBDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables the I2C module support for SMBus electrical signaling levels.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The SMB feature can be re-enabled by calling the [PLIB_I2C_SMBEnable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSMBus](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SMBDisable ( MY_I2C_ID );
```

5.6.9.5.1.14 PLIB_I2C_SMBEnable Function**C**

```
void PLIB_I2C_SMBEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables the I2C module to support SMBus electrical signaling levels.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The SMB feature can be disabled by calling the [PLIB_I2C_SMBDisable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSMBus](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1  
  
PLIB_I2C_SMBEnable ( MY_I2C_ID );
```

5.6.9.5.1.15 PLIB_I2C_StopInIdleDisable Function**C**

```
void PLIB_I2C_StopInIdleDisable(  
    I2C_MODULE_ID index  
);
```

Description

This function disables the Stop-in-Idle feature, preventing the I2C module from stopping when the processor enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The Stop-in-Idle feature can be re-enabled by calling the [PLIB_I2C_StopInIdleEnable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1  
  
PLIB_I2C_StopInIdleDisable ( MY_I2C_ID );
```

5.6.9.5.1.16 PLIB_I2C_StopInIdleEnable Function

C

```
void PLIB_I2C_StopInIdleEnable(  
    I2C_MODULE_ID index  
);
```

Description

This function enables the I2C module to stop when the processor enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The Stop-in-Idle feature can be disabled by calling the [PLIB_I2C_StopInIdleDisable](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1  
  
PLIB_I2C_StopInIdleEnable ( MY_I2C_ID );
```

5.6.9.5.2 General Status Functions

5.6.9.5.2.1 PLIB_I2C_ArbitrationLossClear Function

C

```
void PLIB_I2C_ArbitrationLossClear(  
    I2C_MODULE_ID index  
);
```

Description

This function clears the arbitration loss status flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

This flag is set by hardware when bus arbitration loss occurs. Its status can be tested using the [PLIB_I2C_ArbitrationLossHasOccurred](#) function. This flag should be cleared by software after the arbitration loss has been handled. To handle the error, the entire transmission (from the Start condition to the Stop or restart condition) must be retried later when the bus becomes idle.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsArbitrationLoss](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ArbitrationLossHasOccurred ( MY_I2C_ID ) )
{
    // Handle bus collision

    PLIB_I2C_ArbitrationLossClear ( MY_I2C_ID );
}
```

5.6.9.5.2.2 PLIB_I2C_ArbitrationLossHasOccurred Function**C**

```
bool PLIB_I2C_ArbitrationLossHasOccurred(
    I2C_MODULE_ID index
);
```

Description

This function identifies if bus arbitration has been lost.

Preconditions

Bus collisions can occur during any master-mode transmission including:

- Sending a Start condition
- Sending a repeated Start condition
- Sending an address or data byte
- sending an ACK/NAK bit
- Sending a Stop condition

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If software if a bus collision occurred, resulting in loss of bus arbitration
- false - If no bus collision occurred

Remarks

The arbitration status should be checked after any Master mode transmission or if an error interrupt occurs. If a bus collision occurs, the entire transmission (from the Start condition to the Stop or restart condition) must be retried later when the bus becomes idle.

This flag should be cleared by software using the [PLIB_I2C_ArbitrationLossClear](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsArbitrationLoss](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ArbitrationLossHasOccurred ( MY_I2C_ID ) )
{
    // Handle bus collision

    PLIB_I2C_ArbitrationLossClear( MY_I2C_ID );
}
```

5.6.9.5.2.3 PLIB_I2C_BusIsIdle Function

C

```
bool PLIB_I2C_BusIsIdle(
    I2C_MODULE_ID index
);
```

Description

This function checks to see if the I2C bus is currently idle or if there is some activity currently taking place.

Preconditions

The module must be configured appropriately and enabled.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - The bus is currently idle. It is safe to start a master transfer.
- false - The bus is currently busy. Do not start a master transfer.

Remarks

When this function returns true it does not guarantee that a bus arbitration loss cannot occur. Two or more masters can start a transfer within the minimum start signal hold time. (Refer to the I2C specification for a definition of the minimum Start condition hold time.)

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsBusIsIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
```

```
if (PLIB_I2C_BusIsIdle ( MY_I2C_ID ))
{
    PLIB_I2C_MasterStart ( MY_I2C_ID );
}
```

5.6.9.5.2.4 PLIB_I2C_StartClear Function

C

```
void PLIB_I2C_StartClear(
    I2C_MODULE_ID index
);
```

Description

This function clears the start status flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

This flag is cleared automatically by the hardware when a Stop condition is detected, but it can also be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsStartDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StartWasDetected( MY_I2C_ID ) )
{
    // Handle Start condition

    PLIB_I2C_StartClear(MY_I2C_ID);
}
```

5.6.9.5.2.5 PLIB_I2C_StartWasDetected Function

C

```
bool PLIB_I2C_StartWasDetected(
    I2C_MODULE_ID index
);
```

Description

This function identifies when a Start condition has been detected.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - A Start Condition has been detected
- false - No Start condition has been detected since the last time a Stop condition was detected (or the module was initialized)

Remarks

This flag is cleared automatically by the hardware when a stop condition is detected, but it can also be cleared by software using [PLIB_I2C_StartClear](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsStartDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StartWasDetected ( MY_I2C_ID ) )
{
    // Handle Start condition
}
```

5.6.9.5.2.6 PLIB_I2C_StopClear Function**C**

```
void PLIB_I2C_StopClear(
    I2C_MODULE_ID index
);
```

Description

This function clears the stop status flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

This flag is cleared automatically by the hardware when a Start condition is detected, but it can also be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsStopDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StopWasDetected ( MY_I2C_ID ) )
{
    // Handle stop condition

    PLIB_I2C_StopClear ( MY_I2C_ID );
}
```

5.6.9.5.2.7 PLIB_I2C_StopWasDetected Function**C**

```
bool PLIB_I2C_StopWasDetected(
    I2C_MODULE_ID index
);
```

Description

This function identifies when a Stop condition has been detected.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - A Stop condition has been detected
- false - No Stop condition has been detected since the last time a Start condition was detected (or the module was initialized)

Remarks

This flag is cleared automatically by the hardware when a Start condition is detected, but it can also be cleared by software using the [PLIB_I2C_StopClear](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsStopDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StopWasDetected ( MY_I2C_ID ) )
{
    // Handle stop condition
}
```

5.6.9.5.3 Baud Rate Generator Control Functions

5.6.9.5.3.1 PLIB_I2C_BaudRateGet Function

C

```
I2C_BAUD_RATE PLIB_I2C_BaudRateGet(  
    I2C_MODULE_ID index,  
    uint32_t clockFrequencyHz  
);
```

Description

This function calculates the I2C module's current SCL clock frequency.

Preconditions

The returned value may not be valid if [PLIB_I2C_BaudRateSet](#) has not been previously called to set the SCL clock frequency.

Parameters

Parameters	Description
index	Identifies the desired I2C module
clockFrequencyHz	Clock Frequency (Hz) provided for the I2C module

Returns

SCL frequency currently used

Remarks

The actual frequency provided may be slightly different than the frequency requested due to truncation errors. The actual frequency observed on the SCL line may be lower due to clock stretching.

The MY_CLOCK_FREQUENCY macro in the example is used as placeholder for the actual clock frequency.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsBaudRate](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1  
#define MY_CLOCK_FREQ_INPUT 80000000  
uint32_t myBaudRate;  
  
myBaudRate = PLIB_I2C_BaudRateGet ( MY_I2C_ID, MY_CLOCK_FREQ_INPUT );
```

5.6.9.5.3.2 PLIB_I2C_BaudRateSet Function

C

```
void PLIB_I2C_BaudRateSet(  
    I2C_MODULE_ID index,  
    uint32_t clockFrequencyHz,  
    I2C_BAUD_RATE baudRate  
);
```

Description

This function sets the desired baud rate so that the I2C module will operate at the desired clock frequency (on the SCL line of the bus.)

Preconditions

The source clock, being sent to the I2C module (internal to the microcontroller) must be operating at the frequency passed.

Parameters

Parameters	Description
index	Identifies the desired I2C module
clockFrequencyHz	Clock Frequency (Hz)
baudRate	The desired baud rate value. This should be an appropriate value for the frequency and microcontroller in use.

Returns

None.

Remarks

IMPORTANT: The I2C module's high-frequency mode must be enabled for frequencies higher than 100 kHz using the [PLIB_I2C_HighFrequencyEnable](#) function. Otherwise, the high-frequency mode must be disabled using the [PLIB_I2C_HighFrequencyDisable](#) function.

The actual frequency selected may be slightly different than the frequency requested due to truncation errors. Use the [PLIB_I2C_BaudRateGet](#) function to obtain the actual baud rate value that has been programmed.

The actual frequency observed on the SCL line may be lower due to clock stretching.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue, verify that no receptions are in progress before changing the system clock.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsBaudRate](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
#define MY_CLOCK_FREQUENCY_INPUT 80000000

PLIB_I2C_HighFrequencyDisable ( MY_I2C_ID );
PLIB_I2C_BaudRateSet ( MY_I2C_ID, MY_CLOCK_FREQUENCY_INPUT, 40000 );
```

5.6.9.5.4 Slave Address Control Functions**5.6.9.5.4.1 PLIB_I2C_SlaveAddress10BitGet Function****C**

```
uint16_t PLIB_I2C_SlaveAddress10BitGet(
    I2C_MODULE_ID index
);
```

Description

This function identifies the 10-bit slave address to which the module will currently respond.

Preconditions

The address provided may not be valid if it has not been previously set.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

The 10-bit slave address to which the module is currently set to respond.

Remarks

The 16-bit address will be right-aligned in the 16-bit return value.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveAddress10Bit](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint16_t slaveAddress;

if (PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slaveAddress = PLIB_I2C_SlaveAddress10BitGet ( MY_I2C_ID );
}
```

5.6.9.5.4.2 PLIB_I2C_SlaveAddress10BitSet Function

C

```
void PLIB_I2C_SlaveAddress10BitSet(
    I2C_MODULE_ID index,
    uint16_t address
);
```

Description

This function selects 10-bit Slave mode addressing sets the slave address to which the module will respond when operating in Slave mode.

Preconditions

None

Parameters

Parameters	Description
index	Identifies the desired I2C module
address	The 10-bit slave address to which the module will respond (The address should be right-aligned in the 16-bit parameter, without any read/write bit in the 0 position)

Returns

None.

Remarks

I2C modules on some microcontroller families may also support an address mask (to allow the module to respond to multiple addresses. When using these microcontrollers, the [PLIB_I2C_SlaveAddress10BitSetMasked](#) may be used to support the mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_10_BIT macro in the example is used as a placeholder for the actual desired slave address.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveAddress10Bit](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
#define MY_SLAVE_ADDR_10_BIT    0x23

PLIB_I2C_SlaveAddress10BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_10_BIT );
```

5.6.9.5.4.3 PLIB_I2C_SlaveAddress10BitWasDetected Function**C**

```
bool PLIB_I2C_SlaveAddress10BitWasDetected(
    I2C_MODULE_ID index
);
```

Description

This function detects if the second byte of a 10-bit slave address (containing the low-order 8 bits) matching the local address has been received.

Preconditions

The module must have been configured for 10-bit addressing in Slave mode, enabled, and the first byte of the 10-bit local slave address must have already been received and matched.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the second byte of the local 10-bit address has been received
- false - If the second byte of the local 10-bit address has not been received

Remarks

This function should only be used by slave receivers.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveAddressDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t i2cReadData;

if ( PLIB_I2C_SlaveAddress10BitWasDetected ( MY_I2C_ID ) )
{
    i2cReadData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

5.6.9.5.4.4 PLIB_I2C_SlaveAddress7BitGet Function**C**

```
uint8_t PLIB_I2C_SlaveAddress7BitGet(
    I2C_MODULE_ID index
);
```

Description

This function identifies the 7-bit slave address to which the module will currently respond.

Preconditions

The address provided may not be valid if it has not been previously set.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

The 7-bit slave address to which the module is currently set to respond.

Remarks

The 7-bit address will be right-aligned in the 8-bit return value.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveAddress7Bit](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t slave_address7bit;

if (!PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slave_address7bit = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
}
```

5.6.9.5.4.5 PLIB_I2C_SlaveAddress7BitSet Function**C**

```
void PLIB_I2C_SlaveAddress7BitSet(
    I2C_MODULE_ID index,
    uint8_t address
);
```

Description

This function selects 7-bit Slave mode addressing sets the address to which the module will respond when operating in Slave mode.

Preconditions

None

Parameters

Parameters	Description
index	Identifies the desired I2C module
address	The 7-bit slave address to which the module will respond (The address should be right-aligned in the 8-bit parameter, without any read/write bit in the 0 position)

Returns

None.

Remarks

I2C modules on some microcontroller families may also support an address mask (to allow the module to respond to multiple addresses. When using these microcontrollers, the [PLIB_I2C_SlaveAddress7BitSetMasked](#) may be used to support the mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_7_BIT macro in the example is used as a placeholder for the actual desired slave address.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveAddress7Bit](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddress7BitSet ( MY_I2C_ID, MY_SLAVE_ADDRESS_7_BIT );
```

5.6.9.5.4.6 PLIB_I2C_SlaveAddressHoldDisable Function

C

```
void PLIB_I2C_SlaveAddressHoldDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables address holding.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_SlaveAddressHoldEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddressHoldDisable ( MY_I2C_ID );
```

5.6.9.5.4.7 PLIB_I2C_SlaveAddressHoldEnable Function

C

```
void PLIB_I2C_SlaveAddressHoldEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables address holding. If address byte is received, following the 8th falling edge of clock for a clock line will be held low.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_I2C_SlaveAddressHoldEnable` in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddressHoldEnable ( MY_I2C_ID );
```

5.6.9.5.4.8 PLIB_I2C_SlaveAddressIsDetected Function**C**

```
bool PLIB_I2C_SlaveAddressIsDetected(
    I2C_MODULE_ID index
);
```

Description

This function identifies if the most recent byte received was a data byte or an address byte.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the byte received is an address byte
- false - If the byte received is a data byte

Remarks

This function should only be used by slave receivers.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveReadRequest](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t getData;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_SlaveAddressIsDetected ( MY_I2C_ID ) )
    {
```

```

    if ( PLIB_I2C_SlaveReadIsRequested ( MY_I2C_ID ) )
    {
        // Begin slave transmit mode
    }
    else
    {
        // Begin slave receive mode
    }
}

// Read the data and release the bus
getData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
PLIB_I2C_SlaveClockRelease ( MY_I2C_ID );
}

```

5.6.9.5.4.9 PLIB_I2C_SlaveAddressIsGeneralCall Function

C

```

bool PLIB_I2C_SlaveAddressIsGeneralCall(
    I2C_MODULE_ID index
);

```

Description

This function identifies if the current slave address received is the general call address.

Preconditions

A slave address must have been received.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the slave address received is the general call address
- false - if the slave address received is not the general call address

Remarks

This bit is set when the general call address has been received.

This bit is automatically cleared by hardware when a Stop condition occurs and cannot be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsGeneralCallAddressDetect](#) in your application to determine whether this feature is available.

Example

```

#define MY_I2C_ID I2C_ID_1
uint8_t slaveAddress7Bit;

if ( !PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_SlaveAddressIsGeneralCall( MY_I2C_ID ) )
    {
        // Handle general call address
    }

    slaveAddress7Bit = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
}

```

```
}
```

5.6.9.5.4.10 PLIB_I2C_SlaveAddressModeIs10Bits Function

C

```
bool PLIB_I2C_SlaveAddressModeIs10Bits(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies if the current slave addressing mode is 7-bits or 10-bits.

Preconditions

The mode provided may not be valid if the address mode has not been previously set.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the current slave addressing mode is 10-bits
- false - if the current slave addressing mode is 7-bits

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveAddress10Bit](#) in your application to determine whether this feature is available.

Example

```
uint8_t slave_address7Bit;  
uint16_t slave_address10Bit;  
  
if ( PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )  
{  
    slave_address10Bit = PLIB_I2C_SlaveAddress10BitGet ( MY_I2C_ID );  
}  
else  
{  
    slave_address7Bit = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );  
}
```

5.6.9.5.4.11 PLIB_I2C_SlaveDataIsDetected Function

C

```
bool PLIB_I2C_SlaveDataIsDetected(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies if the most recent byte received was a data byte or an address byte.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Parameters

Parameters	Description
index	Identifies the desired I2C module.

Returns

- true - If the byte received is a data byte
- false - If the byte received is an address byte

Remarks

This function should only be used by slave receivers.

This function returns the opposite of the [PLIB_I2C_SlaveAddressIsDetected](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveDataDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t dataReceived;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_SlaveDataIsDetected( MY_I2C_ID ) )
    {
        // Read the data and release the bus
        dataReceived = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
        PLIB_I2C_SlaveClockRelease ( MY_I2C_ID );
    }
}
```

5.6.9.5.4.12 PLIB_I2C_SlaveReadIsRequested Function**C**

```
bool PLIB_I2C_SlaveReadIsRequested(
    I2C_MODULE_ID index
);
```

Description

This function identifies if a slave read (transmit) or a slave write (receive) was requested by the master that addressed the module.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If an external master is requesting data (slave read/transmit)
- false - If an external master is sending data (slave write/receive)

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveReadRequest](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t slaveTxData;
uint8_t slaveRxData;

if ( PLIB_I2C_SlaveReadIsRequested ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_TransmitterIsReady ( MY_I2C_ID ) )
    {
        PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, slaveTxData );
    }
}
else
{
    slaveRxData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

5.6.9.5.5 Slave Mask Control Functions

5.6.9.5.5.1 PLIB_I2C_SlaveMask10BitGet Function

C

```
uint16_t PLIB_I2C_SlaveMask10BitGet(
    I2C_MODULE_ID index
);
```

Description

This function identifies the 10-bit slave address mask that is currently being used to determine which slave addresses the module will respond.

Preconditions

The address mask provided may not be valid if it has not been previously been set.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

The 10-bit slave address mask that the module is currently using to determine to which addresses the module will respond.

Remarks

The 10-bit address mask will be Right-aligned in the 16-bit return value.

This function is not supported on microcontrollers with I2C modules that do not support the slave address mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_I2C_ExistsSlaveMask](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

uint16_t slave_address;
uint16_t slave_addressMask;

if ( PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slave_address      = PLIB_I2C_SlaveAddress10BitGet ( MY_I2C_ID );
    slave_addressMask = PLIB_I2C_SlaveMask10BitGet ( MY_I2C_ID );
}
```

5.6.9.5.5.2 PLIB_I2C_SlaveMask10BitSet Function

C

```
void PLIB_I2C_SlaveMask10BitSet(
    I2C_MODULE_ID index,
    uint16_t mask
);
```

Description

This function sets the 10-bit mask for Slave mode addressing. The address mask (if supported) is used, along with the slave address to identify to which addresses the module will respond when operating in Slave mode. It acts as an "ignore" mask, allowing the module to ignore bits within the slave address and thus respond to multiple slave addresses on microcontrollers with I2C modules that support the mask feature.

Preconditions

None

Parameters

Parameters	Description
index	Identifies the desired I2C module
mask	This parameter identifies bits in the address that are "don't care" bits. These bits will be ignored when attempting to match the address, effectively allowing the module to recognize multiple slave addresses. (To match an address exactly, this
value must be zero (0).) (This value must also be right	aligned in the 16-bit parameter.)

Returns

None.

Remarks

I2C modules on some microcontroller families may not support the mask feature, in which case the `PLIB_I2C_SlaveMask10BitSet` function will not be supported. Refer to the specific device data sheet to determine whether his feature is supported for your device.

The `MY_SLAVE_ADDRESS_10_BIT` and `MY_SLAVE_ADDRESS_MASK_10_BIT` macros in the example code are used as placeholders for the actual desired slave address and mask that must be filled in by the caller.

The `MY_I2C_ID` macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific `I2C_MODULE_ID` enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveMask](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID          I2C_ID_1
#define MY_SLAVE_ADDR_10_BIT 0x23
#define MY_SLAVE_ADDR_MASK_10_BIT 0x12

PLIB_I2C_SlaveAddress10BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_10_BIT );
PLIB_I2C_SlaveMask10BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_MASK_10_BIT );
```

5.6.9.5.5.3 PLIB_I2C_SlaveMask7BitGet Function**C**

```
uint8_t PLIB_I2C_SlaveMask7BitGet(
    I2C_MODULE_ID index
);
```

Description

This function identifies the 7-bit Slave mode address mask that is currently being used to determine which slave addresses the module will respond.

Preconditions

The address mask provided may not be valid if it has not been previously been set.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

The 7-bit Slave mode address mask that the module is currently using to determine to which addresses the module will respond.

Remarks

The 7-bit address mask will be right-aligned in the 8-bit return value.

This function is not supported on microcontrollers with I2C modules that do not support the slave address mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveMask](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t slaveAddr;
uint8_t slaveAddressMsk;

if ( !PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slaveAddr      = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
    slaveAddressMsk = PLIB_I2C_SlaveMask7BitGet ( MY_I2C_ID );
}
```

5.6.9.5.5.4 PLIB_I2C_SlaveMask7BitSet Function**C**

```
void PLIB_I2C_SlaveMask7BitSet(
    I2C_MODULE_ID index,
    uint8_t mask
```

```
);
```

Description

This function sets the 7-bit mask for Slave mode addressing. The address mask (if supported) is used, along with the slave address to identify to which addresses the module will respond when operating in Slave mode. It acts as an "ignore" mask, allowing the module to ignore bits within the slave address and will respond to multiple slave addresses on microcontrollers with I2C modules that support the mask feature.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module
mask	This parameter identifies bits in the address that are "don't care" bits. These bits will be ignored when attempting to match the address, effectively allowing the module to recognize multiple slave addresses. (To match an address exactly, this parameter must be zero (0).)

Returns

None.

Remarks

I2C modules on some microcontroller families may not support the mask feature, in which case the `PLIB_I2C_SlaveMask7BitSet` function will not be supported. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The `MY_SLAVE_ADDRESS_7_BIT` and `MY_SLAVE_ADDRESS_MASK_7_BIT` macros in the example code are used as placeholders for the actual desired slave address and mask that must be filled in by the caller.

The `MY_I2C_ID` macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific `I2C_MODULE_ID` enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveMask](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID           I2C_ID_1
#define MY_SLAVE_ADDR_7_BIT 0x23
#define MY_SLAVE_ADDR_MASK_7_BIT 0x12

PLIB_I2C_SlaveAddress7BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_7_BIT );
PLIB_I2C_SlaveMask7BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_MASK_7_BIT );
```

5.6.9.5.6 Slave Control Functions

5.6.9.5.6.1 PLIB_I2C_AcksequenceIsInProgress Function

C

```
bool PLIB_I2C_AcksequenceIsInProgress(
    I2C_MODULE_ID index
);
```

Description

This function checks whether the acknowledge sequence is in progress or it is completed.

Preconditions

The module must have been configured appropriately and enabled.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

true - Acknowledge sequence is in progress. false - Acknowledge sequence is not started or completed.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsAcksequenceProgress](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_AcksequenceIsInProgress ( MY_I2C_ID ) )
{
    //Transmission completed
}
```

5.6.9.5.6.2 PLIB_I2C_DataLineHoldTimeSet Function**C**

```
void PLIB_I2C_DataLineHoldTimeSet(
    I2C_MODULE_ID index,
    I2C_SDA_MIN_HOLD_TIME sdaHoldTimeNs
);
```

Description

This function sets the data line hold time.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module.
sdaHoldTimeNs	SDA hold time in nano seconds.

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsDataLineHoldTime](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_PLIB_I2C_DataLineHoldTimeSet ( MY_I2C_ID, I2C_SDA_MIN_HOLD_TIME_300NS );
```

5.6.9.5.6.3 PLIB_I2C_SlaveBufferOverwriteDisable Function**C**

```
void PLIB_I2C_SlaveBufferOverwriteDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables buffer overwrite. If the buffer overwrite is disabled, on data receive, when the previous data is not read, the buffer will not be overwritten.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveBufferOverwrite](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveBufferOverwriteDisable ( MY_I2C_ID );
```

5.6.9.5.6.4 PLIB_I2C_SlaveBufferOverwriteEnable Function**C**

```
void PLIB_I2C_SlaveBufferOverwriteEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables buffer overwrite. If the buffer overwrite is enabled, on data receive, even if the previous data is not read, the buffer will be filled with new data and acknowledge will be generated.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveBufferOverwrite](#) in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveBufferOverwriteEnable ( MY_I2C_ID );
```

5.6.9.5.6.5 PLIB_I2C_SlaveBusCollisionDetectDisable Function**C**

```
void PLIB_I2C_SlaveBusCollisionDetectDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables bus collision detect interrupts.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_SlaveBusCollisionDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveBusCollisionDetectDisable ( MY_I2C_ID );
```

5.6.9.5.6.6 PLIB_I2C_SlaveBusCollisionDetectEnable Function**C**

```
void PLIB_I2C_SlaveBusCollisionDetectEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables bus collision interrupts.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveBusCollisionDetect in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddressHoldEnable ( MY_I2C_ID );
```

5.6.9.5.6.7 PLIB_I2C_SlaveClockHold Function**C**

```
void PLIB_I2C_SlaveClockHold(
    I2C_MODULE_ID index
);
```

Description

This function allows an I2C slave to stretch the SCL clock line, holding it low to throttle data transfer from a master transmitter.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started by an external master.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

This function will cause the SCL line to be forced low, after the current byte has been fully received.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_I2C_ExistsSlaveClockHold](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockHold ( MY_I2C_ID );
```

5.6.9.5.6.8 PLIB_I2C_SlaveClockRelease Function**C**

```
void PLIB_I2C_SlaveClockRelease(
    I2C_MODULE_ID index
);
```

Description

This function allows a slave receiver to release a previously held SCL clock line, allowing it to go high and allowing data transfer to continue.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started by an external master, and the SCL clock line should have been previously held (forced low) by the I2C module.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

Calling this function when the clock has not been held will not cause any problems.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveClockHold](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockRelease ( MY_I2C_ID );
```

5.6.9.5.6.9 PLIB_I2C_SlaveDataHoldDisable Function**C**

```
void PLIB_I2C_SlaveDataHoldDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables data holding.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_SlaveDataHoldEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveDataHoldDisable ( MY_I2C_ID );
```

5.6.9.5.6.10 PLIB_I2C_SlaveDataHoldEnable Function**C**

```
void PLIB_I2C_SlaveDataHoldEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables data holding. If a data byte is received, following the 8th falling edge of clock for a clock line will be held low.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_SlaveDataHoldEnable](#) in your application to determine whether this feature is available.

This API is supported only for the slave mode of I2C.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveDataHoldEnable ( MY_I2C_ID );
```


5.6.9.5.6.11 PLIB_I2C_SlaveInterruptOnStartDisable Function

C

```
void PLIB_I2C_SlaveInterruptOnStartDisable(  
    I2C_MODULE_ID index  
);
```

Description

This function disables the feature of generating interrupt on start condition.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveInterruptOnStart](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1  
  
PLIB_I2C_SlaveInterruptOnStartDisable ( MY_I2C_ID );
```

5.6.9.5.6.12 PLIB_I2C_SlaveInterruptOnStartEnable Function

C

```
void PLIB_I2C_SlaveInterruptOnStartEnable(  
    I2C_MODULE_ID index  
);
```

Description

This function enables the feature of generating interrupt on start condition.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveInterruptOnStart](#) in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStartEnable ( MY_I2C_ID );
```

5.6.9.5.6.13 PLIB_I2C_SlaveInterruptOnStopDisable Function

C

```
void PLIB_I2C_SlaveInterruptOnStopDisable(
    I2C_MODULE_ID index
);
```

Description

This function disables the feature of generating interrupt on stop condition.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveInterruptOnStop](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStopDisable ( MY_I2C_ID );
```

5.6.9.5.6.14 PLIB_I2C_SlaveInterruptOnStopEnable Function

C

```
void PLIB_I2C_SlaveInterruptOnStopEnable(
    I2C_MODULE_ID index
);
```

Description

This function enables the feature of generating interrupt on stop condition.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsSlaveInterruptOnStop](#) in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStopEnable ( MY_I2C_ID );
```

5.6.9.5.7 Master Control Functions

5.6.9.5.7.1 PLIB_I2C_MasterReceiverClock1Byte Function

C

```
void PLIB_I2C_MasterReceiverClock1Byte(
    I2C_MODULE_ID index
);
```

Description

This function drives the bus clock to receive 1 byte of data from a slave device.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and the module must be the intended receiver of the next byte of data.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

The module stops driving the bus clock after the reception of a single byte of data and this function must be called again to receive another byte.

After the module has finished receiving a data byte (determined by responding to an I2C master interrupt and/or by checking the [PLIB_I2C_ReceivedByteIsAvailable](#) function), software should check the [PLIB_I2C_ReceiverOverflowHasOccurred](#) function to ensure that no data was lost because the receiver buffer was full when the byte was completely received and ready to be placed into the receiver buffer.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the

processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsMasterReceiverClock1Byte](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterReceiverClock1Byte ( MY_I2C_ID );
```

5.6.9.5.7.2 PLIB_I2C_MasterStart Function

C

```
void PLIB_I2C_MasterStart(
    I2C_MODULE_ID index
);
```

Description

This function sends the start signal (a falling edge on SDA while SCL is high) to start a transfer on the I2C bus when acting in Master mode.

Preconditions

The module must have been configured appropriately and enabled and the I2C bus must currently be idle.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

Only an I2C master can start a transfer on the bus. The bus is considered to be busy after a Start condition.

After the Start condition has completed (detected by responding to the I2C master interrupt), software should check for arbitration loss by calling the [PLIB_I2C_ArbitrationLossHasOccurred](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsMasterStart](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_BusIsIdle ( MY_I2C_ID ) )
{
    PLIB_I2C_MasterStart ( MY_I2C_ID );
}
```

5.6.9.5.7.3 PLIB_I2C_MasterStartRepeat Function

C

```
void PLIB_I2C_MasterStartRepeat(
    I2C_MODULE_ID index
);
```

Description

This function supports sending a repeated Start condition to change slave targets or transfer direction to support certain I2C transfer formats in Master mode.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

Only an I2C master that has already started a transfer can [send](#) a repeated Start condition.

After the repeated-start condition has completed (detected by responding to the I2C master interrupt), software should check for arbitration loss by the [PLIB_I2C_ArbitrationLossHasOccurred](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsMasterStartRepeat](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStartRepeat ( MY_I2C_ID );
```

5.6.9.5.7.4 PLIB_I2C_MasterStop Function

C

```
void PLIB_I2C_MasterStop(
    I2C_MODULE_ID index
);
```

Description

This function sends the stop signal (a rising edge on SDA while SCL is high) on the I2C bus, to end a transfer on the I2C bus when in Master mode.

Preconditions

The module must have been configured appropriately, enabled, and a previously started transfer must be completed.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

Only an I2C master that has already started a transfer can [send](#) a Stop condition.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsMasterStop](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStop ( MY_I2C_ID );
```

5.6.9.5.8 Transmitter Control Functions

5.6.9.5.8.1 PLIB_I2C_TransmitterByteHasCompleted Function

C

```
bool PLIB_I2C_TransmitterByteHasCompleted(
    I2C_MODULE_ID index
);
```

Description

This function determines if the transmitter has finished sending the most recently sent byte on the I2C bus.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and a data or address byte must have been sent.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the transmitter has completed sending the data byte
- false - If the transmitter is still busy sending the data byte

Remarks

This function should be used by both master and slave transmitters.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsTransmitterByteComplete](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_TransmitterByteHasCompleted ( MY_I2C_ID ) )
{
    //Transmission completed
}
```

5.6.9.5.8.2 PLIB_I2C_TransmitterByteSend Function

C

```
void PLIB_I2C_TransmitterByteSend(
```

```

    I2C_MODULE_ID index,
    uint8_t data
);

```

Description

This function allows the caller to [send](#) a byte of data on the I2C bus.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and the I2C module's transmitter must be ready to [accept](#) a byte of data to [send](#).

Parameters

Parameters	Description
index	Identifies the desired I2C module
data	Data byte to send on the I2C bus

Returns

None.

Remarks

This function should be used by both master and slave transmitters.

The caller must either first call the [PLIB_I2C_TransmitterIsReady](#) function before calling this function to ensure that the transmitter is ready to receive a new byte to transmit or call the [PLIB_I2C_TransmitterOverflowHasOccurred](#) function immediately after calling this function to ensure that a transmitter write collision did not occur.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsTransmitterByteSend](#) in your application to determine whether this feature is available.

Example

```

#define MY_I2C_ID I2C_ID_1
uint8_t transmitData;

//set 'transmitData' variable

if ( PLIB_I2C_TransmitterIsReady ( MY_I2C_ID ) )
{
    PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, transmitData );
}

```

5.6.9.5.8.3 PLIB_I2C_TransmitterByteWasAcknowledged Function

C

```

bool PLIB_I2C_TransmitterByteWasAcknowledged(
    I2C_MODULE_ID index
);

```

Description

This function allows a transmitter to determine if the byte just sent was positively acknowledged (ACKed) or negatively acknowledged (NAKed) by the receiver.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, a byte of data must have been sent on the I2C bus, and the transmission must have completed.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the receiver ACKed the byte
- false - If the receiver NAKed the byte

Remarks

This function can be used by both master or slave transmitters.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsTransmitterByteAcknowledge](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_TransmitterByteHasCompleted ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_TransmitterByteWasAcknowledged ( MY_I2C_ID ) )
    {
        // transmission successful
    }
}
```

5.6.9.5.8.4 PLIB_I2C_TransmitterIsBusy Function**C**

```
bool PLIB_I2C_TransmitterIsBusy(
    I2C_MODULE_ID index
);
```

Description

This function identifies if the transmitter of the specified I2C module is currently busy (unable to [accept](#) more data).

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - The transmitter is busy (unable to [accept](#) new data)
- false - The transmitter is ready (able to [accept](#) new data)

Remarks

This function returns the inverse of the [PLIB_I2C_TransmitterIsReady](#) function.

This flag is cleared automatically by the hardware when the transmitter is ready. It cannot be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsTransmitterIsBusy](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t transData;

//set 'transData' variable

if ( !PLIB_I2C_TransmitterIsBusy( MY_I2C_ID ) )
{
    PLIB_I2C_TransmitterByteSend( MY_I2C_ID, transData );
}
```

5.6.9.5.8.5 PLIB_I2C_TransmitterIsReady Function**C**

```
bool PLIB_I2C_TransmitterIsReady(
    I2C_MODULE_ID index
);
```

Description

This function determines if the transmitter is ready to [accept](#) more data to be transmitted on the I2C bus.

Preconditions

The module must have been configured appropriately, enabled, and a transfer must have been previously started.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the transmitter is ready to [accept](#) more data
- false - If the transmitter is not ready to [accept](#) more data

Remarks

This function should be used by both master and slave transmitters.

This function returns the inverse of the [PLIB_I2C_TransmitterIsBusy](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsTransmitterIsBusy](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t sendData;

//set 'sendData' variable

if ( PLIB_I2C_TransmitterIsReady ( MY_I2C_ID ) )
{
    PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, sendData );
}
```

5.6.9.5.8.6 PLIB_I2C_TransmitterOverflowClear Function

C

```
void PLIB_I2C_TransmitterOverflowClear(  
    I2C_MODULE_ID index  
)  
;
```

Description

This function clears the transmitter overflow status flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

This flag is set by hardware when an overflow error occurs. Its status can be tested using the [PLIB_I2C_TransmitterOverflowHasOccurred](#) function. This flag must be cleared by software after the overflow error has been handled. To handle the error, software must retry the write later after the [PLIB_I2C_TransmitterByteSend](#) function returns TRUE.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsTransmitterOverflow](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1  
uint8_t my_data;  
  
//set 'my_data' variable  
PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, my_data );  
if ( PLIB_I2C_TransmitterOverflowHasOccurred ( MY_I2C_ID ) )  
{  
    // Handle overflow error  
  
    PLIB_I2C_TransmitterOverflowClear ( MY_I2C_ID );  
}
```

5.6.9.5.8.7 PLIB_I2C_TransmitterOverflowHasOccurred Function

C

```
bool PLIB_I2C_TransmitterOverflowHasOccurred(  
    I2C_MODULE_ID index  
)  
;
```

Description

This function identifies if a transmitter overflow error has occurred.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

true - If software attempted to write a byte to the transmitter buffer while the transmitter was busy and unable to [accept](#) a new byte (i.e., the write will not occur)

false - If no transmitter overflow occurred when software attempted to write a byte to the transmitter buffer (i.e., the write occurred successfully)

Remarks

This flag must be cleared by software using the [PLIB_I2C_TransmitterOverflowClear](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsTransmitterOverflow](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t txData;

//set 'txData' variable

PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, txData );
if ( PLIB_I2C_TransmitterOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_TransmitterOverflowClear ( MY_I2C_ID );
}
```

5.6.9.5.9 Receiver Control Functions

5.6.9.5.9.1 PLIB_I2C_ReceivedByteAcknowledge Function

C

```
void PLIB_I2C_ReceivedByteAcknowledge(
    I2C_MODULE_ID index,
    bool ack
);
```

Description

This function allows a receiver to positively acknowledge (ACK) or negatively acknowledge (NAK) a byte of data that has been received from the I2C bus.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and a byte of data must have been received from the I2C bus.

Parameters

Parameters	Description
index	Identifies the desired I2C module

5.6.9.5.9.2 PLIB_I2C_ReceivedByteGet Function

C

```
uint8_t PLIB_I2C_ReceivedByteGet(  
    I2C_MODULE_ID index  
);
```

Description

This function allows the caller to read a byte of data received from the I2C bus.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and a byte of data must have been received from the I2C bus.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

A byte of data received from the I2C bus.

Remarks

This function should be used by both master and slave receivers.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsReceivedByteGet](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1  
uint8_t receivedData;  
  
if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )  
{  
    PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );  
    receivedData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );  
}
```

5.6.9.5.9.3 PLIB_I2C_ReceivedByteIsAvailable Function

C

```
bool PLIB_I2C_ReceivedByteIsAvailable(  
    I2C_MODULE_ID index  
);
```

Description

This function determines if the receiver has data available to be read by software.

Preconditions

The I2C module must have been configured appropriately and enabled, and a transfer must have been previously started.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the receiver has data available
- false - If the receiver does not have data available

Remarks

This function should be used by both master and slave receivers.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsReceivedByteAvailable](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t readData;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    readData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

5.6.9.5.9.4 PLIB_I2C_ReceiverByteAcknowledgeHasCompleted Function**C**

```
bool PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(
    I2C_MODULE_ID index
);
```

Description

This function allows the receiver to determine if the acknowledgment signal has completed.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, a byte of data must have been received on the I2C bus, and the acknowledgement must have been started (by calling the [PLIB_I2C_ReceivedByteAcknowledge](#) function).

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the acknowledgment has completed
- false - If the acknowledgment has not completed

Remarks

This function can only be used by master receivers. Slave receivers automatically ACK or NAK bytes.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsReceivedByteAcknowledge](#) in your application to determine whether this feature is available.

Example

```
PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );
```

```
if ( PLIB_I2C_ReceiverByteAcknowledgeHasCompleted ( MY_I2C_ID ) )
{
    // acknowledgment completed
}
```

5.6.9.5.9.5 PLIB_I2C_ReceiverOverflowClear Function

C

```
void PLIB_I2C_ReceiverOverflowClear(
    I2C_MODULE_ID index
);
```

Description

This function clears the receiver overflow status flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

None.

Remarks

This flag is set by hardware when an overflow error occurs. It's status can be tested using the [PLIB_I2C_ReceiverOverflowHasOccurred](#) function. This flag should be cleared by software after the overflow error has been handled by reading the byte in the receiver buffer.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsReceiverOverflow](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ReceiverOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_ReceiverOverflowClear ( MY_I2C_ID );
}
```

5.6.9.5.9.6 PLIB_I2C_ReceiverOverflowHasOccurred Function

C

```
bool PLIB_I2C_ReceiverOverflowHasOccurred(
    I2C_MODULE_ID index
);
```

Description

This function identifies if a receiver overflow error has occurred.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If a byte was received while the receiver buffer still held a previously received byte (The new byte will be lost)
- false - If no receiver overflow occurred when a byte has been received

Remarks

This flag should be cleared by software using the [PLIB_I2C_ReceiverOverflowClear](#) function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsReceiverOverflow](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ReceiverOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_ReceiverOverflowClear ( MY_I2C_ID );
}
```

5.6.9.5.9.7 PLIB_I2C_MasterReceiverReadyToAcknowledge Function**C**

```
bool PLIB_I2C_MasterReceiverReadyToAcknowledge(
    I2C_MODULE_ID index
);
```

Description

This function checks for preconditions before acknowledging a slave.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started and a byte of data must have been received from the I2C bus.

Parameters

Parameters	Description
index	Identifies the desired I2C module

Returns

- true - If the hardware status is ready to acknowledge
- false - If the hardware is not ready to acknowledge

Remarks

This function can only be used by master receivers.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific [I2C_MODULE_ID](#) enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_I2C_ExistsReceivedByteAcknowledge](#) in your application to determine whether this feature is available.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t data;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_MasterReceiverReadyToAcknowledge ( MY_I2C_ID ) )
    {
        PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );
        data = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
    }
}
```

5.6.9.5.10 Existence Functions

5.6.9.5.10.1 PLIB_I2C_ExistsAcksequenceProgress Function

C

```
bool PLIB_I2C_ExistsAcksequenceProgress(
    I2C_MODULE_ID index
);
```

Description

This function identifies whether the AcksequenceInProgress feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_AcksequenceInProgress](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the AcksequenceInProgress feature is supported on the device
- false - If the AcksequenceInProgress feature is not supported on the device

Remarks

None.

5.6.9.5.10.2 PLIB_I2C_ExistsArbitrationLoss Function

C

```
bool PLIB_I2C_ExistsArbitrationLoss(
    I2C_MODULE_ID index
);
```


Description

This function identifies whether the ArbitrationLoss feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_ArbitrationLossHasOccurred](#)
- [PLIB_I2C_ArbitrationLossClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ArbitrationLoss feature is supported on the device
- false - If the ArbitrationLoss feature is not supported on the device

Remarks

None.

5.6.9.5.10.3 PLIB_I2C_ExistsBaudRate Function**C**

```
bool PLIB_I2C_ExistsBaudRate(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the BaudRate feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_BaudRateSet](#)
- [PLIB_I2C_BaudRateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the BaudRate feature is supported on the device
- false - If the BaudRate feature is not supported on the device

Remarks

None.

5.6.9.5.10.4 PLIB_I2C_ExistsBusIsIdle Function

C

```
bool PLIB_I2C_ExistsBusIsIdle(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the BusIdle feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_BusIsIdle](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the BusIdle feature is supported on the device
- false - If the BusIdle feature is not supported on the device

Remarks

None.

5.6.9.5.10.5 PLIB_I2C_ExistsClockStretching Function

C

```
bool PLIB_I2C_ExistsClockStretching(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the ClockStretching feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveClockStretchingEnable](#)
- [PLIB_I2C_SlaveClockStretchingDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ClockStretching feature is supported on the device
- false - If the ClockStretching feature is not supported on the device

Remarks

None.

5.6.9.5.10.6 PLIB_I2C_ExistsDataLineHoldTime Function

C

```
bool PLIB_I2C_ExistsDataLineHoldTime(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the DataLineHoldTime feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_DataLineHoldTimeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the DataLineHoldTime feature is supported on the device
- false - If the DataLineHoldTime feature is not supported on the device

Remarks

None.

5.6.9.5.10.7 PLIB_I2C_ExistsGeneralCall Function

C

```
bool PLIB_I2C_ExistsGeneralCall(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the GeneralCall feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_GeneralCallEnable](#)
- [PLIB_I2C_GeneralCallDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the GeneralCall feature is supported on the device
- false - If the GeneralCall feature is not supported on the device

Remarks

None.

5.6.9.5.10.8 PLIB_I2C_ExistsGeneralCallAddressDetect Function

C

```
bool PLIB_I2C_ExistsGeneralCallAddressDetect(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the GeneralCallAddressDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveAddressIsGeneralCall](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the GeneralCallAddressDetect feature is supported on the device
- false - If the GeneralCallAddressDetect feature is not supported on the device

Remarks

None.

5.6.9.5.10.9 PLIB_I2C_ExistsHighFrequency Function

C

```
bool PLIB_I2C_ExistsHighFrequency(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the HighFrequency feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_HighFrequencyEnable](#)
- [PLIB_I2C_HighFrequencyDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the HighFrequency feature is supported on the device
- false - If the HighFrequency feature is not supported on the device

Remarks

None.

5.6.9.5.10.10 PLIB_I2C_ExistsIPMI Function

C

```
bool PLIB_I2C_ExistsIPMI(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the IPMI feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_IPMIEnable](#)
- [PLIB_I2C_IPMIDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the IPMI feature is supported on the device
- false - If the IPMI feature is not supported on the device

Remarks

None.

5.6.9.5.10.11 PLIB_I2C_ExistsMasterReceiverClock1Byte Function

C

```
bool PLIB_I2C_ExistsMasterReceiverClock1Byte(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the MasterReceiverClock1Byte feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_MasterReceiverClock1Byte](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the MasterReceiverClock1Byte feature is supported on the device
- false - If the MasterReceiverClock1Byte feature is not supported on the device

Remarks

None.

5.6.9.5.10.12 PLIB_I2C_ExistsMasterStart Function

C

```
bool PLIB_I2C_ExistsMasterStart(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the MasterStart feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_MasterStart](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the MasterStart feature is supported on the device
- false - If the MasterStart feature is not supported on the device

Remarks

None.

5.6.9.5.10.13 PLIB_I2C_ExistsMasterStartRepeat Function

C

```
bool PLIB_I2C_ExistsMasterStartRepeat(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the MasterStartRepeat feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_MasterStartRepeat](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the MasterStartRepeat feature is supported on the device
- false - If the MasterStartRepeat feature is not supported on the device

Remarks

None.

5.6.9.5.10.14 PLIB_I2C_ExistsMasterStop Function

C

```
bool PLIB_I2C_ExistsMasterStop(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the MasterStop feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_MasterStop](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the MasterStop feature is supported on the device
- false - If the MasterStop feature is not supported on the device

Remarks

None.

5.6.9.5.10.15 PLIB_I2C_ExistsModuleEnable Function

C

```
bool PLIB_I2C_ExistsModuleEnable(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the ModuleEnable feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_Enable](#)
- [PLIB_I2C_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ModuleEnable feature is supported on the device
- false - If the ModuleEnable feature is not supported on the device

Remarks

None.

5.6.9.5.10.16 PLIB_I2C_ExistsReceivedByteAcknowledge Function

C

```
bool PLIB_I2C_ExistsReceivedByteAcknowledge(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the ReceivedByteAcknowledge feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_ReceivedByteAcknowledge](#)
- [PLIB_I2C_ReceiverByteAcknowledgeHasCompleted](#)
- [PLIB_I2C_MasterReceiverReadyToAcknowledge](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReceivedByteAcknowledge feature is supported on the device
- false - If the ReceivedByteAcknowledge feature is not supported on the device

Remarks

None.

5.6.9.5.10.17 PLIB_I2C_ExistsReceivedByteAvailable Function

C

```
bool PLIB_I2C_ExistsReceivedByteAvailable(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the ReceivedByteAvailable feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_ReceivedByteIsAvailable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReceivedByteAvailable feature is supported on the device
- false - If the ReceivedByteAvailable feature is not supported on the device

Remarks

None.

5.6.9.5.10.18 PLIB_I2C_ExistsReceivedByteGet Function**C**

```
bool PLIB_I2C_ExistsReceivedByteGet(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the ReceivedByteGet feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_ReceivedByteGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReceivedByteGet feature is supported on the device
- false - If the ReceivedByteGet feature is not supported on the device

Remarks

None.

5.6.9.5.10.19 PLIB_I2C_ExistsReceiverOverflow Function**C**

```
bool PLIB_I2C_ExistsReceiverOverflow(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiverOverflow feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_ReceiverOverflowHasOccurred](#)
- [PLIB_I2C_ReceiverOverflowClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReceiverOverflow feature is supported on the device

- false - If the ReceiverOverflow feature is not supported on the device

Remarks

None.

5.6.9.5.10.20 PLIB_I2C_ExistsReservedAddressProtect Function**C**

```
bool PLIB_I2C_ExistsReservedAddressProtect(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the ReservedAddressProtect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_ReservedAddressProtectEnable](#)
- [PLIB_I2C_ReservedAddressProtectDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReservedAddressProtect feature is supported on the device
- false - If the ReservedAddressProtect feature is not supported on the device

Remarks

None.

5.6.9.5.10.21 PLIB_I2C_ExistsSlaveAddress10Bit Function**C**

```
bool PLIB_I2C_ExistsSlaveAddress10Bit(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveAddress10Bit feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveAddress10BitSet](#)
- [PLIB_I2C_SlaveAddress10BitGet](#)
- [PLIB_I2C_SlaveAddressModels10Bits](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveAddress10Bit feature is supported on the device
- false - If the SlaveAddress10Bit feature is not supported on the device

Remarks

None.

5.6.9.5.10.22 PLIB_I2C_ExistsSlaveAddress7Bit Function**C**

```
bool PLIB_I2C_ExistsSlaveAddress7Bit(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveAddress7Bit feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveAddress7BitSet](#)
- [PLIB_I2C_SlaveAddress7BitGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveAddress7Bit feature is supported on the device
- false - If the SlaveAddress7Bit feature is not supported on the device

Remarks

None.

5.6.9.5.10.23 PLIB_I2C_ExistsSlaveAddressDetect Function**C**

```
bool PLIB_I2C_ExistsSlaveAddressDetect(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveAddressDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveAddress10BitWasDetected](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveAddressDetect feature is supported on the device
- false - If the SlaveAddressDetect feature is not supported on the device

Remarks

None.

5.6.9.5.10.24 PLIB_I2C_ExistsSlaveAddressHoldEnable Function**C**

```
bool PLIB_I2C_ExistsSlaveAddressHoldEnable(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveAddressHoldEnable feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveAddressHoldEnable](#)
- [PLIB_I2C_SlaveAddressHoldDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveAddressHoldEnable feature is supported on the device
- false - If the SlaveAddressHoldEnable feature is not supported on the device

Remarks

None.

5.6.9.5.10.25 PLIB_I2C_ExistsSlaveBufferOverwrite Function**C**

```
bool PLIB_I2C_ExistsSlaveBufferOverwrite(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveBufferOverwrite feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveBufferOverwriteEnable](#)
- [PLIB_I2C_SlaveBufferOverwriteDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveBufferOverwrite feature is supported on the device
- false - If the SlaveBufferOverwrite feature is not supported on the device

Remarks

None.

5.6.9.5.10.26 PLIB_I2C_ExistsSlaveBusCollisionDetect Function**C**

```
bool PLIB_I2C_ExistsSlaveBusCollisionDetect(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveBusCollisionDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveBusCollisionDetectEnable](#)
- [PLIB_I2C_SlaveBusCollisionDetectDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveBusCollisionDetect feature is supported on the device
- false - If the SlaveBusCollisionDetect feature is not supported on the device

Remarks

None.

5.6.9.5.10.27 PLIB_I2C_ExistsSlaveClockHold Function**C**

```
bool PLIB_I2C_ExistsSlaveClockHold(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveClockHold feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveClockHold](#)
- [PLIB_I2C_SlaveClockRelease](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveClockHold feature is supported on the device
- false - If the SlaveClockHold feature is not supported on the device

Remarks

None.

5.6.9.5.10.28 PLIB_I2C_ExistsSlaveDataDetect Function**C**

```
bool PLIB_I2C_ExistsSlaveDataDetect(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveDataDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveDataIsDetected](#)
- [PLIB_I2C_SlaveAddressIsDetected](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveDataDetect feature is supported on the device
- false - If the SlaveDataDetect feature is not supported on the device

Remarks

None.

5.6.9.5.10.29 PLIB_I2C_ExistsSlaveInterruptOnStart Function**C**

```
bool PLIB_I2C_ExistsSlaveInterruptOnStart(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveInterruptOnStart feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveInterruptOnStartEnable](#)
- [PLIB_I2C_SlaveInterruptOnStartDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveInterruptOnStart feature is supported on the device
- false - If the SlaveInterruptOnStart feature is not supported on the device

Remarks

None.

5.6.9.5.10.30 PLIB_I2C_ExistsSlaveInterruptOnStop Function**C**

```
bool PLIB_I2C_ExistsSlaveInterruptOnStop(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveInterruptOnStop feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveInterruptOnStopEnable](#)
- [PLIB_I2C_SlaveInterruptOnStopDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveInterruptOnStop feature is supported on the device
- false - If the SlaveInterruptOnStop feature is not supported on the device

Remarks

None.

5.6.9.5.10.31 PLIB_I2C_ExistsSlaveMask Function

C

```
bool PLIB_I2C_ExistsSlaveMask(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveMask feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveMask7BitSet](#)
- [PLIB_I2C_SlaveMask7BitGet](#)
- [PLIB_I2C_SlaveMask10BitSet](#)
- [PLIB_I2C_SlaveMask10BitGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveMask feature is supported on the device
- false - If the SlaveMask feature is not supported on the device

Remarks

None.

5.6.9.5.10.32 PLIB_I2C_ExistsSlaveReadRequest Function

C

```
bool PLIB_I2C_ExistsSlaveReadRequest(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveReadRequest feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SlaveReadIsRequested](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SlaveReadRequest feature is supported on the device
- false - If the SlaveReadRequest feature is not supported on the device

Remarks

None.

5.6.9.5.10.33 PLIB_I2C_ExistsSMBus Function**C**

```
bool PLIB_I2C_ExistsSMBus(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the SMBus feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_SMBEnable](#)
- [PLIB_I2C_SMBDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SMBus feature is supported on the device
- false - If the SMBus feature is not supported on the device

Remarks

None.

5.6.9.5.10.34 PLIB_I2C_ExistsStartDetect Function**C**

```
bool PLIB_I2C_ExistsStartDetect(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the StartDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_StartWasDetected](#)
- [PLIB_I2C_StartClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the StartDetect feature is supported on the device
- false - If the StartDetect feature is not supported on the device

Remarks

None.

5.6.9.5.10.35 PLIB_I2C_ExistsStopDetect Function**C**

```
bool PLIB_I2C_ExistsStopDetect(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the StopDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_StopWasDetected](#)
- [PLIB_I2C_StopClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the StopDetect feature is supported on the device
- false - If the StopDetect feature is not supported on the device

Remarks

None.

5.6.9.5.10.36 PLIB_I2C_ExistsStopInIdle Function**C**

```
bool PLIB_I2C_ExistsStopInIdle(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_StopInIdleEnable](#)
- [PLIB_I2C_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the StopInIdle feature is supported on the device
- false - If the StopInIdle feature is not supported on the device

Remarks

None.

5.6.9.5.10.37 PLIB_I2C_ExistsTransmitterByteAcknowledge Function**C**

```
bool PLIB_I2C_ExistsTransmitterByteAcknowledge(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterByteAcknowledge feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_TransmitterByteWasAcknowledged](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TransmitterByteAcknowledge feature is supported on the device
- false - If the TransmitterByteAcknowledge feature is not supported on the device

Remarks

None.

5.6.9.5.10.38 PLIB_I2C_ExistsTransmitterByteComplete Function**C**

```
bool PLIB_I2C_ExistsTransmitterByteComplete(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterByteComplete feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_TransmitterByteHasCompleted](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TransmitterByteComplete feature is supported on the device
- false - If the TransmitterByteComplete feature is not supported on the device

Remarks

None.

5.6.9.5.10.39 PLIB_I2C_ExistsTransmitterByteSend Function**C**

```
bool PLIB_I2C_ExistsTransmitterByteSend(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterByteSend feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_TransmitterByteSend](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TransmitterByteSend feature is supported on the device
- false - If the TransmitterByteSend feature is not supported on the device

Remarks

None.

5.6.9.5.10.40 PLIB_I2C_ExistsTransmitterIsBusy Function**C**

```
bool PLIB_I2C_ExistsTransmitterIsBusy(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterBusy feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_TransmitterIsBusy](#)
- [PLIB_I2C_TransmitterIsReady](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TransmitterBusy feature is supported on the device
- false - If the TransmitterBusy feature is not supported on the device

Remarks

None.

5.6.9.5.10.41 PLIB_I2C_ExistsTransmitterOverflow Function**C**

```
bool PLIB_I2C_ExistsTransmitterOverflow(  
    I2C_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterOverflow feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- [PLIB_I2C_TransmitterOverflowHasOccurred](#)
- [PLIB_I2C_TransmitterOverflowClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TransmitterOverflow feature is supported on the device
- false - If the TransmitterOverflow feature is not supported on the device

Remarks

None.

5.6.9.5.11 Data Types and Constants**5.6.9.5.11.1 I2C_MODULE_ID Enumeration****C**

```
typedef enum {  
    I2C_ID_1,  
    I2C_ID_2,  
    I2C_ID_3,  
    I2C_ID_4,  
};
```

```

    I2C_ID_5,
    I2C_NUMBER_OF_MODULES
} I2C_MODULE_ID;

```

Description

I2C Module ID

This enumeration identifies the available I2C modules.

Members

Members	Description
I2C_ID_1	I2C Module 1 ID
I2C_ID_2	I2C Module 2 ID
I2C_ID_3	I2C Module 3 ID
I2C_ID_4	I2C Module 4 ID
I2C_ID_5	I2C Module 5 ID
I2C_NUMBER_OF_MODULES	Number of available I2C modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers specified in the data sheet.

5.6.9.6 Files

Files

Name	Description
plib_i2c.h	This file contains the interface definition for the I2C Peripheral Library.

Description

5.6.9.6.1 plib_i2c.h

I2C Peripheral Library Interface Header

This library provides a low-level abstraction of the Inter-Integrated Circuit (I2C) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

Functions

	Name	Description
≡	PLIB_I2C_AcksequenceInProgress	Checks whether the acknowledge sequence is in progress or it is completed.
≡	PLIB_I2C_ArbitrationLossClear	Clears the arbitration loss status flag
≡	PLIB_I2C_ArbitrationLossHasOccurred	Identifies if bus arbitration has been lost.
≡	PLIB_I2C_BaudRateGet	Calculates the I2C module's current SCL clock frequency.
≡	PLIB_I2C_BaudRateSet	Sets the desired baud rate.

	PLIB_I2C_BusIdle	Determines whether the I2C bus is idle or busy.
	PLIB_I2C_DataLineHoldTimeSet	Sets the data line hold time.
	PLIB_I2C_Disable	Disables the specified I2C module.
	PLIB_I2C_Enable	Enables the specified I2C module.
	PLIB_I2C_ExistsAckSequenceProgress	Identifies whether the AckSequenceInProgress feature exists on the I2C module
	PLIB_I2C_ExistsArbitrationLoss	Identifies whether the ArbitrationLoss feature exists on the I2C module
	PLIB_I2C_ExistsBaudRate	Identifies whether the BaudRate feature exists on the I2C module
	PLIB_I2C_ExistsBusIdle	Identifies whether the BusIdle feature exists on the I2C module
	PLIB_I2C_ExistsClockStretching	Identifies whether the ClockStretching feature exists on the I2C module
	PLIB_I2C_ExistsDataLineHoldTime	Identifies whether the DataLineHoldTime feature exists on the I2C module
	PLIB_I2C_ExistsGeneralCall	Identifies whether the GeneralCall feature exists on the I2C module
	PLIB_I2C_ExistsGeneralCallAddressDetect	Identifies whether the GeneralCallAddressDetect feature exists on the I2C module
	PLIB_I2C_ExistsHighFrequency	Identifies whether the HighFrequency feature exists on the I2C module
	PLIB_I2C_ExistsIPMI	Identifies whether the IPMI feature exists on the I2C module
	PLIB_I2C_ExistsMasterReceiverClock1Byte	Identifies whether the MasterReceiverClock1Byte feature exists on the I2C module
	PLIB_I2C_ExistsMasterStart	Identifies whether the MasterStart feature exists on the I2C module
	PLIB_I2C_ExistsMasterStartRepeat	Identifies whether the MasterStartRepeat feature exists on the I2C module
	PLIB_I2C_ExistsMasterStop	Identifies whether the MasterStop feature exists on the I2C module
	PLIB_I2C_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteAcknowledge	Identifies whether the ReceivedByteAcknowledge feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteAvailable	Identifies whether the ReceivedByteAvailable feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteGet	Identifies whether the ReceivedByteGet feature exists on the I2C module
	PLIB_I2C_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the I2C module
	PLIB_I2C_ExistsReservedAddressProtect	Identifies whether the ReservedAddressProtect feature exists on the I2C module
	PLIB_I2C_ExistsSlaveAddress10Bit	Identifies whether the SlaveAddress10Bit feature exists on the I2C module
	PLIB_I2C_ExistsSlaveAddress7Bit	Identifies whether the SlaveAddress7Bit feature exists on the I2C module
	PLIB_I2C_ExistsSlaveAddressDetect	Identifies whether the SlaveAddressDetect feature exists on the I2C module
	PLIB_I2C_ExistsSlaveAddressHoldEnable	Identifies whether the SlaveAddressHoldEnable feature exists on the I2C module

	PLIB_I2C_ExistsSlaveBufferOverwrite	Identifies whether the SlaveBufferOverwrite feature exists on the I2C module
	PLIB_I2C_ExistsSlaveBusCollisionDetect	Identifies whether the SlaveBusCollisionDetect feature exists on the I2C module
	PLIB_I2C_ExistsSlaveClockHold	Identifies whether the SlaveClockHold feature exists on the I2C module
	PLIB_I2C_ExistsSlaveDataDetect	Identifies whether the SlaveDataDetect feature exists on the I2C module
	PLIB_I2C_ExistsSlaveInterruptOnStart	Identifies whether the SlaveInterruptOnStart feature exists on the I2C module
	PLIB_I2C_ExistsSlaveInterruptOnStop	Identifies whether the SlaveInterruptOnStop feature exists on the I2C module
	PLIB_I2C_ExistsSlaveMask	Identifies whether the SlaveMask feature exists on the I2C module
	PLIB_I2C_ExistsSlaveReadRequest	Identifies whether the SlaveReadRequest feature exists on the I2C module
	PLIB_I2C_ExistsSMBus	Identifies whether the SMBus feature exists on the I2C module
	PLIB_I2C_ExistsStartDetect	Identifies whether the StartDetect feature exists on the I2C module
	PLIB_I2C_ExistsStopDetect	Identifies whether the StopDetect feature exists on the I2C module
	PLIB_I2C_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the I2C module
	PLIB_I2C_ExistsTransmitterByteAcknowledge	Identifies whether the TransmitterByteAcknowledge feature exists on the I2C module
	PLIB_I2C_ExistsTransmitterByteComplete	Identifies whether the TransmitterByteComplete feature exists on the I2C module
	PLIB_I2C_ExistsTransmitterByteSend	Identifies whether the TransmitterByteSend feature exists on the I2C module
	PLIB_I2C_ExistsTransmitterIsBusy	Identifies whether the TransmitterBusy feature exists on the I2C module
	PLIB_I2C_ExistsTransmitterOverflow	Identifies whether the TransmitterOverflow feature exists on the I2C module
	PLIB_I2C_GeneralCallDisable	Disables the I2C module from recognizing the general call address.
	PLIB_I2C_GeneralCallEnable	Enables the I2C module to recognize the general call address.
	PLIB_I2C_HighFrequencyDisable	Disables the I2C module from using high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_HighFrequencyEnable	Enables the I2C module to use high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_IPMIDisable	Disables the I2C module's support for the IPMI specification
	PLIB_I2C_IPMIEnable	Enables the I2C module to support the Intelligent Platform Management Interface (IPMI) specification (see Remarks).
	PLIB_I2C_MasterReceiverClock1Byte	Drives the bus clock to receive 1 byte of data from a slave device.
	PLIB_I2C_MasterReceiverReadyToAcknowledge	Checks whether the hardware is ready to acknowledge.
	PLIB_I2C_MasterStart	Sends an I2C Start condition on the I2C bus in Master mode.
	PLIB_I2C_MasterStartRepeat	Sends a repeated Start condition during an ongoing transfer in Master mode.

	PLIB_I2C_MasterStop	Sends an I2C Stop condition to terminate a transfer in Master mode.
	PLIB_I2C_ReceivedByteAcknowledge	Allows a receiver to acknowledge a that a byte of data has been received.
	PLIB_I2C_ReceivedByteGet	Gets a byte of data received from the I2C bus interface.
	PLIB_I2C_ReceivedByteIsAvailable	Detects whether the receiver has data available.
	PLIB_I2C_ReceiverByteAcknowledgeHasCompleted	Determines if the previous acknowledge has completed.
	PLIB_I2C_ReceiverOverflowClear	Clears the receiver overflow status flag.
	PLIB_I2C_ReceiverOverflowHasOccurred	Identifies if a receiver overflow error has occurred.
	PLIB_I2C_ReservedAddressProtectDisable	Disables the I2C module from protecting reserved addresses, allowing it to respond to them.
	PLIB_I2C_ReservedAddressProtectEnable	Enables the I2C module to protect (not respond to) reserved addresses.
	PLIB_I2C_SlaveAddress10BitGet	Identifies the current 10-bit Slave mode address.
	PLIB_I2C_SlaveAddress10BitSet	Selects 10-bit Slave mode addressing and sets the address value.
	PLIB_I2C_SlaveAddress10BitWasDetected	Detects reception of the second byte of a 10-bit slave address.
	PLIB_I2C_SlaveAddress7BitGet	Identifies the current 7-bit Slave mode address.
	PLIB_I2C_SlaveAddress7BitSet	Selects 7-bit Slave mode addressing and sets the slave address value.
	PLIB_I2C_SlaveAddressHoldDisable	Disables Address holding.
	PLIB_I2C_SlaveAddressHoldEnable	Enables address holding.
	PLIB_I2C_SlaveAddressIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveAddressIsGeneralCall	Identifies if the current slave address received is the general call address.
	PLIB_I2C_SlaveAddressModels10Bits	Identifies if the current slave address mode is 7-bits or 10-bits.
	PLIB_I2C_SlaveBufferOverwriteDisable	Disables buffer overwrite.
	PLIB_I2C_SlaveBufferOverwriteEnable	Enables buffer overwrite.
	PLIB_I2C_SlaveBusCollisionDetectDisable	Disables bus collision detect interrupts.
	PLIB_I2C_SlaveBusCollisionDetectEnable	Enables slave bus collision interrupts.
	PLIB_I2C_SlaveClockHold	Holds the SCL clock line low after receiving a byte.
	PLIB_I2C_SlaveClockRelease	Releases a previously held SCL clock line.
	PLIB_I2C_SlaveClockStretchingDisable	Disables the I2C module from stretching the slave clock.
	PLIB_I2C_SlaveClockStretchingEnable	Enables the I2C module to stretch the slave clock.
	PLIB_I2C_SlaveDataHoldDisable	Disables data holding.
	PLIB_I2C_SlaveDataHoldEnable	Enables data holding.
	PLIB_I2C_SlaveDataIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveInterruptOnStartDisable	Disables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStartEnable	Enables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStopDisable	Disables the feature of generating interrupt on stop condition.
	PLIB_I2C_SlaveInterruptOnStopEnable	Enables the feature of generating interrupt on stop condition.
	PLIB_I2C_SlaveMask10BitGet	Identifies the current 10-bit Slave mode address mask.
	PLIB_I2C_SlaveMask10BitSet	Sets the 10-bit mask for Slave mode addressing.
	PLIB_I2C_SlaveMask7BitGet	Identifies the current 7-bit Slave mode address mask.

	PLIB_I2C_SlaveMask7BitSet	Sets the 7-bit mask for Slave mode addressing .
	PLIB_I2C_SlaveReadIsRequested	Detects if the request from the master was a read or write.
	PLIB_I2C_SMBDisable	Disable the I2C module support for SMBus electrical signaling levels.
	PLIB_I2C_SMBEnable	Enables the I2C module to support System Management Bus (SMBus) electrical signaling levels.
	PLIB_I2C_StartClear	Clears the start status flag
	PLIB_I2C_StartWasDetected	Identifies when a Start condition has been detected.
	PLIB_I2C_StopClear	Clears the stop status flag
	PLIB_I2C_StopInIdleDisable	Disables the Stop-in-Idle feature.
	PLIB_I2C_StopInIdleEnable	Enables the I2C module to stop when the processor enters Idle mode
	PLIB_I2C_StopWasDetected	Identifies when a Stop condition has been detected
	PLIB_I2C_TransmitterByteHasCompleted	Detects whether the module has finished transmitting the most recent byte.
	PLIB_I2C_TransmitterByteSend	Sends a byte of data on the I2C bus.
	PLIB_I2C_TransmitterByteWasAcknowledged	Determines whether the most recently sent byte was acknowledged.
	PLIB_I2C_TransmitterIsBusy	Identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).
	PLIB_I2C_TransmitterIsReady	Detects if the transmitter is ready to accept data to transmit.
	PLIB_I2C_TransmitterOverflowClear	Clears the transmitter overflow status flag.
	PLIB_I2C_TransmitterOverflowHasOccurred	Identifies if a transmitter overflow error has occurred.

File Name

plib_i2c.h

Company

Microchip Technology Inc.

5.6.10 Input Capture Peripheral Library

5.6.10.1 Introduction

Input Capture Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Input Capture (IC) Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The Input Capture (IC) module is used to capture a timer value on the occurrence of an event on an input pin. The IC module plays a major role in applications requiring frequency measurement.

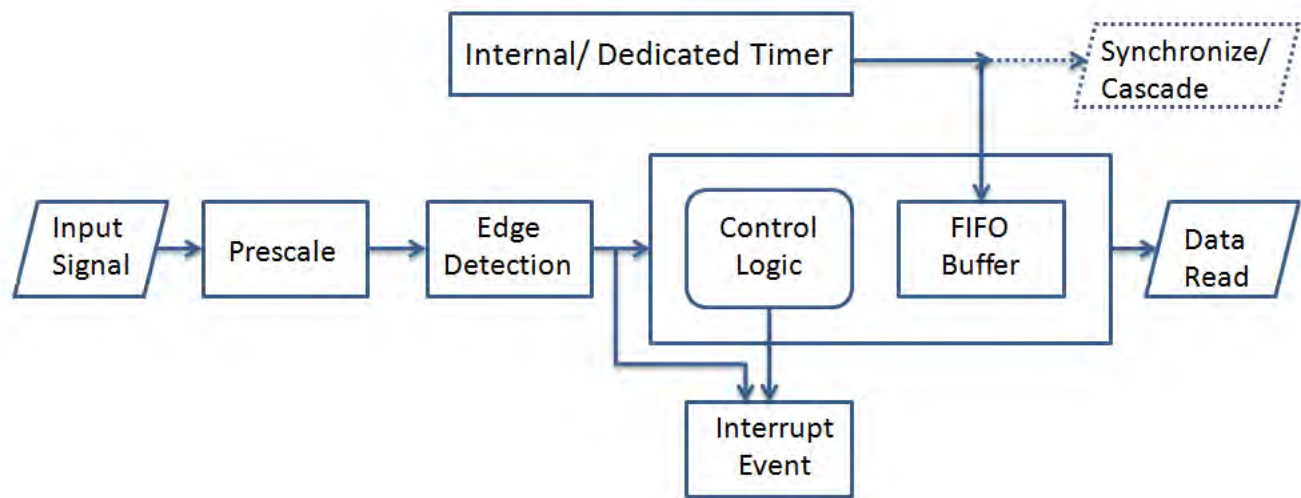


Figure 1: Input Capture Module

Figure 1 shows the functionality of an Input Capture module. An input signal is provided to the input of this module. The input may be prescaled. An edge detection block detects if the input is undergoing a rising or falling transition. Control logic undertakes the capture of the timer value depending upon the operating mode selected.

Some devices have Input Capture modules with dedicated timers that can be used to synchronize multiple IC modules together or cascade 16-bit timers to obtain a 32-bit timer. The timer may be triggered by an external, asynchronous input to start counting as well.

The IC module has multiple modes of operation which can be used for different applications. Operating modes include the following:

- Timer value capture at falling edge of input signal
- Timer value capture at rising edge of input signal
- Timer value capture at both edges of input signal
- Timer value capture on a specified edge of the input signal and every edge thereafter
- Timer value capture on every fourth or sixteenth rising edge of the input signal
- Interrupt-only mode in which Input Capture module acts as a source of external interrupt. This mode is functional only in device sleep and idle modes.

The IC module also incorporates a First In First Out (FIFO) buffer, which temporarily holds the captured value until it is read by the user application. The IC module is capable of generating interrupts when a set number of captures have taken place.

5.6.10.2 Release Notes

MPLAB Harmony Version: v0.70b **IC Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.10.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.10.4 Using the Library

This topic describes the basic architecture of the IC Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_ic.h](#)

The interface to the IC Peripheral library is defined in the "[plib_ic.h](#)" header file. This file is included by the "peripheral.h" file. Any C language source (.c) file that uses the IC Peripheral library should include "peripheral.h".

Library File: The IC Peripheral library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the Peripheral interacts with the framework.

5.6.10.4.1 Hardware Abstraction Model

This library provides the low-level abstraction of the Input Capture (IC) module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all of the functionality of the available IC module on the device. Refer to the specific device data sheet or related family reference manual section to determine the set of functions that are supported for each IC module on your device.

Description

The IC Peripheral Library is used to simplify low-level access to the IC module without having the need to directly interact with the module registers. The names of the functions are generic and lead to easier access to the IC module on different device variants.

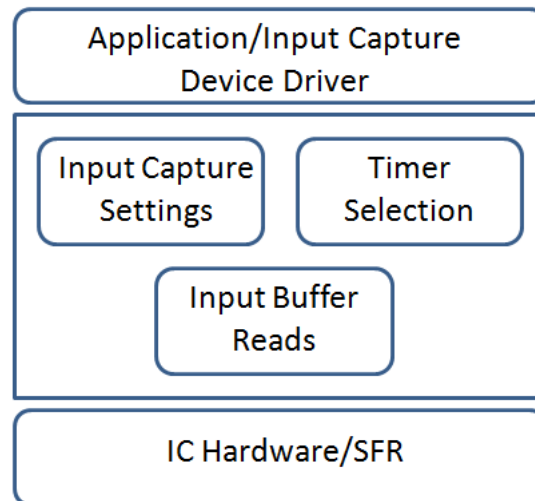


Figure 2: Input Capture module Software Abstraction Block Diagram

The IC module can be described from a software standpoint as having functions to configure the module itself and select an intended mode of operation. Separate functions exist to select the timer whose value is captured by the IC module. When an external input event occurs, the IC module captures the timer value and stores it in a First In First Out (FIFO) buffer. The captured value can be read from the FIFO buffer by performing an input buffer read by calling the respective function.

5.6.10.4.2 How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

All of the interface functions in this library are classified according to:

- General setup functions: These functions deal with general setup of the IC module
- Timer functions: These routines are used to select the timer, input for the timer, and configure synchronous or triggered operation for the timer
- Operation Modes routine: The [PLIB_IC_ModeSelect](#) function is used to select an operation mode of IC module
- Input capture buffer routines: These routines deal with reading the capture value stored in the IC buffer and performing status checks on the IC buffer
- Power-Saving modes:
 - In Sleep mode, the IC module can function only as an external interrupt source. A rising edge input wakes the device from sleep. An interrupt will be generated if it is enabled for that module.
 - In Idle mode, the IC module can be configured to either stop its operation or continue its operation
- Exists functions: These functions determine whether or not a particular feature is present on a device

Subsequent sections provide examples that depict the use of interface routines to perform general tasks such as IC module setup and reading captured values.

5.6.10.4.2.1 Input Capture Module Setup

This section shows the steps required to initialize and configure the IC module. The IC module is configured such that a capture event occurs at the first falling edge of the input signal, and then at every subsequent edge of the input signal.

Example:

```

/* This example shows how to initialize and configure IC module */
/* Disable IC module */
PLIB_IC_Disable(MY_IC_ID);

/* Disable any peripherals multiplexed with Input Capture pin */

/* Configure I/O pin as input pin. Refer PORTS Peripheral Library */

/* Stop in Idle mode is disabled for IC module. IC mode functions even in Idle mode */
PLIB_IC_StopInIdleDisable(MY_IC_ID);

/* Every Edge Capture mode of operation is selected for IC module */
PLIB_IC_ModeSelect(MY_IC_ID, IC_INPUT_CAPTURE_EVERY_EDGE_MODE);

/* Falling edge capture is set for IC module. IC module captures the timer value at the first falling edge of input signal and subsequently at every edge */
PLIB_IC_FirstEdgeCaptureSet(MY_IC_ID, IC_EDGE_FALLING);

/* 32-bit timer is selected*/
PLIB_IC_BufferSizeSelect(MY_IC_ID, IC_BUFFER_SIZE_32BIT);

/* IC module is enabled */
PLIB_IC_Enable(MY_IC_ID);

```

5.6.10.4.2.2 Reading a Capture Value

This section shows how to read a capture value from the input capture buffer.

Example:

```

/* This example shows how to read a Capture value */
uint32_t bufferVal;

/* Reading a value from an empty Capture Buffer leads to indeterminate data*/
/* Read value from a non-empty buffer */
if(!PLIB_IC_BufferIsEmpty(MY_IC_ID))
{
    /* Read the buffer value */
    bufferVal = PLIB_IC_Buffer32BitGet(MY_IC_ID);
}

/* Buffer Overflow condition */
while(PLIB_IC_BufferOverflowHasOccurred)
{
    /*Read buffer values till buffer is empty and overflow condition ceases*/
    bufferVal = PLIB_IC_Buffer32BitGet(MY_IC_ID);
}

```

5.6.10.4.2.3 Synchronizing Timer

This section shows how to enable synchronous IC timer operation for the input capture buffer. The timer of IC2 is used as the synchronization source for the IC1 module.

Example:

```

/* This example shows how to enable synchronous timer operation. Timer of IC1 is operated in synchronism with timer of IC2 */
#define MY_IC_ID1 IC_ID_1

#define MY_IC_ID2 IC_ID_2

/* Disable IC2 module */
PLIB_IC_Disable(MY_IC_ID2);

/* Disable IC1 module */

```

```

PLIB_IC_Disable(MY_IC_ID1);

/* Select synchronous mode of operation for IC1 */
PLIB_IC_SyncModeSelect(MY_IC_ID1, IC_MODE_SYNC);

/* Timer of IC2 is selected as synchronizing input */
PLIB_IC_SyncModeInputSelect(MY_IC_ID1, IC_SYNC_MODE_INPUT_ICAP2);

/*Timer-2 is selected as clock source for IC1 timer. It is important that
clock source should match with the clock source of synchronizing input*/
PLIB_IC_ClockSourceSelect(MY_IC_ID1, IC_CLOCK_SOURCE_TIMER2);

/* Select synchronous mode of operation for IC2 */
PLIB_IC_SyncModeSelect(MY_IC_ID2, IC_MODE_SYNC);

/*Timer-2 is selected as clock source for IC2 timer*/
PLIB_IC_ClockSourceSelect(MY_IC_ID2, IC_CLOCK_SOURCE_TIMER2);

/* Disable any peripherals multiplexed with IC1 pin */
/* Disable any peripherals multiplexed with IC2 pin */
/* Configure I/O pin as input pin for IC1. Refer PORTS Peripheral Library */
/* Configure I/O pin as input pin for IC2. Refer PORTS Peripheral Library */
/* IC1 module is enabled. It is important that IC2 is enabled last as IC2 timer
is being used as the source of synchronization */
PLIB_IC_Enable(MY_IC_ID1);

/* IC2 module is enabled. It is important that IC2 is enabled last as IC2 timer
is being used as the source of synchronization */
PLIB_IC_Enable(MY_IC_ID2);

```

5.6.10.4.3 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.10.5 Library Interface

Data Types and Constants

	Name	Description
	<u>IC_BUFFER_SIZE</u>	Selects the IC Buffer size.
	<u>IC_CLOCK_SOURCES</u>	Lists all of the input capture modes for the IC module.
	<u>IC_EDGE_TYPES</u>	Lists the options to select a rising or falling edge for input capture.
	<u>IC_EVENTS_PER_INTERRUPT</u>	Lists the number of input capture events for an interrupt to occur.
	<u>IC_INPUT_CAPTURE_MODES</u>	Lists all of the input capture modes for the IC module.
	<u>IC_MODULE_ID</u>	IC_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller. Refer to the specific device data sheet to determine the correct number of modules defined for the device.
	<u>IC_SYNC_MODE_INPUTS</u>	Lists the input sources for synchronous and asynchronous operation of the IC module.
	<u>IC_SYNC_MODES</u>	Lists the synchronization modes for the IC module.

	IC_TIMERS	Lists the different 16-bit timers for the IC module.

Feature Existence Functions

	Name	Description
	PLIB_IC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the IC module
	PLIB_IC_ExistsBufferIsEmptyStatus	Identifies whether the BufferIsEmptyStatus feature exists on the IC module
	PLIB_IC_ExistsBufferOverflowStatus	Identifies whether the BufferOverflowStatus feature exists on the IC module.
	PLIB_IC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the IC module.
	PLIB_IC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the IC module.
	PLIB_IC_ExistsCaptureMode	Identifies whether the CaptureMode feature exists on the IC module.
	PLIB_IC_ExistsClockSource	Identifies whether the ClockSource feature exists on the IC module.
	PLIB_IC_ExistsEdgeCapture	Identifies whether the EdgeCapture feature exists on the IC module.
	PLIB_IC_ExistsEnable	Identifies whether the EnableControl feature exists on the IC module
	PLIB_IC_ExistsEventsPerInterruptSelect	Identifies whether the EventsPerInterruptSelect feature exists on the IC module.
	PLIB_IC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the IC module.
	PLIB_IC_ExistsSyncMode	Identifies whether the SyncMode feature exists on the IC module.
	PLIB_IC_ExistsSyncModeInput	Identifies whether the SyncModeInput feature exists on the IC module.
	PLIB_IC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the IC module.
	PLIB_IC_ExistsTimerTriggered	Identifies whether the TimerTriggered feature exists on the IC module.

General Setup Functions

	Name	Description
	PLIB_IC_Disable	Disables the IC module.
	PLIB_IC_Enable	Enables the IC module.
	PLIB_IC_EventsPerInterruptSelect	Selects the number of capture events before an interrupt is issued.
	PLIB_IC_FirstCaptureEdgeSelect	Captures the timer count value at the first selected edge of input signal.

Input Capture Buffer Functions

	Name	Description
	PLIB_IC_Buffer16BitGet	Obtains the 16-bit input capture buffer value.
	PLIB_IC_Buffer32BitGet	Obtains the 32-bit input capture buffer value.
	PLIB_IC_BufferIsEmpty	Checks whether the input capture buffer is empty.
	PLIB_IC_BufferOverflowHasOccurred	Checks whether an input capture buffer overflow has occurred.
	PLIB_IC_BufferSizeSelect	Sets the input capture buffer size.

Operation Modes Functions

	Name	Description
	PLIB_IC_ModeSelect	Selects the input capture mode for IC module.

Power-Saving Modes Functions

	Name	Description
	PLIB_IC_StopInIdleDisable	Continues module operation when the device enters Idle mode
	PLIB_IC_StopInIdleEnable	Discontinues IC module operation when the device enters Idle mode.

Timer Functions

	Name	Description
≡	PLIB_IC_ClockSourceSelect	Selects the clock source for the IC module.
≡	PLIB_IC_SyncModeInputSelect	Selects the input source for synchronous or asynchronous (triggered) operation of the IC module.
≡	PLIB_IC_SyncModeSelect	Selects the synchronization mode for Input Capture module.
≡	PLIB_IC_TimerIsTriggered	Checks whether or not the input capture timer has been triggered.
≡	PLIB_IC_TimerSelect	Selects the 16-bit timer source for the IC module.
≡	PLIB_IC_AlternateClockDisable	Selects Timer 2/3 instead of the alternate clock source.
≡	PLIB_IC_AlternateClockEnable	Selects the alternate clock source.

Description

This section describes the Application Programming Interface (API) functions of the Input Capture Peripheral.

Refer to each section for a detailed description.

5.6.10.5.1 General Setup Functions**5.6.10.5.1.1 PLIB_IC_Disable Function****C**

```
void PLIB_IC_Disable(
    IC_MODULE_ID index
);
```

Description

This function disables the IC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the IC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_Disable(MY_IC_ID);
```

5.6.10.5.1.2 PLIB_IC_Enable Function**C**

```
void PLIB_IC_Enable(
    IC_MODULE_ID index
```

```
);
```

Description

This function enables the IC module.

Preconditions

The module should be appropriately configured before being enabled.

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1

//Do all the other configurations before enabling.

PLIB_IC_Enable(MY_IC_ID);
```

5.6.10.5.1.3 PLIB_IC_EventsPerInterruptSelect Function

C

```
void PLIB_IC_EventsPerInterruptSelect(
    IC_MODULE_ID index,
    IC_EVENTS_PER_INTERRUPT event
);
```

Description

The IC module can be configured to generate interrupts depending upon the the occurrence of a certain number of capture events. The number of capture events before an interrupt is generated is set by this function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module
event	Identifies the interrupt control mode

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsEventsPerInterruptSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
// IC module is configured to generate interrupt on every capture event
```

```
PLIB_IC_EventsPerInterruptSelect(MY_IC_ID, IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT );
```

5.6.10.5.1.4 PLIB_IC_FirstCaptureEdgeSelect Function

C

```
void PLIB_IC_FirstCaptureEdgeSelect(  
    IC_MODULE_ID index,  
    IC_EDGE_TYPES edgeType  
);
```

Description

This function captures the timer count value at the first selected edge of the input signal.

Preconditions

This setting applies only when the Every Edge Capture mode is set. The capture mode is set by the [PLIB_IC_ModeSelect\(\)](#) function.

Parameters

Parameters	Description
index	Identifies the desired IC module
edgeType	Identifies the edge type (i.e., whether rising or falling edge)

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsEdgeCapture](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1  
  
PLIB_IC_FirstCaptureEdgeSelect(MY_IC_ID, IC_EDGE_FALLING);
```

5.6.10.5.2 Timer Functions

5.6.10.5.2.1 PLIB_IC_ClockSourceSelect Function

C

```
void PLIB_IC_ClockSourceSelect(  
    IC_MODULE_ID index,  
    IC_CLOCK_SOURCES src  
);
```

Description

This function selects the clock source for the IC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module
src	Identifies the clock source

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsClockSource](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
// Clock source of Timer2 is selected as clock source for Input Capture module
PLIB_IC_ClockSourceSelect(MY_IC_ID, IC_CLOCK_SOURCE_TIMER2);
```

5.6.10.5.2 PLIB_IC_SyncModeInputSelect Function**C**

```
void PLIB_IC_SyncModeInputSelect(
    IC_MODULE_ID index,
    IC_SYNC_MODE_INPUTS input
);
```

Description

This function selects the input source for synchronous or asynchronous (triggered) operation of the IC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsSyncModeInput](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
//
PLIB_IC_SyncModeInputSelect(MY_IC_ID, IC_SYNC_MODE_INPUT_OC1);
```

5.6.10.5.2.3 PLIB_IC_SyncModeSelect Function**C**

```
void PLIB_IC_SyncModeSelect(
    IC_MODULE_ID index,
    IC_SYNC_MODES syncMode
);
```

Description

This function selects the synchronization mode for the IC module. The Synchronous or Asynchronous mode of operation can be selected for the IC module timer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module
syncMode	Identifies the mode (Synchronous or Asynchronous)

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsSyncMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
// External input triggers Input Capture module asynchronously
PLIB_IC_SyncModeSelect(MY_IC_ID, IC_MODE_ASYNC);
```

5.6.10.5.2.4 PLIB_IC_TimerIsTriggered Function**C**

```
bool PLIB_IC_TimerIsTriggered(
    IC_MODULE_ID index
);
```

Description

This function checks whether or not the input capture timer has been triggered. This function returns 'true' if the IC timer is triggered and running and returns 'false' if the IC timer is cleared and stopped.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

- bool - true - The IC timer is triggered and running
- false - The IC timer is stopped and held in clear

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsTimerTriggered](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
if(!PLIB_IC_TimerIsTriggered(MY_IC_ID))
{
    // Do some operation
};
```

5.6.10.5.2.5 PLIB_IC_TimerSelect Function**C**

```
void PLIB_IC_TimerSelect(
```

```
    IC_MODULE_ID index,  
    IC_TIMERS tmr  
);
```

Description

This function selects the 16-bit timer source for the IC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module
tmr	Identifies the 16-bit timer

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsTimerSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1  
// 16-bit Timer-2 is selected  
PLIB_IC_TimerSelect(MY_IC_ID, IC_TIMER_TMR2);
```

5.6.10.5.2.6 PLIB_IC_AlternateClockDisable Function**C**

```
void PLIB_IC_AlternateClockDisable(  
    IC_MODULE_ID index  
);
```

Description

Selects Timer 2/3 instead of the alternate clock source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

None.

Remarks

The feature is not supported on all devices. Check your device's data sheet. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the [IC_MODULE_ID](#) passed in the call.

Example

```
// Call system service to unlock oscillator  
#define MY_IC_ID IC_ID_1  
PLIB_IC_AlternateClockDisable( MY_IC_ID );
```

5.6.10.5.2.7 PLIB_IC_AlternateClockEnable Function

C

```
void PLIB_IC_AlternateClockEnable(  
    IC_MODULE_ID index  
);
```

Description

Selects the alternate clock source instead of Timer 2/3.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

None.

Remarks

The feature is not supported on all devices. Check your device's data sheet. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the `IC_MODULE_ID` passed in the call.

Example

```
// Call system service to unlock oscillator  
#define MY_IC_ID IC_ID_1  
PLIB_IC_AlternateClockEnable( MY_IC_ID );
```

5.6.10.5.3 Input Capture Buffer Functions

5.6.10.5.3.1 PLIB_IC_Buffer16BitGet Function

C

```
uint16_t PLIB_IC_Buffer16BitGet(  
    IC_MODULE_ID index  
);
```

Description

This function reads the 16-bit value of the IC buffer to obtain the captured timer value. This function is used when the buffer size is set to 16-bits.

Preconditions

The buffer size should be set to 16 bits (i.e., `PLIB_IC_BufferSizeSet(MY_IC_ID, IC_BUFFER_SIZE_16BIT)`).

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsBufferValue](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
uint16_t bufferVal;
// Read input capture buffer value and store it in 'bufferVal'
bufferVal = PLIB_IC_Buffer16BitGet(MY_IC_ID);
```

5.6.10.5.3.2 PLIB_IC_Buffer32BitGet Function**C**

```
uint32_t PLIB_IC_Buffer32BitGet(
    IC_MODULE_ID index
);
```

Description

This function reads the 32-bit value of the IC buffer to obtain the captured timer value. This function is used when the buffer size is set to 32-bits.

Preconditions

The buffer size should be set to 32 bits (i.e., `PLIB_IC_BufferSizeSet(MY_IC_ID, IC_BUFFER_SIZE_32BIT)`).

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsBufferValue](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
uint32_t bufferVal;
// Read input capture buffer value and store it in 'bufferVal'
bufferVal = PLIB_IC_Buffer32BitGet(MY_IC_ID);
```

5.6.10.5.3.3 PLIB_IC_BufferIsEmpty Function**C**

```
bool PLIB_IC_BufferIsEmpty(
    IC_MODULE_ID index
);
```

Description

This function returns 'true' if the input capture buffer is empty and 'false' if the buffer is not empty.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

- bool - true - The input capture buffer is empty
- false - The input capture buffer is not empty

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsBufferIsEmptyStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
if(!PLIB_IC_BufferIsEmpty(MY_IC_ID))
{
    // Do some operation
};
```

5.6.10.5.3.4 PLIB_IC_BufferOverflowHasOccurred Function**C**

```
bool PLIB_IC_BufferOverflowHasOccurred(
    IC_MODULE_ID index
);
```

Description

This function returns 'true' if an input capture buffer has overflowed and 'false' if the buffer has not overflowed.

Preconditions

This function only applies when not in Edge Detect mode.

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

- bool - true - An overflow of the input capture buffer has occurred
- false - An overflow of the input capture buffer has not occurred

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsBufferOverflowStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
if(!PLIB_IC_BufferOverflowHasOccurred(MY_IC_ID))
{
    // Do some operation
};
```

5.6.10.5.3.5 PLIB_IC_BufferSizeSelect Function**C**

```
void PLIB_IC_BufferSizeSelect(
```

```
    IC_MODULE_ID index,  
    IC_BUFFER_SIZE bufSize  
);
```

Description

This function sets the input capture buffer size for IC module. The buffer size can be set to 16 bits or 32 bits. The buffer size should be consistent with the timer selected. A 32-bit timer requires a 32-bit buffer and a 16-bit timer requires a 16-bit buffer.

Preconditions

The buffer size should be consistent with the timer selected.

Parameters

Parameters	Description
index	Identifies the desired IC module
bufSize	Sets the buffer size to 16 bits or 32 bits

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsBufferSize](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1  
// 32-bit timer resource is selected  
PLIB_IC_BufferSizeSelect(MY_IC_ID, IC_BUFFER_SIZE_32BIT);
```

5.6.10.5.4 Operation Modes Functions

5.6.10.5.4.1 PLIB_IC_ModeSelect Function

C

```
void PLIB_IC_ModeSelect(  
    IC_MODULE_ID index,  
    IC_INPUT_CAPTURE_MODES modeSel  
);
```

Description

This function selects the input capture mode for the IC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module
modeSel	Identifies the timer type required

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_IC_ExistsCaptureMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1
// Every Edge Mode is selected
PLIB_IC_ModeSelect(MY_IC_ID, IC_INPUT_CAPTURE_EVERY_EDGE_MODE);
```

5.6.10.5.5 Power-Saving Modes Functions

5.6.10.5.5.1 PLIB_IC_StopInIdleDisable Function

C

```
void PLIB_IC_StopInIdleDisable(
    IC_MODULE_ID index
);
```

Description

The function continues operation of the IC module when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_StopInIdleDisable(MY_IC_ID);
```

5.6.10.5.5.2 PLIB_IC_StopInIdleEnable Function

C

```
void PLIB_IC_StopInIdleEnable(
    IC_MODULE_ID index
);
```

Description

This function discontinues IC module operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired IC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_IC_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_StopInIdleEnable(MY_IC_ID);
```

5.6.10.5.6 Data Types and Constants

5.6.10.5.6.1 IC_BUFFER_SIZE Enumeration

C

```
typedef enum {
    IC_BUFFER_SIZE_32BIT,
    IC_BUFFER_SIZE_16BIT
} IC_BUFFER_SIZE;
```

Description

IC Buffer Size

The IC buffer can be set to 16 bits or 32 bits. The buffer size is set by the [PLIB_IC_BufferSizeSelect\(\)](#) function.

Members

Members	Description
IC_BUFFER_SIZE_32BIT	32-bit timer source select
IC_BUFFER_SIZE_16BIT	16-bit timer source select

5.6.10.5.6.2 IC_CLOCK_SOURCES Enumeration

C

```
typedef enum {
    IC_CLOCK_SOURCE_TIMER1,
    IC_CLOCK_SOURCE_TIMER2,
    IC_CLOCK_SOURCE_TIMER3,
    IC_CLOCK_SOURCE_TIMER4,
    IC_CLOCK_SOURCE_TIMER5,
    IC_CLOCK_SOURCE_SYSTEM
} IC_CLOCK_SOURCES;
```

Description

IC Clock Sources Select

This enumeration lists all of the input capture modes for IC module. The capture mode is set via the [PLIB_IC_ClockSourceSelect\(\)](#) function.

Members

Members	Description
IC_CLOCK_SOURCE_TIMER1	Clock source of Timer1 is the clock source of the capture counter
IC_CLOCK_SOURCE_TIMER2	Clock source of Timer2 is the clock source of the capture counter

IC_CLOCK_SOURCE_TIMER3	Clock source of Timer3 is the clock source of the capture counter
IC_CLOCK_SOURCE_TIMER4	Clock source of Timer4 is the clock source of the capture counter
IC_CLOCK_SOURCE_TIMER5	Clock source of Timer5 is the clock source of the capture counter
IC_CLOCK_SOURCE_SYSTEM	System Clock is the clock source of the capture counter

5.6.10.5.6.3 IC_EDGE_TYPES Enumeration

C

```
typedef enum {
    IC_EDGE_FALLING,
    IC_EDGE_RISING
} IC_EDGE_TYPES;
```

Description

IC Edge Types Select

This enumeration lists the options to select a rising edge input capture or a falling edge input capture. The selection is made by the [PLIB_IC_FirstCaptureEdgeSelect\(\)](#) function.

Members

Members	Description
IC_EDGE_FALLING	Select Falling Edge
IC_EDGE_RISING	Select Rising Edge

5.6.10.5.6.4 IC_EVENTS_PER_INTERRUPT Enumeration

C

```
typedef enum {
    IC_INTERRUPT_ON_EVERY_4TH_CAPTURE_EVENT,
    IC_INTERRUPT_ON_EVERY_3RD_CAPTURE_EVENT,
    IC_INTERRUPT_ON_EVERY_2ND_CAPTURE_EVENT,
    IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT
} IC_EVENTS_PER_INTERRUPT;
```

Description

IC Capture Events Per Interrupt Select

This enumeration lists the number of events that should occur before an interrupt is generated. An interrupt can occur on every capture event, every second, every third, or every fourth capture event.

Members

Members	Description
IC_INTERRUPT_ON_EVERY_4TH_CAPTURE_EVENT	Interrupt on every 4th capture event
IC_INTERRUPT_ON_EVERY_3RD_CAPTURE_EVENT	Interrupt on every 3rd capture event
IC_INTERRUPT_ON_EVERY_2ND_CAPTURE_EVENT	Interrupt on every 2nd capture event
IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT	Interrupt on every capture event

5.6.10.5.6.5 IC_INPUT_CAPTURE_MODES Enumeration

C

```
typedef enum {
    IC_INPUT_CAPTURE_INTERRUPT_MODE,
    IC_INPUT_CAPTURE_EVERY_EDGE_MODE,
    IC_INPUT_CAPTURE_EVERY_16TH_EDGE_MODE,
    IC_INPUT_CAPTURE_EVERY_4TH_EDGE_MODE,
```

```

    IC_INPUT_CAPTURE_RISING_EDGE_MODE,
    IC_INPUT_CAPTURE_FALLING_EDGE_MODE,
    IC_INPUT_CAPTURE_EDGE_DETECT_MODE,
    IC_INPUT_CAPTURE_DISABLE_MODE
} IC_INPUT_CAPTURE_MODES;

```

Description

IC Input Capture Mode Select

This enumeration lists all of the input capture modes for IC module. The capture mode is set via the [PLIB_IC_ModeSelect\(\)](#) function.

Members

Members	Description
IC_INPUT_CAPTURE_INTERRUPT_MODE	Interrupt only mode: Rising edge on input triggers an interrupt. This mode is used only when device is in Sleep/Idle mode
IC_INPUT_CAPTURE_EVERY_EDGE_MODE	Every Edge Capture mode: The first edge of the input signal is specified via PLIB_IC_RisingEdgeCaptureSet() or PLIB_IC_FallingEdgeCaptureSet() routines. Subsequently, timer count value is captured on every rising and falling of the input signal
IC_INPUT_CAPTURE_EVERY_16TH_EDGE_MODE	Prescaled Capture mode: Timer count value is captured every 16th rising edge of input signal
IC_INPUT_CAPTURE_EVERY_4TH_EDGE_MODE	Prescaled Capture mode: Timer count value is captured every 4th rising edge of input signal
IC_INPUT_CAPTURE_RISING_EDGE_MODE	Rising Edge mode: Timer count value is captured on every rising edge of input signal
IC_INPUT_CAPTURE_FALLING_EDGE_MODE	Falling Edge mode: Timer count value is captured on every falling edge of input signal
IC_INPUT_CAPTURE_EDGE_DETECT_MODE	Edge Detect mode: Timer count value is captured on every rising and falling of the input signal. Interrupt control bits are ignored and an interrupt event is generated for every capture. Overflow status is not updated.
IC_INPUT_CAPTURE_DISABLE_MODE	Capture module is disabled. Input signal is ignored and no capture events or interrupts are generated

5.6.10.5.6.6 IC_MODULE_ID Enumeration

C

```

typedef enum {
    IC_ID_1,
    IC_ID_2,
    IC_ID_3,
    IC_ID_4,
    IC_ID_5,
    IC_ID_6,
    IC_ID_7,
    IC_ID_8,
    IC_ID_9,
    IC_NUMBER_OF_MODULES
} IC_MODULE_ID;

```

Description

IC_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller.

Refer to the specific device data sheet to determine the correct number of modules defined for the device.

Members

Members	Description
IC_NUMBER_OF_MODULES	The total number of modules available.

5.6.10.5.6.7 IC_SYNC_MODE_INPUTS Enumeration**C**

```
typedef enum {
    IC_SYNC_MODE_INPUT_OC1,
    IC_SYNC_MODE_INPUT_OC2,
    IC_SYNC_MODE_INPUT_OC3,
    IC_SYNC_MODE_INPUT_OC4,
    IC_SYNC_MODE_INPUT_OC5,
    IC_SYNC_MODE_INPUT_OC6,
    IC_SYNC_MODE_INPUT_OC7,
    IC_SYNC_MODE_INPUT_OC8,
    IC_SYNC_MODE_INPUT_OC9,
    IC_SYNC_MODE_INPUT_ICAP1,
    IC_SYNC_MODE_INPUT_ICAP2,
    IC_SYNC_MODE_INPUT_ICAP3,
    IC_SYNC_MODE_INPUT_ICAP4,
    IC_SYNC_MODE_INPUT_ICAP5,
    IC_SYNC_MODE_INPUT_ICAP6,
    IC_SYNC_MODE_INPUT_ICAP7,
    IC_SYNC_MODE_INPUT_ICAP8,
    IC_SYNC_MODE_INPUT_ICAP9,
    IC_SYNC_MODE_INPUT_TMR1,
    IC_SYNC_MODE_INPUT_TMR2,
    IC_SYNC_MODE_INPUT_TMR3,
    IC_SYNC_MODE_INPUT_TMR4,
    IC_SYNC_MODE_INPUT_TMR5,
    IC_SYNC_MODE_INPUT_PTG,
    IC_SYNC_MODE_INPUT_CMP1,
    IC_SYNC_MODE_INPUT_CMP2,
    IC_SYNC_MODE_INPUT_CMP3,
    IC_SYNC_MODE_INPUT_AD,
    IC_SYNC_MODE_INPUT_CTMU,
    IC_SYNC_MODE_INPUT_NONE
} IC_SYNC_MODE_INPUTS;
```

Description

IC Synchronization and Trigger input select

This enumeration lists all of the input sources for synchronous and asynchronous operation of IC module. The Input source can be selected by the [PLIB_IC_SyncModeInputSelect\(\)](#) function.

Members

Members	Description
IC_SYNC_MODE_INPUT_OC1	Synchronization and trigger input is Output Compare1
IC_SYNC_MODE_INPUT_OC2	Synchronization and trigger input is Output Compare2
IC_SYNC_MODE_INPUT_OC3	Synchronization and trigger input is Output Compare3
IC_SYNC_MODE_INPUT_OC4	Synchronization and trigger input is Output Compare4
IC_SYNC_MODE_INPUT_OC5	Synchronization and trigger input is Output Compare5
IC_SYNC_MODE_INPUT_OC6	Synchronization and trigger input is Output Compare6
IC_SYNC_MODE_INPUT_OC7	Synchronization and trigger input is Output Compare7
IC_SYNC_MODE_INPUT_OC8	Synchronization and trigger input is Output Compare8
IC_SYNC_MODE_INPUT_OC9	Synchronization and trigger input is Output Compare9

IC_SYNC_MODE_INPUT_ICAP1	Synchronization and trigger input is Input Capture1
IC_SYNC_MODE_INPUT_ICAP2	Synchronization and trigger input is Input Capture2
IC_SYNC_MODE_INPUT_ICAP3	Synchronization and trigger input is Input Capture3
IC_SYNC_MODE_INPUT_ICAP4	Synchronization and trigger input is Input Capture4
IC_SYNC_MODE_INPUT_ICAP5	Synchronization and trigger input is Input Capture5
IC_SYNC_MODE_INPUT_ICAP6	Synchronization and trigger input is Input Capture6
IC_SYNC_MODE_INPUT_ICAP7	Synchronization and trigger input is Input Capture7
IC_SYNC_MODE_INPUT_ICAP8	Synchronization and trigger input is Input Capture8
IC_SYNC_MODE_INPUT_ICAP9	Synchronization and trigger input is Input Capture9
IC_SYNC_MODE_INPUT_TMR1	Synchronization and trigger input is Timer1
IC_SYNC_MODE_INPUT_TMR2	Synchronization and trigger input is Timer2
IC_SYNC_MODE_INPUT_TMR3	Synchronization and trigger input is Timer3
IC_SYNC_MODE_INPUT_TMR4	Synchronization and trigger input is Timer4
IC_SYNC_MODE_INPUT_TMR5	Synchronization and trigger input is Timer5
IC_SYNC_MODE_INPUT_PTG	Synchronization and trigger input is Peripheral Trigger Generator Module
IC_SYNC_MODE_INPUT_CMP1	Trigger input is Comparator1. This input is not used for Synchronization
IC_SYNC_MODE_INPUT_CMP2	Trigger input is Comparator2. This input is not used for Synchronization
IC_SYNC_MODE_INPUT_CMP3	Trigger input is Comparator3. This input is not used for Synchronization
IC_SYNC_MODE_INPUT_AD	Trigger input is ADC. This input is not used for Synchronization
IC_SYNC_MODE_INPUT_CTMU	Trigger input is Charge Time Measurement Unit. This input is not used for Synchronization
IC_SYNC_MODE_INPUT_NONE	No synchronization and trigger input

5.6.10.5.6.8 IC_SYNC_MODES Enumeration

C

```
typedef enum {
    IC_MODE_ASYNC,
    IC_MODE_SYNC
} IC_SYNC_MODES;
```

Description

IC Synchronization Mode Select

This enumeration lists the synchronization modes for the IC module. The IC module timer can function in synchronism with the input source or triggered asynchronously by an input trigger.

Members

Members	Description
IC_MODE_ASYNC	Input source acts as an asynchronous trigger to start Input Capture timer. The Input source can be selected by PLIB_IC_SyncModeInputSelect() routine
IC_MODE_SYNC	Input source synchronizes Input Capture timer to other modules. The Input source can be selected by PLIB_IC_SyncModeInputSelect() routine

5.6.10.5.6.9 IC_TIMERS Enumeration

C

```
typedef enum {
    IC_TIMER_TMR3,
    IC_TIMER_TMR2
} IC_TIMERS;
```


Description

IC 16-bit Timer Select

This enumeration lists the different 16-bit timers for the IC timebase when the IC module is configured with a 16-bit timer resource.

Members

Members	Description
IC_TIMER_TMR3	Timer3 select
IC_TIMER_TMR2	Timer2 select

5.6.10.5.7 Feature Existence Functions

5.6.10.5.7.1 PLIB_IC_ExistsAlternateClock Function

C

```
bool PLIB_IC_ExistsAlternateClock(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the AlternateClock feature is available on the IC module. When this function returns true, these functions are supported on the device:

- [PLIB_IC_AlternateClockEnable](#)
- [PLIB_IC_AlternateClockDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlternateClock feature is supported on the device
- false - The AlternateClock feature is not supported on the device

Remarks

None.

5.6.10.5.7.2 PLIB_IC_ExistsBufferIsEmptyStatus Function

C

```
bool PLIB_IC_ExistsBufferIsEmptyStatus(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the BufferIsEmptyStatus feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_BufferIsEmpty](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferIsEmptyStatus feature is supported on the device
- false - The BufferIsEmptyStatus feature is not supported on the device

Remarks

None.

5.6.10.5.7.3 PLIB_IC_ExistsBufferOverflowStatus Function**C**

```
bool PLIB_IC_ExistsBufferOverflowStatus(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the BufferOverflowStatus feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_BufferOverflowHasOccurred](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferOverflowStatus feature is supported on the device
- false - The BufferOverflowStatus feature is not supported on the device

Remarks

None.

5.6.10.5.7.4 PLIB_IC_ExistsBufferSize Function**C**

```
bool PLIB_IC_ExistsBufferSize(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the BufferSize feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_BufferSizeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferSize feature is supported on the device
- false - The BufferSize feature is not supported on the device

Remarks

None.

5.6.10.5.7.5 PLIB_IC_ExistsBufferValue Function**C**

```
bool PLIB_IC_ExistsBufferValue(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the BufferValue feature is available on the IC module. When this function returns true, these functions are supported on the device:

- [PLIB_IC_Buffer32BitGet](#)
- [PLIB_IC_Buffer16BitGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferValue feature is supported on the device
- false - The BufferValue feature is not supported on the device

Remarks

None.

5.6.10.5.7.6 PLIB_IC_ExistsCaptureMode Function**C**

```
bool PLIB_IC_ExistsCaptureMode(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the CaptureMode feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_ModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CaptureMode feature is supported on the device
- false - The CaptureMode feature is not supported on the device

Remarks

None.

5.6.10.5.7.7 PLIB_IC_ExistsClockSource Function**C**

```
bool PLIB_IC_ExistsClockSource(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the ClockSource feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_ClockSourceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ClockSource feature is supported on the device
- false - If the ClockSource feature is not supported on the device

Remarks

None.

5.6.10.5.7.8 PLIB_IC_ExistsEdgeCapture Function**C**

```
bool PLIB_IC_ExistsEdgeCapture(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the EdgeCapture feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_FirstCaptureEdgeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EdgeCapture feature is supported on the device
- false - The EdgeCapture feature is not supported on the device

Remarks

None.

5.6.10.5.7.9 PLIB_IC_ExistsEnable Function**C**

```
bool PLIB_IC_ExistsEnable(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the IC module. When this function returns true, these functions are supported on the device:

- [PLIB_IC_Enable](#)
- [PLIB_IC_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Remarks

None.

5.6.10.5.7.10 PLIB_IC_ExistsEventsPerInterruptSelect Function**C**

```
bool PLIB_IC_ExistsEventsPerInterruptSelect(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the EventsPerInterruptSelect feature is available on the IC module. When this function returns true, this function is are supported on the device:

- [PLIB_IC_EventsPerInterruptSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EventsPerInterruptSelect feature is supported on the device
- false - The EventsPerInterruptSelect feature is not supported on the device

Remarks

None.

5.6.10.5.7.11 PLIB_IC_ExistsStopInIdle Function

C

```
bool PLIB_IC_ExistsStopInIdle(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the IC module. When this function returns true, these functions are supported on the device:

- [PLIB_IC_StopInIdleEnable](#)
- [PLIB_IC_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Remarks

None.

5.6.10.5.7.12 PLIB_IC_ExistsSyncMode Function

C

```
bool PLIB_IC_ExistsSyncMode(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the SyncMode feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_SyncModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SyncMode feature is supported on the device
- false - If the SyncMode feature is not supported on the device

Remarks

None.

5.6.10.5.7.13 PLIB_IC_ExistsSyncModeInput Function

C

```
bool PLIB_IC_ExistsSyncModeInput(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the SyncModeInput feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_SyncModeInputSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SyncModeInput feature is supported on the device
- false - If the SyncModeInput feature is not supported on the device

Remarks

None.

5.6.10.5.7.14 PLIB_IC_ExistsTimerSelect Function

C

```
bool PLIB_IC_ExistsTimerSelect(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the TimerSelect feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_TimerSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TimerSelect feature is supported on the device
- false - The TimerSelect feature is not supported on the device

Remarks

None.

5.6.10.5.7.15 PLIB_IC_ExistsTimerTriggered Function

C

```
bool PLIB_IC_ExistsTimerTriggered(  
    IC_MODULE_ID index  
);
```

Description

This function identifies whether the TimerTriggered feature is available on the IC module. When this function returns true, this function is supported on the device:

- [PLIB_IC_TimerIsTriggered](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TimerTriggered feature is supported on the device
- false - If the TimerTriggered feature is not supported on the device

Remarks

None.

5.6.10.6 Files

Files

Name	Description
plib_ic.h	This file contains the interface definition for the Input Capture (IC) peripheral library.

Description

5.6.10.6.1 plib_ic.h

IC Module Peripheral Library Interface Header

This library provides a low-level abstraction of the Input Capture (IC) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

Functions

	Name	Description
≡	<u>PLIB_IC_AlternateClockDisable</u>	Selects Timer 2/3 instead of the alternate clock source.
≡	<u>PLIB_IC_AlternateClockEnable</u>	Selects the alternate clock source.
≡	<u>PLIB_IC_Buffer16BitGet</u>	Obtains the 16-bit input capture buffer value.
≡	<u>PLIB_IC_Buffer32BitGet</u>	Obtains the 32-bit input capture buffer value.
≡	<u>PLIB_IC_BufferIsEmpty</u>	Checks whether the input capture buffer is empty.
≡	<u>PLIB_IC_BufferOverflowHasOccurred</u>	Checks whether an input capture buffer overflow has occurred.
≡	<u>PLIB_IC_BufferSizeSelect</u>	Sets the input capture buffer size.
≡	<u>PLIB_IC_ClockSourceSelect</u>	Selects the clock source for the IC module.
≡	<u>PLIB_IC_Disable</u>	Disables the IC module.
≡	<u>PLIB_IC_Enable</u>	Enables the IC module.
≡	<u>PLIB_IC_EventsPerInterruptSelect</u>	Selects the number of capture events before an interrupt is issued.
≡	<u>PLIB_IC_ExistsAlternateClock</u>	Identifies whether the AlternateClock feature exists on the IC module
≡	<u>PLIB_IC_ExistsBufferIsEmptyStatus</u>	Identifies whether the BufferIsEmptyStatus feature exists on the IC module
≡	<u>PLIB_IC_ExistsBufferOverflowStatus</u>	Identifies whether the BufferOverflowStatus feature exists on the IC module.
≡	<u>PLIB_IC_ExistsBufferSize</u>	Identifies whether the BufferSize feature exists on the IC module.
≡	<u>PLIB_IC_ExistsBufferValue</u>	Identifies whether the BufferValue feature exists on the IC module.
≡	<u>PLIB_IC_ExistsCaptureMode</u>	Identifies whether the CaptureMode feature exists on the IC module.
≡	<u>PLIB_IC_ExistsClockSource</u>	Identifies whether the ClockSource feature exists on the IC module.
≡	<u>PLIB_IC_ExistsEdgeCapture</u>	Identifies whether the EdgeCapture feature exists on the IC module.
≡	<u>PLIB_IC_ExistsEnable</u>	Identifies whether the EnableControl feature exists on the IC module
≡	<u>PLIB_IC_ExistsEventsPerInterruptSelect</u>	Identifies whether the EventsPerInterruptSelect feature exists on the IC module.
≡	<u>PLIB_IC_ExistsStopInIdle</u>	Identifies whether the StopInIdle feature exists on the IC module.
≡	<u>PLIB_IC_ExistsSyncMode</u>	Identifies whether the SyncMode feature exists on the IC module.
≡	<u>PLIB_IC_ExistsSyncModeInput</u>	Identifies whether the SyncModeInput feature exists on the IC module.
≡	<u>PLIB_IC_ExistsTimerSelect</u>	Identifies whether the TimerSelect feature exists on the IC module.
≡	<u>PLIB_IC_ExistsTimerTriggered</u>	Identifies whether the TimerTriggered feature exists on the IC module.
≡	<u>PLIB_IC_FirstCaptureEdgeSelect</u>	Captures the timer count value at the first selected edge of input signal.
≡	<u>PLIB_IC_ModeSelect</u>	Selects the input capture mode for IC module.
≡	<u>PLIB_IC_StopInIdleDisable</u>	Continues module operation when the device enters Idle mode
≡	<u>PLIB_IC_StopInIdleEnable</u>	Discontinues IC module operation when the device enters Idle mode.
≡	<u>PLIB_IC_SyncModeInputSelect</u>	Selects the input source for synchronous or asynchronous (triggered) operation of the IC module.
≡	<u>PLIB_IC_SyncModeSelect</u>	Selects the synchronization mode for Input Capture module.
≡	<u>PLIB_IC_TimerIsTriggered</u>	Checks whether or not the input capture timer has been triggered.

	PLIB_IC_TimerSelect	Selects the 16-bit timer source for the IC module.

File Name

plib_ic.h

Company

Microchip Technology Inc.

5.6.11 Interrupt Peripheral Library

5.6.11.1 Introduction

Interrupt Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Interrupt Controller module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description**Interrupt Controller Overview**

The Interrupt Controller is a key component of a microcontroller. The Interrupt Controller reduces the numerous peripheral interrupt request signals to a single interrupt request signal to the microcontroller core.

The key features present on an Interrupt Controller are:

- **Interrupt source configuration:** Provides the ability to enable, disable, and prioritize a source that may interrupt the core
- **Interrupt status flags:** Identify the source of an interrupt request or exception, usually generated by a peripheral

The interrupt will be sent to the core, only if the source is enabled to generate an interrupt and interrupts have been enabled (if supported). Sources will be prioritized so that the highest priority interrupt will be sent to the core if multiple sources cause interrupts at the same time.

Many interrupt controllers also include, other optional features:

- **Global enable configuration:** Controls the generation of any interrupt to the core from the Interrupt Controller
- **Choice of edge transition:** Controls the edge transition (high-to-low or low-to-high) on which the interrupts are generated
- **Priority configuration:** Associates a programmable priority with the interrupt source

Interrupts can be handled inside the core in the following ways:

- **Vectored:** Each interrupt has an associated interrupt handler routine
- **Non-vectored:** All interrupts have a single (shared) interrupt handler routine. The software determines the interrupt source based on the interrupt status-flags register.
- **Combination of vectored and non-vectored:** Interrupts combined in groups. Each group has a single common interrupt vector or handler, but there are multiple groups (i.e., multiple vectors). Within a specific vector group, software determines which one in the group sources caused the interrupt.

A microcontroller can support one, or more than one of the these modes.

Note: Please refer to the the specific device data sheet to determine availability of these features for your device.

5.6.11.2 Release Notes

MPLAB Harmony Version: v0.70b **Interrupt Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.11.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.11.4 Using the Library

This topic describes the basic architecture of the Interrupt Peripheral Library and provides information and examples on how to use it.

Interface Header File: plib_int.h

The interface to the Interrupt Peripheral Library is defined in the "plib_int.h" header file. Any C language source (.c) file that uses the Interrupt Peripheral Library should include "plib_int.h".

Library File:

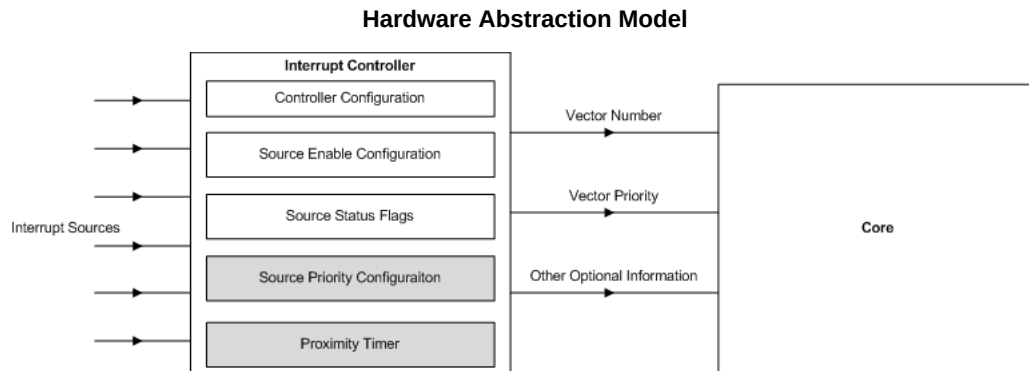
The Interrupt Peripheral Library (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the peripheral interacts with the framework.

5.6.11.4.1 Hardware Abstraction Model

This library provides a low-level abstraction of the Interrupt Controller module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description



The **Interrupt Controller** receives requests from multiple **Interrupt Sources**. The **Interrupt Controller Configuration** controls the overall operation of the interrupt controller. Not all microcontrollers support all the features listed below:

- Interrupt nesting configuration
- Edge selection of the external interrupts
- Vector mode selection
- Priority level configuration
- Current CPU interrupt and priority status information

Source Enable Configuration enables or disables a particular source. On some microcontrollers other CPU exceptions or traps can also be controlled. **Source Status Flags** give the status of the interrupt request from a specific interrupt source.

Other optional features include the **Vector Priority Configuration**, which is available on some micro controllers and the **Proximity Timer**. Priority configuration allows a priority level to be associated with each interrupt vector. The proximity timer creates a temporal window in which a group of interrupts of the same, or lower, priority will be held off. This provides an opportunity to queue these interrupt requests and process them using tail-chaining of multiple IRQs and allows higher priority interrupts to pass through during the hold-off window.

Traps and Exceptions

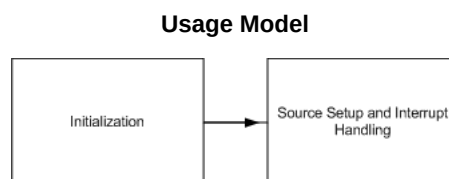
Some of the things that can interrupt the normal flow of instruction execution of the CPU core are not actual interrupt sources. These are things such as a division-by-zero error or attempting to address unimplemented memory. These errors are either internal to the CPU core itself or are so catastrophic that they are non-maskable and are trapped separately. This library refers to them in general as "traps" and there are interface elements provided to allow software to handle them.

5.6.11.4.2 Library Usage Model

This section provides library usage model information.

Description

The following diagram shows the major components of the usage model.



Each of the blocks in the diagram correspond to the library interface section.

Library Interface Section	Description
General Configuration Functions	Provides setup, configuration, and status interface routines for the overall operation of the Interrupt Controller module.

5.6.11.4.3 How the Library Works

This section provides information on initialization and interrupt source setup and handling.

Description

Initialization

The steps that are required to initialize the Interrupt Controller module vary for different microcontrollers.

Note: Refer to the specific device data sheet for the correct initialization sequence for your device.

Interrupt Source Setup

An interrupt Source needs to be enabled in order for it to be able to generate a interrupt to the core. Use the function [PLIB_INT_SourceEnable](#) to enable the interrupt source. [INT_SOURCE](#) is a list of the possible interrupt sources

On some microcontrollers, each interrupt vector can be associated with a priority. Use the function [PLIB_INT_VectorPrioritySet](#) to set the interrupt vector priority. [INT_VECTOR](#) is a list of the possible interrupt vectors and [INT_PRIORITY_LEVEL](#) is the list of possible priority levels.

On some microcontrollers, each interrupt vector can be associated with a sub-priority. Use the function [PLIB_INT_VectorSubPrioritySet](#) to set the interrupt source sub-priority. [INT_SUBPRIORITY_LEVEL](#) is a list of possible sub-priority levels.

On most microcontrollers, it will be necessary to enable interrupt generation to the core by calling the [PLIB_INT_Enable](#) function.

Interrupt Handling

The status of the interrupt indicates whether or not the interrupt request was generated by the source. The status of the trap source can be obtained using the function [PLIB_INT_TrapSourceFlagGet](#) and the status of the interrupt source can be obtained using the function [PLIB_INT_SourceFlagGet](#). The interrupt status flags can be used to indicate the interrupt status even if the interrupt is not enabled for the source. When the interrupt is not enabled for the source, no interrupt is generated to the core even though the interrupt controller still recognizes that the source requested an interrupt. [INT_TRAP_SOURCE](#) is a list of possible trap events, and [INT_SOURCE](#) is a list of possible interrupt sources.

Before the Interrupt Service Routine (ISR) ends, the status of the interrupt needs to be cleared to avoid entering the ISR repeatedly for the same interrupt event. A trap source status can be cleared using the function [PLIB_INT_TrapSourceFlagClear](#), and an interrupt source status can be cleared using the function [PLIB_INT_SourceFlagClear](#). [INT_TRAP_SOURCE](#) is a list of possible trap events, and [INT_SOURCE](#) is a list of possible interrupt sources.

5.6.11.5 Configuring the Library

The library is configured for the supported Interrupt Peripheral Library.

5.6.11.6 Library Interface

Data Types and Constants

	Name	Description
	<u>INT_EXTERNAL_SOURCES</u>	Lists the possible external sources that can cause an interrupt to occur.
	<u>INT_PRIORITY_LEVEL</u>	Lists the possible interrupt priority levels
	<u>INT_SOURCE</u>	Lists the possible sources that can cause an interrupt to occur
	<u>INT_SUBPRIORITY_LEVEL</u>	Lists the possible interrupt sub priority levels
	<u>INT_TRAP_SOURCE</u>	Lists the names of the trap or exception sources
	<u>INT_VECTOR</u>	Lists the possible interrupt vectors.

Feature Existence Functions

	Name	Description
	<u>PLIB_INT_ExistsAlternateVectorTable</u>	Identifies whether the AlternateVectorTable feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsCPUCurrentPriorityLevel</u>	Identifies whether the CPUCurrentPriorityLevel feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsEnableControl</u>	Identifies whether the EnableControl feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsExternalINTEdgeSelect</u>	Identifies whether the ExternalINTEdgeSelect feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsHighPriority</u>	Identifies whether the HighPriority feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsINTCPUPriority</u>	Identifies whether the INTCPUPriority feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsINTCPUVector</u>	Identifies whether the INTCPUVector feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsINTNesting</u>	Identifies whether the INTNesting feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsLowPriority</u>	Identifies whether the LowPriority feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsPeripheralControl</u>	Identifies whether the PeripheralControl feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsPriority</u>	Identifies whether the Priority feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsProximityTimerControl</u>	Identifies whether the ProximityTimerControl feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsProximityTimerEnable</u>	Identifies whether the ProximityTimerEnable feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsSingleVectorShadowSet</u>	Identifies whether the SingleVectorShadowSet feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsSourceControl</u>	Identifies whether the SourceControl feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsSourceFlag</u>	Identifies whether the SourceFlag feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsTrapSource</u>	Identifies whether the TrapSource feature exists on the Interrupt Controller module.
	<u>PLIB_INT_ExistsVectorPriority</u>	Identifies whether the VectorPriority feature exists on the Interrupt Controller module.

	PLIB_INT_ExistsVectorSelect	Identifies whether the VectorSelect feature exists on the Interrupt Controller module.

General Configuration Functions

	Name	Description
	PLIB_INT_AlternateVectorTableSelect	Enables the use of the alternate vector table.
	PLIB_INT_Disable	Disables the generation of interrupts to the CPU.
	PLIB_INT_DISIStatusGet	Returns the status of the DISI instruction status.
	PLIB_INT_Enable	Enables the generation of interrupts to the CPU.
	PLIB_INT_ExternalFallingEdgeSelect	Selects the falling edge as the edge polarity of the external interrupt.
	PLIB_INT_ExternalRisingEdgeSelect	Selects the rising edge as the edge polarity of the external interrupt.
	PLIB_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.
	PLIB_INT_MultiVectorSelect	Configures the Interrupt Controller for Multiple Vector mode.
	PLIB_INT_NestingDisable	Disables interrupt nesting.
	PLIB_INT_NestingEnable	Enables interrupt nesting.
	PLIB_INT_PeripheralsDisable	Disables the generation of interrupts to the CPU by the peripherals.
	PLIB_INT_PeripheralsEnable	Enables the generation of interrupts to the CPU by the peripherals.
	PLIB_INT_PriorityDisable	Disables interrupt priority levels.
	PLIB_INT_PriorityEnable	Enables interrupt priority levels.
	PLIB_INT_PriorityGet	Returns the priority level of the latest interrupt presented to the CPU.
	PLIB_INT_PriorityHighDisable	Disables the generation of high-priority interrupts to the CPU.
	PLIB_INT_PriorityHighEnable	Enables the generation of high-priority interrupts to the CPU.
	PLIB_INT_PriorityLowDisable	Disables the generation of low-priority interrupts to the CPU.
	PLIB_INT_PriorityLowEnable	Enables generation of low-priority interrupts to the CPU.
	PLIB_INT_ProximityTimerDisable	Disables the interrupt temporal-proximity timer.
	PLIB_INT_ProximityTimerEnable	Enables the interrupt temporal-proximity timer and selects the priority levels that start the timer.
	PLIB_INT_SingleVectorSelect	Configures the Interrupt Controller for Single Vector mode.
	PLIB_INT_SingleVectorShadowSetDisable	Disables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SingleVectorShadowSetEnable	Enables the Shadow Register Set in Single Vector mode.
	PLIB_INT_StandardVectorTableSelect	Enables the use of the standard vector table.

Interrupt Source Control Functions

	Name	Description
	PLIB_INT_SourceDisable	Disables the interrupt source
	PLIB_INT_SourceEnable	Enables the interrupt source. This section provides information on initialization and interrupt source setup and handling.
	PLIB_INT_SourceFlagClear	Clears the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagGet	Returns the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagSet	Sets the status of the interrupt flag for the selected source.
	PLIB_INT_SourceIsEnabled	Gets the enable state of the interrupt source.
	PLIB_INT_VectorPriorityGet	Gets the priority of the interrupt vector.
	PLIB_INT_VectorPrioritySet	Sets the priority of the interrupt vector.
	PLIB_INT_VectorSubPriorityGet	Gets the sub-priority of the interrupt vector.
	PLIB_INT_VectorSubPrioritySet	Sets the sub-priority of the interrupt vector.

Other Status and Control Functions

	Name	Description
≡	PLIB_INT_CPUCurrentPriorityLevelGet	Gets the interrupt priority level at which the CPU is currently operating.
≡	PLIB_INT_ProximityTimerGet	Returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.
≡	PLIB_INT_ProximityTimerSet	Sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.
≡	PLIB_INT_TrapSourceFlagClear	Clears the flag for the selected trap.
≡	PLIB_INT_TrapSourceFlagGet	Returns the status of the flag for the selected trap.
≡	PLIB_INT_VectorGet	Returns the interrupt vector that is presented to the CPU.

Description

This section provides descriptions and examples for the functions provided in the Interrupt Peripheral Library.

5.6.11.6.1 General Configuration Functions

5.6.11.6.1.1 PLIB_INT_AlternateVectorTableSelect Function

C

```
void PLIB_INT_AlternateVectorTableSelect(
    INT_MODULE_ID index
);
```

Description

This function enables the processor to use the alternate vector table instead of the default vector table. If the alternate vector table is enabled using this function, all interrupt and exception processing uses the alternate vectors instead of the default vectors. The alternate vectors are organized in the same manner as the default vectors.

Alternate vectors support emulation and debugging by providing a means to switch between an application and support environment, without requiring the interrupt vectors to be reprogrammed. This feature can also be used to enable switching between different software algorithms during run time for evaluation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

None.

Remarks

This function implements an operation of the AlternateVectorTable feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsAlternateVectorTable](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_AlternateVectorTableSelect( INT_ID_0 );
```


5.6.11.6.1.2 PLIB_INT_Disable Function

C

```
void PLIB_INT_Disable(  
    INT_MODULE_ID index  
);
```

Description

This function disables the generation of interrupts to the CPU.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only one interrupt module).

Returns

None.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsEnableControl](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_Disable( INT_ID_0 );
```

5.6.11.6.1.3 PLIB_INT_DISIStatusGet Function

C

```
bool PLIB_INT_DISIStatusGet(  
    INT_MODULE_ID index  
);
```

Description

This function returns the status of the DISI instruction status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only one interrupt module).

Returns

- true - If DISI is active
- false - If DISI is not active

Remarks

This function implements an operation of the DISI feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsDISI](#) API to write flexible code to automatically

determine if this feature is available.

Example

```
PLIB_INT_DISISStatusGet( INT_ID_0 );
```

5.6.11.6.1.4 PLIB_INT_Enable Function**C**

```
void PLIB_INT_Enable(  
    INT_MODULE_ID index  
);
```

Description

This function enables the generation of interrupts to the CPU.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

None.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsEnableControl](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_Enable( INT_ID_0 );
```

5.6.11.6.1.5 PLIB_INT_ExternalFallingEdgeSelect Function**C**

```
void PLIB_INT_ExternalFallingEdgeSelect(  
    INT_MODULE_ID index,  
    INT_EXTERNAL_SOURCES source  
);
```

Description

This function selects the falling edge as the edge polarity of the external interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
source	One or more of the values from INT_EXTERNAL_SOURCES . Values can be combined using a bitwise "OR" operation.

Returns

None.

Remarks

This function implements an operation of the ExternalINTEdgeSelect feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsExternalINTEdgeSelect](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_ExternalFallingEdgeSelect( INT_ID_0,
                                   (INT_EXTERNAL_INT_SOURCE0 |
                                   INT_EXTERNAL_INT_SOURCE1) );
```

5.6.11.6.1.6 PLIB_INT_ExternalRisingEdgeSelect Function

C

```
void PLIB_INT_ExternalRisingEdgeSelect(
    INT_MODULE_ID index,
    INT_EXTERNAL_SOURCES source
);
```

Description

This function selects the rising edge as the edge polarity of the external interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
source	One or more of the values from INT_EXTERNAL_SOURCES . Values can be combined using a bitwise "OR" operation.

Returns

None.

Remarks

This function implements an operation of the ExternalINTEdgeSelect feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsExternalINTEdgeSelect](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_ExternalRisingEdgeSelect( INT_ID_0,
                                   ( INT_EXTERNAL_INT_SOURCE0 |
                                   INT_EXTERNAL_INT_SOURCE1 ) );
```

5.6.11.6.1.7 PLIB_INT_IsEnabled Function

C

```
bool PLIB_INT_IsEnabled(
    INT_MODULE_ID index
);
```

Description

This function identifies if interrupts are enabled or disabled at the top level.

Preconditions

None.

Returns

- true - If the interrupts are currently enabled
- false - If the interrupts are currently disabled

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsEnableControl](#) API to write flexible code to automatically determine if this feature is available.

Example

```
status = PLIB_INT_IsEnabled( INT_ID_0 );
```

5.6.11.6.1.8 PLIB_INT_MultiVectorSelect Function**C**

```
void PLIB_INT_MultiVectorSelect(  
    INT_MODULE_ID index  
);
```

Description

This function configures the Interrupt Controller for Multiple Vector mode. Interrupt requests will serviced at the calculated vector addresses. The CPU vectors to the unique address for each vector number.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

None.

Remarks

While the user can, during run-time, reconfigure the interrupt controller from Single Vector mode to Multiple Vector mode, such action is strongly discouraged, as it may result in undefined behavior.

This function implements an operation of the VectorSelect feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsVectorSelect](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_MultiVectorSelect( INT_ID_0 );
```

5.6.11.6.1.9 PLIB_INT_NestingDisable Function

C

```
void PLIB_INT_NestingDisable(  
    INT_MODULE_ID index  
);
```

Description

This function disables interrupt nesting on processors that support it.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

None.

Remarks

This function implements an operation of the INTNesting feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsINTNesting](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_NestingDisable( INT_ID_0 );
```

5.6.11.6.1.10 PLIB_INT_NestingEnable Function

C

```
void PLIB_INT_NestingEnable(  
    INT_MODULE_ID index  
);
```

Description

This function enables interrupt nesting on processors that support it.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

None.

Remarks

This function implements an operation of the INTNesting feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsINTNesting](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_NestingEnable( INT_ID_0 );
```

5.6.11.6.1.11 PLIB_INT_PeripheralsDisable Function**C**

```
void PLIB_INT_PeripheralsDisable(  
    INT_MODULE_ID index  
);
```

Description

This function disables the generation of interrupts to the CPU by the peripherals on devices that support this feature.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the PeripheralControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsPeripheralControl](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_PeripheralsDisable( INT_ID_0 );
```

5.6.11.6.1.12 PLIB_INT_PeripheralsEnable Function**C**

```
void PLIB_INT_PeripheralsEnable(  
    INT_MODULE_ID index  
);
```

Description

This function enables the generation of interrupts to the CPU by the peripherals on devices with the ability to disable them.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the PeripheralControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsPeripheralControl](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_PeripheralsEnable( INT_ID_0 );
```

5.6.11.6.1.13 PLIB_INT_PriorityDisable Function**C**

```
void PLIB_INT_PriorityDisable(  

```

```
    INT_MODULE_ID index
);
```

Description

This function disables interrupt priority levels on devices that support programmable interrupt priorities.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only one interrupt module).

Returns

None.

Remarks

This function implements an operation of the Priority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_PriorityDisable( INT_ID_0 );
```

5.6.11.6.1.14 PLIB_INT_PriorityEnable Function**C**

```
void PLIB_INT_PriorityEnable(
    INT_MODULE_ID index
);
```

Description

This function enables interrupt priority levels on devices that support programmable interrupt priorities.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only one interrupt module).

Returns

None.

Remarks

This function implements an operation of the Priority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_PriorityEnable( INT_ID_0 );
```

5.6.11.6.1.15 PLIB_INT_PriorityGet Function

C

```
INT_PRIORITY_LEVEL PLIB_INT_PriorityGet(  
    INT_MODULE_ID index  
);
```

Description

This function returns the priority level of the latest interrupt presented to the CPU.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

One of the possible values from [INT_PRIORITY_LEVEL](#).

Remarks

This value should only be used when the interrupt controller is configured for single vector mode using the function [PLIB_INT_SingleVectorSelect](#).

This function implements an operation of the INTCPUPriority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsINTCPUPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
INT_PRIORITY_LEVEL level = PLIB_INT_RequestedPriorityLevelGet( INT_ID_0 );
```

5.6.11.6.1.16 PLIB_INT_PriorityHighDisable Function

C

```
void PLIB_INT_PriorityHighDisable(  
    INT_MODULE_ID index  
);
```

Description

This function disables the generation of high-priority interrupts to the CPU on devices that support high-priority interrupts.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only one interrupt module).

Returns

None.

Remarks

This function implements an operation of the HighPriority feature. This feature may not be available on all parts. Please refer to

the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsHighPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_PriorityHighDisable( INT_ID_0 );
```

5.6.11.6.1.17 PLIB_INT_PriorityHighEnable Function**C**

```
void PLIB_INT_PriorityHighEnable(  
    INT_MODULE_ID index  
);
```

Description

This function enables the generation of high-priority interrupts to the CPU on devices that support this feature.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the HighPriority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsHighPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_PriorityHighEnable( INT_ID_0 );
```

5.6.11.6.1.18 PLIB_INT_PriorityLowDisable Function**C**

```
void PLIB_INT_PriorityLowDisable(  
    INT_MODULE_ID index  
);
```

Description

This function disables the generation of low-priority interrupts to the CPU on devices that support this feature.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only one interrupt module).

Returns

None.

Remarks

This function implements an operation of the LowPriority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsLowPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_PriorityLowDisable( INT_ID_0 );
```

5.6.11.6.1.19 PLIB_INT_PriorityLowEnable Function**C**

```
void PLIB_INT_PriorityLowEnable(  
    INT_MODULE_ID index  
);
```

Description

This function enables generation of low-priority interrupts to the CPU on devices that support this feature.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only one interrupt module).

Returns

None.

Remarks

This function implements an operation of the LowPriority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsLowPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_PriorityLowEnable( INT_ID_0 );
```

5.6.11.6.1.20 PLIB_INT_ProximityTimerDisable Function**C**

```
void PLIB_INT_ProximityTimerDisable(  
    INT_MODULE_ID index  
);
```

Description

This function disables the interrupt temporal-proximity timer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

None.

Remarks

This function implements an operation of the ProximityTimerEnable feature. This feature may not be available on all parts.

Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsProximityTimerEnable](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_ProximityTimerDisable(INT_PRIORITY_LEVEL3);
```

5.6.11.6.1.21 PLIB_INT_ProximityTimerEnable Function**C**

```
void PLIB_INT_ProximityTimerEnable(
    INT_MODULE_ID index,
    INT_PRIORITY_LEVEL priority
);
```

Description

This function enables the interrupt temporal-proximity timer and selects the priority levels that start the timer. One of the possible values from `PLIB_INT_PRIORITY_LEVEL` can be selected. Interrupts of that priority and lower start the temporal proximity timer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be <code>INT_ID_0</code> for all the devices which has only interrupt module).
priority	One of possible values from INT_PRIORITY_LEVEL .

Returns

None.

Remarks

This function implements an operation of the `ProximityTimerEnable` feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsProximityTimerEnable](#) API to write flexible code to automatically determine if this feature is available.

Choosing `INT_PRIORITY_0` disables the proximity timer (exactly the same as if [PLIB_INT_ProximityTimerDisable](#) had been called.

Example

```
//Interrupts of group priority 3 or lower start the temporal proximity timer
PLIB_INT_ProximityTimerEnable( INT_ID_0, INT_PRIORITY_LEVEL3 );
```

5.6.11.6.1.22 PLIB_INT_SingleVectorSelect Function**C**

```
void PLIB_INT_SingleVectorSelect(
    INT_MODULE_ID index
);
```

Description

The function configures the Interrupt Controller for Single Vector mode. All interrupt requests will serviced at one vector address. The CPU vectors to the same address for all interrupt sources.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

None.

Remarks

While the user can, during run-time, reconfigure the Interrupt Controller from Single Vector mode to Multiple Vector mode, such action is strongly discouraged, as it may result in undefined behavior.

This function implements an operation of the VectorSelect feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsVectorSelect](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_SingleVectorSelect( INT_ID_0 );
```

5.6.11.6.1.23 PLIB_INT_SingleVectorShadowSetDisable Function**C**

```
void PLIB_INT_SingleVectorShadowSetDisable(  
    INT_MODULE_ID index  
);
```

Description

This function disables usage of the "shadow" set of processor registers when operating in Single Vector mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

None.

Remarks

This function implements an operation of the SingleVectorShadowSet feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsSingleVectorShadowSet](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_SingleVectorShadowSetEnable( INT_ID_0 );
```

5.6.11.6.1.24 PLIB_INT_SingleVectorShadowSetEnable Function**C**

```
void PLIB_INT_SingleVectorShadowSetEnable(  
    INT_MODULE_ID index  
);
```

Description

This function enables usage of the "shadow" set of processor registers when operating in Single Vector mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

None.

Remarks

This function implements an operation of the SingleVectorShadowSet feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsSingleVectorShadowSet](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_SingleVectorShadowSetEnable( INT_ID_0 );
```

5.6.11.6.1.25 PLIB_INT_StandardVectorTableSelect Function

C

```
void PLIB_INT_StandardVectorTableSelect(  
    INT_MODULE_ID index  
);
```

Description

This function enables use of the standard vector table. This is the default. It is only necessary to use this function if the alternate vector table was enabled using the [PLIB_INT_AlternateVectorTableSelect](#) function.

Preconditions

None.

Returns

None.

Remarks

This function implements an operation of the AlternateVectorTable feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsAlternateVectorTable](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_StandardVectorTableSelect( INT_ID_0 );
```

5.6.11.6.2 Interrupt Source Control Functions

5.6.11.6.2.1 PLIB_INT_SourceDisable Function

C

```
void PLIB_INT_SourceDisable(  

```

```
    INT_MODULE_ID index,
    INT_SOURCE source
);
```

Description

This function disables the given interrupt source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
source	One of the possible values from INT_SOURCE .

Returns

None.

Remarks

This function implements an operation of the SourceControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsSourceControl](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_TIMER_1);
```

5.6.11.6.2.2 PLIB_INT_SourceEnable Function

C

```
void PLIB_INT_SourceEnable(
    INT_MODULE_ID index,
    INT_SOURCE source
);
```

Description

This function enables the interrupt source. The interrupt flag is set when the interrupt request is sampled. The pending interrupt request will not cause further processing if the interrupt is not enabled using this function or if interrupts are not enabled (on processors that support the [PLIB_INT_Enable](#) feature).

Initialization

The steps that are required to initialize the Interrupt Controller module vary for different microcontrollers.

Note: Refer to the specific device data sheet for the correct initialization sequence for your device.

Interrupt Source Setup <<<<<<< .mine

1. An interrupt Source needs to be enabled in order for it to be able to generate a interrupt to the core. Use the functions PLIB_INT_SourceEnable to enable the interrupt source. [INT_SOURCE](#) is a list of possible interrupt sources
2. On some micro controllers, each interrupt can be associated with a priority. Use the function PLIB_INT_SourcePrioritySet to set the interrupt source priority. INT_SOURCES is list of possible interrupt sources and INT_PRIORITY_LEVELS is the list of possible priority levels.
3. On some micro controllers, each interrupt can be associated with a sub priority. Use the function PLIB_INT_SourceSubPrioritySet to set the interrupt source sub priority. INT_SUBPRIORITY_LEVELS is a list of possible sub priority levels.
4. On most microcontrollers, it will be necessary to enable interrupt generation to the core by calling the [PLIB_INT_Enable](#)

routine.

=====

An interrupt Source needs to be enabled in order for it to be able to generate a interrupt to the core. Use the function `PLIB_INT_SourceEnable` to enable the interrupt source. [INT_SOURCE](#) is a list of the possible interrupt sources

On some microcontrollers, each interrupt can be associated with a priority. Use the function `PLIB_INT_VectorPrioritySet` to set the interrupt source priority. [INT_SOURCE](#) is a list of the possible interrupt sources and [INT_PRIORITY_LEVEL](#) is the list of possible priority levels.

On some microcontrollers, each interrupt can be associated with a sub-priority. Use the function `PLIB_INT_VectorSubPrioritySet` to set the interrupt source sub-priority. [INT_SUBPRIORITY_LEVEL](#) is a list of possible sub-priority levels.

On most microcontrollers, it will be necessary to enable interrupt generation to the core by calling the `PLIB_INT_Enable` function.

>>>>>> .r21563 Interrupt Handling

The status of the interrupt indicates whether or not the interrupt request was generated by the source. The status of the trap source can be obtained using the function `PLIB_INT_TrapSourceFlagGet` and the status of the interrupt source can be obtained using the function `PLIB_INT_SourceFlagGet`. The interrupt status flags can be used to indicate the interrupt status even if the interrupt is not enabled for the source. When the interrupt is not enabled for the source, no interrupt is generated to the core even though the interrupt controller still recognizes that the source requested an interrupt. [INT_TRAP_SOURCE](#) is a list of possible trap events, and [INT_SOURCE](#) is a list of possible interrupt sources.

Before the Interrupt Service Routine (ISR) ends, the status of the interrupt needs to be cleared to avoid entering the ISR repeatedly for the same interrupt event. A trap source status can be cleared using the function `PLIB_INT_TrapSourceFlagClear`, and an interrupt source status can be cleared using the function `PLIB_INT_SourceFlagClear`. [INT_TRAP_SOURCE](#) is a list of possible trap events, and [INT_SOURCE](#) is a list of possible interrupt sources.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be <code>INT_ID_0</code> for all the devices which has only interrupt module).
source	One of the possible values from INT_SOURCE .

Returns

None.

Remarks

This function implements an operation of the SourceControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the `PLIB_INT_ExistsSourceControl` API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_SourceEnable(INT_ID_0, INT_SOURCE_TIMER_1);
```

5.6.11.6.2.3 PLIB_INT_SourceFlagClear Function

C

```
void PLIB_INT_SourceFlagClear(
    INT_MODULE_ID index,
    INT_SOURCE source
);
```

Description

This function clears the status of the interrupt flag for the selected source. The flag is set when the interrupt request is identified. The pending interrupt request will not cause further processing if the interrupt is not enabled using the function `PLIB_INT_SourceEnable` or if interrupts are not enabled (on processors that support the [PLIB_INT_Enable](#) feature).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be <code>INT_ID_0</code> for all the devices which has only interrupt module).
source	One of the possible values from INT_SOURCE .

Returns

None.

Remarks

This function implements an operation of the SourceFlag feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsSourceFlag](#) API to write flexible code to automatically determine if this feature is available.

Example

```
if(PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_TIMER_1))
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
}
```

5.6.11.6.2.4 PLIB_INT_SourceFlagGet Function**C**

```
bool PLIB_INT_SourceFlagGet(
    INT_MODULE_ID index,
    INT_SOURCE source
);
```

Description

This function returns the status of the interrupt flag for the selected source. The flag is set when the interrupt request is recognized. The pending interrupt request will not cause further processing if the interrupt is not enabled using the function `PLIB_INT_SourceEnable` or if interrupts are not enabled (on processors that support the [PLIB_INT_Enable](#) feature).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be <code>INT_ID_0</code> for all the devices which has only interrupt module).
source	One of the possible values from INT_SOURCE .

Returns

true - If the interrupt request is recognized for the source
false - If the interrupt request is not recognized for the source

Remarks

This function implements an operation of the SourceFlag feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsSourceFlag](#) API to write flexible code to automatically determine if this feature is available.

Example

```
if(PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_TIMER_1))
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
}
```

5.6.11.6.2.5 PLIB_INT_SourceFlagSet Function**C**

```
void PLIB_INT_SourceFlagSet(
    INT_MODULE_ID index,
    INT_SOURCE source
);
```

Description

This function sets the status of the interrupt flag for the selected source. This function will not be used during normal operation of the system. It is used to generate test interrupts for debug and testing purposes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
source	One of the possible values from INT_SOURCE .

Returns

None.

Remarks

This function implements an operation of the SourceFlag feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsSourceFlag](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_SourceFlagSet(INT_ID_0, INT_SOURCE_TIMER_1);
```

5.6.11.6.2.6 PLIB_INT_SourceIsEnabled Function**C**

```
bool PLIB_INT_SourceIsEnabled(
    INT_MODULE_ID index,
    INT_SOURCE source
);
```

Description

This function gets the enable state of the interrupt source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
source	One of the possible values from INT_SOURCE .

Returns

- true - If the interrupt source is enabled
- false - If the interrupt source is disabled

Remarks

This function implements an operation of the SourceControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsSourceControl](#) API to write flexible code to automatically determine if this feature is available.

Example

```
if(PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_TIMER_1) != true)
{
    PLIB_INT_SourceEnable(INT_ID_0, INT_SOURCE_TIMER_1);
}
```

5.6.11.6.2.7 PLIB_INT_VectorPriorityGet Function**C**

```
INT_PRIORITY_LEVEL PLIB_INT_VectorPriorityGet(
    INT_MODULE_ID index,
    INT_VECTOR vector
);
```

Description

Gets the priority of the interrupt vector. The priority is one of the possible values from [INT_PRIORITY_LEVEL](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
vector	One of the possible values from INT_VECTOR

Returns

priority - One of the possible values from [INT_PRIORITY_LEVEL](#)

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsVectorPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
INT_PRIORITY_LEVEL level;
```

```
level = PLIB_INT_VectorPriorityGet(INT_ID_0, INT_VECTOR_T1);
```

5.6.11.6.2.8 PLIB_INT_VectorPrioritySet Function

C

```
void PLIB_INT_VectorPrioritySet(  
    INT_MODULE_ID index,  
    INT_VECTOR vector,  
    INT_PRIORITY_LEVEL priority  
);
```

Description

Sets the priority of the interrupt vector. The priority is one of the possible values from [INT_PRIORITY_LEVEL](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
vector	One of the possible values from INT_VECTOR
priority	One of the possible values from INT_PRIORITY_LEVEL

Returns

None.

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsVectorPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1, INT_PRIORITY_LEVEL_4);
```

5.6.11.6.2.9 PLIB_INT_VectorSubPriorityGet Function

C

```
INT_SUBPRIORITY_LEVEL PLIB_INT_VectorSubPriorityGet(  
    INT_MODULE_ID index,  
    INT_VECTOR vector  
);
```

Description

This function gets the sub-priority of the interrupt vector. The priority is one of the possible values from [INT_SUBPRIORITY_LEVEL](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
vector	One of the possible values from INT_VECTOR

Returns

priority - One of the possible values from [INT_SUBPRIORITY_LEVEL](#)

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsVectorPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
INT_SUBPRIORITY_LEVEL level;  
level = PLIB_INT_VectorSubPriorityGet(INT_ID_0, INT_VECTOR_T1);
```

5.6.11.6.2.10 PLIB_INT_VectorSubPrioritySet Function**C**

```
void PLIB_INT_VectorSubPrioritySet(  
    INT_MODULE_ID index,  
    INT_VECTOR vector,  
    INT_SUBPRIORITY_LEVEL subPriority  
);
```

Description

This function sets the sub priority of the interrupt vector. The sub-priority is one of the possible values from [INT_SUBPRIORITY_LEVEL](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
vector	One of the possible values from INT_VECTOR
subPriority	One of the possible values from INT_SUBPRIORITY_LEVEL

Returns

None.

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsVectorPriority](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1, INT_SUBPRIORITY_LEVEL_1);
```

5.6.11.6.3 Other Status and Control Functions**5.6.11.6.3.1 PLIB_INT_CPUCurrentPriorityLevelGet Function****C**

```
INT_PRIORITY_LEVEL PLIB_INT_CPUCurrentPriorityLevelGet(  
    INT_MODULE_ID index
```

```
);
```

Description

This function gets the interrupt priority level at which the CPU is currently operating.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

priority - The current interrupt priority of the CPU. Range (0 - 15)

Remarks

User interrupts are disabled when the CPU priority is more than 7.

This function implements an operation of the CPUCurrentPriorityLevel feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsCPUCurrentPriorityLevel](#) API to write flexible code to automatically determine if this feature is available.

Example

```
INT_PRIORITY_LEVEL currentCPUPriority;  
currentCPUPriority = PLIB_INT_CPUCurrentPriorityLevelGet( INT_ID_0 );
```

5.6.11.6.3.2 PLIB_INT_ProximityTimerGet Function**C**

```
uint32_t PLIB_INT_ProximityTimerGet(  
    INT_MODULE_ID index  
);
```

Description

This function returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

Timer reload value.

Remarks

This function implements an operation of the ProximityTimerControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsProximityTimerControl](#) API to write flexible code to automatically determine if this feature is available.

Example

```
uint32_t timer;  
  
timer = PLIB_INT_ProximityTimerGet( INT_ID_0 );
```

5.6.11.6.3.3 PLIB_INT_ProximityTimerSet Function**C**

```
void PLIB_INT_ProximityTimerSet(  
    INT_MODULE_ID index,  
    uint32_t timerreloadvalue  
);
```

Description

This function sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
timerReloadValue	Timer reload value.

Returns

None.

Remarks

This function implements an operation of the ProximityTimerControl feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsProximityTimerControl](#) API to write flexible code to automatically determine if this feature is available.

Example

```
PLIB_INT_ProximityTimerSet(INT_ID_0, 0x12345678);
```

5.6.11.6.3.4 PLIB_INT_TrapSourceFlagClear Function**C**

```
void PLIB_INT_TrapSourceFlagClear(  
    INT_MODULE_ID index,  
    INT_TRAP_SOURCE trapsource  
);
```

Description

This function clears the flag for the selected trap. The flag is set when the request is recognized.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

trapSource	One of the possible values from INT_TRAP_SOURCE

Returns

None.

Remarks

Some processor documentation uses the term "Exception" instead of the term "Trap". For purposes of the Interrupt Peripheral Library, these two terms are synonyms. The difference between a trap (or exception) and an interrupt is that a trap cannot be enabled or disabled, whereas an interrupt can be.

This function implements an operation of the TrapSource feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsTrapSource](#) API to write flexible code to automatically determine if this feature is available.

Example

```
if(PLIB_INT_TrapSourceFlagGet(INT_ID_0, INT_ADDRESS_OVERFLOW_TRAP))
{
    PLIB_INT_TrapSourceFlagClear(INT_ID_0, INT_ADDRESS_OVERFLOW_TRAP);
}
```

5.6.11.6.3.5 PLIB_INT_TrapSourceFlagGet Function**C**

```
bool PLIB_INT_TrapSourceFlagGet(
    INT_MODULE_ID index,
    INT_TRAP_SOURCE trapSource
);
```

Description

This function returns the status of the flag for the selected trap. The flag is set when the trap request is recognized.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).
trapSource	One of the possible values from INT_TRAP_SOURCE

Returns

- true - If the request has been recognized for the trap
- false - If the request is not recognized for the trap

Remarks

Some processor documentation uses the term "Exception" instead of the term "Trap". For purposes of the Interrupt Peripheral Library, these two terms are synonyms. The difference between a trap (or exception) and an interrupt is that a trap cannot be enabled or disabled, whereas an interrupt can be.

This function implements an operation of the TrapSource feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsTrapSource](#) API to write flexible code to automatically determine if this feature is available.

Example

```
if(PLIB_INT_TrapSourceFlagGet(INT_ID_0, INT_ADDRESS_OVERFLOW_TRAP))
{
```

```
    PLIB_INT_TrapSourceFlagClear( INT_ID_0, INT_ADDRESS_OVERFLOW_TRAP );
}
```

5.6.11.6.3.6 PLIB_INT_VectorGet Function

C

```
INT_VECTOR PLIB_INT_VectorGet(
    INT_MODULE_ID index
);
```

Description

This function returns the interrupt vector that is presented to the CPU.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all the devices which has only interrupt module).

Returns

One of the possible values from [INT_VECTOR](#).

Remarks

This value should only be used when the interrupt controller is configured for single vector mode using the function `PLIB_INT_MultiVectorDisable`.

This function implements an operation of the `INTCPUVector` feature. This feature may not be available on all parts. Please refer to the data sheet to determine if this feature is available or call the [PLIB_INT_ExistsINTCPUVector](#) API to write flexible code to automatically determine if this feature is available.

Example

```
INT_VECTOR level = PLIB_INT_VectorGet( INT_ID_0 );
```

5.6.11.6.4 Data Types and Constants

5.6.11.6.4.1 INT_EXTERNAL_SOURCES Enumeration

C

```
typedef enum {
    INT_EXTERNAL_INT_SOURCE0,
    INT_EXTERNAL_INT_SOURCE1,
    INT_EXTERNAL_INT_SOURCE2,
    INT_EXTERNAL_INT_SOURCE3,
    INT_EXTERNAL_INT_SOURCE4,
} INT_EXTERNAL_SOURCES;
```

Description

External Interrupt Sources

This enumeration lists the possible external sources that can cause an interrupt to occur.

Members

Members	Description
INT_EXTERNAL_INT_SOURCE0	External Interrupt Source 0

INT_EXTERNAL_INT_SOURCE1	External Interrupt Source 1
INT_EXTERNAL_INT_SOURCE2	External Interrupt Source 2
INT_EXTERNAL_INT_SOURCE3	External Interrupt Source 3
INT_EXTERNAL_INT_SOURCE4	External Interrupt Source 4

Remarks

This enumeration is processor-specific and is defined in the processor- specific header files (see processor.h).

5.6.11.6.4.2 INT_PRIORITY_LEVEL Enumeration**C**

```
typedef enum {
    INT_PRIORITY_LEVEL0,
    INT_PRIORITY_LEVEL1,
    INT_PRIORITY_LEVEL2,
    INT_PRIORITY_LEVEL3,
    INT_PRIORITY_LEVEL4,
    INT_PRIORITY_LEVEL5,
    INT_PRIORITY_LEVEL6,
    INT_PRIORITY_LEVEL7,
} INT_PRIORITY_LEVEL;
```

Description

Priority Level

This enumeration lists the possible interrupt priority levels that can be selected for each source.

Members

Members	Description
INT_PRIORITY_LEVEL0	Disabled
INT_PRIORITY_LEVEL1	Priority 1
INT_PRIORITY_LEVEL2	Priority 2
INT_PRIORITY_LEVEL3	Priority 3
INT_PRIORITY_LEVEL4	Priority 4
INT_PRIORITY_LEVEL5	Priority 5
INT_PRIORITY_LEVEL6	Priority 6
INT_PRIORITY_LEVEL7	Priority 7

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.11.6.4.3 INT_SOURCE Enumeration**C**

```
typedef enum {
    INT_SOURCE_SOFTWARE_0,
    INT_SOURCE_SOFTWARE_1,
    INT_SOURCE_EXTERNAL_0,
    INT_SOURCE_EXTERNAL_1,
    INT_SOURCE_EXTERNAL_2,
    INT_SOURCE_EXTERNAL_3,
    INT_SOURCE_EXTERNAL_4,
    INT_SOURCE_TIMER_CORE,
    INT_SOURCE_TIMER_1,
    INT_SOURCE_TIMER_2,
```

```
INT_SOURCE_TIMER_3,  
INT_SOURCE_TIMER_4,  
INT_SOURCE_TIMER_5,  
INT_SOURCE_INPUT_CAPTURE_1,  
INT_SOURCE_INPUT_CAPTURE_1_ERROR,  
INT_SOURCE_INPUT_CAPTURE_2,  
INT_SOURCE_INPUT_CAPTURE_2_ERROR,  
INT_SOURCE_INPUT_CAPTURE_3,  
INT_SOURCE_INPUT_CAPTURE_3_ERROR,  
INT_SOURCE_INPUT_CAPTURE_4,  
INT_SOURCE_INPUT_CAPTURE_4_ERROR,  
INT_SOURCE_INPUT_CAPTURE_5,  
INT_SOURCE_INPUT_CAPTURE_5_ERROR,  
INT_SOURCE_OUTPUT_COMPARE_1,  
INT_SOURCE_OUTPUT_COMPARE_2,  
INT_SOURCE_OUTPUT_COMPARE_3,  
INT_SOURCE_OUTPUT_COMPARE_4,  
INT_SOURCE_OUTPUT_COMPARE_5,  
INT_SOURCE_SPI_1_ERROR,  
INT_SOURCE_SPI_1_RECEIVE,  
INT_SOURCE_SPI_1_TRANSMIT,  
INT_SOURCE_SPI_2_ERROR,  
INT_SOURCE_SPI_2_RECEIVE,  
INT_SOURCE_SPI_2_TRANSMIT,  
INT_SOURCE_SPI_3_ERROR,  
INT_SOURCE_SPI_3_RECEIVE,  
INT_SOURCE_SPI_3_TRANSMIT,  
INT_SOURCE_SPI_4_ERROR,  
INT_SOURCE_SPI_4_RECEIVE,  
INT_SOURCE_SPI_4_TRANSMIT,  
INT_SOURCE_USART_1_ERROR,  
INT_SOURCE_USART_1_RECEIVE,  
INT_SOURCE_USART_1_TRANSMIT,  
INT_SOURCE_USART_2_ERROR,  
INT_SOURCE_USART_2_RECEIVE,  
INT_SOURCE_USART_2_TRANSMIT,  
INT_SOURCE_USART_3_ERROR,  
INT_SOURCE_USART_3_RECEIVE,  
INT_SOURCE_USART_3_TRANSMIT,  
INT_SOURCE_USART_4_ERROR,  
INT_SOURCE_USART_4_RECEIVE,  
INT_SOURCE_USART_4_TRANSMIT,  
INT_SOURCE_USART_5_ERROR,  
INT_SOURCE_USART_5_RECEIVE,  
INT_SOURCE_USART_5_TRANSMIT,  
INT_SOURCE_USART_6_ERROR,  
INT_SOURCE_USART_6_RECEIVE,  
INT_SOURCE_USART_6_TRANSMIT,  
INT_SOURCE_I2C_1_ERROR,  
INT_SOURCE_I2C_1_SLAVE,  
INT_SOURCE_I2C_1_MASTER,  
INT_SOURCE_I2C_2_ERROR,  
INT_SOURCE_I2C_2_SLAVE,  
INT_SOURCE_I2C_2_MASTER,  
INT_SOURCE_I2C_3_ERROR,  
INT_SOURCE_I2C_3_SLAVE,  
INT_SOURCE_I2C_3_MASTER,  
INT_SOURCE_I2C_4_ERROR,  
INT_SOURCE_I2C_4_SLAVE,  
INT_SOURCE_I2C_4_MASTER,  
INT_SOURCE_I2C_5_ERROR,  
INT_SOURCE_I2C_5_SLAVE,  
INT_SOURCE_I2C_5_MASTER,  
INT_SOURCE_CHANGE_NOTICE,  
INT_SOURCE_ADC_1,  
INT_SOURCE_PARALLEL_PORT,
```

```

INT_SOURCE_PARALLEL_PORT_ERROR,
INT_SOURCE_COMPARATOR_1,
INT_SOURCE_COMPARATOR_2,
INT_SOURCE_CLOCK_MONITOR,
INT_SOURCE_RTCC,
INT_SOURCE_DMA_0,
INT_SOURCE_DMA_1,
INT_SOURCE_DMA_2,
INT_SOURCE_DMA_3,
INT_SOURCE_DMA_4,
INT_SOURCE_DMA_5,
INT_SOURCE_DMA_6,
INT_SOURCE_DMA_7,
INT_SOURCE_FLASH_CONTROL,
INT_SOURCE_USB_1,
INT_SOURCE_CAN_1,
INT_SOURCE_CAN_2,
INT_SOURCE_ETH_1
} INT_SOURCE;

```

Description

Interrupt Source

This enumeration lists the possible sources that can cause an interrupt to occur.

Members

Members	Description
INT_SOURCE_SOFTWARE_0	Software interrupt 0
INT_SOURCE_SOFTWARE_1	Software interrupt 1
INT_SOURCE_EXTERNAL_0	External interrupt 0
INT_SOURCE_EXTERNAL_1	External interrupt 1
INT_SOURCE_EXTERNAL_2	External interrupt 2
INT_SOURCE_EXTERNAL_3	External interrupt 3
INT_SOURCE_EXTERNAL_4	External interrupt 4
INT_SOURCE_TIMER_CORE	Core timer interrupt
INT_SOURCE_TIMER_1	Timer 1 interrupt
INT_SOURCE_TIMER_2	Timer 2 interrupt
INT_SOURCE_TIMER_3	Timer 3 interrupt
INT_SOURCE_TIMER_4	Timer 4 interrupt
INT_SOURCE_TIMER_5	Timer 5 interrupt
INT_SOURCE_INPUT_CAPTURE_1	Input Capture 1 interrupt
INT_SOURCE_INPUT_CAPTURE_1_ERROR	Input Capture 1 Error interrupt
INT_SOURCE_INPUT_CAPTURE_2	Input Capture 2 interrupt
INT_SOURCE_INPUT_CAPTURE_2_ERROR	Input Capture 2 Error interrupt
INT_SOURCE_INPUT_CAPTURE_3	Input Capture 3 interrupt
INT_SOURCE_INPUT_CAPTURE_3_ERROR	Input Capture 3 Error interrupt
INT_SOURCE_INPUT_CAPTURE_4	Input Capture 4 interrupt
INT_SOURCE_INPUT_CAPTURE_4_ERROR	Input Capture 4 Error interrupt
INT_SOURCE_INPUT_CAPTURE_5	Input Capture 5 interrupt
INT_SOURCE_INPUT_CAPTURE_5_ERROR	Input Capture 5 Error interrupt
INT_SOURCE_OUTPUT_COMPARE_1	Output Compare 1 interrupt
INT_SOURCE_OUTPUT_COMPARE_2	Output Compare 2 interrupt
INT_SOURCE_OUTPUT_COMPARE_3	Output Compare 3 interrupt

INT_SOURCE_OUTPUT_COMPARE_4	Output Compare 4 interrupt
INT_SOURCE_OUTPUT_COMPARE_5	Output Compare 5 interrupt
INT_SOURCE_SPI_1_ERROR	SPI-1 Error interrupt
INT_SOURCE_SPI_1_RECEIVE	SPI-1 Receive Done interrupt
INT_SOURCE_SPI_1_TRANSMIT	SPI-1 Transmit Done interrupt
INT_SOURCE_SPI_2_ERROR	SPI-2 Error interrupt
INT_SOURCE_SPI_2_RECEIVE	SPI-2 Receive Done interrupt
INT_SOURCE_SPI_2_TRANSMIT	SPI-2 Transmit Done interrupt
INT_SOURCE_SPI_3_ERROR	SPI-3 Error interrupt
INT_SOURCE_SPI_3_RECEIVE	SPI-3 Receive Done interrupt
INT_SOURCE_SPI_3_TRANSMIT	SPI-3 Transmit Done interrupt
INT_SOURCE_SPI_4_ERROR	SPI-4 Error interrupt
INT_SOURCE_SPI_4_RECEIVE	SPI-4 Receive Done interrupt
INT_SOURCE_SPI_4_TRANSMIT	SPI-4 Transmit Done interrupt
INT_SOURCE_USART_1_ERROR	UART-1 Error interrupt
INT_SOURCE_USART_1_RECEIVE	UART-1 Receive interrupt
INT_SOURCE_USART_1_TRANSMIT	UART-1 Transmit interrupt
INT_SOURCE_USART_2_ERROR	UART-2 Error interrupt
INT_SOURCE_USART_2_RECEIVE	UART-2 Receive interrupt
INT_SOURCE_USART_2_TRANSMIT	UART-2 Transmit interrupt
INT_SOURCE_USART_3_ERROR	UART-3 Error interrupt
INT_SOURCE_USART_3_RECEIVE	UART-3 Receive interrupt
INT_SOURCE_USART_3_TRANSMIT	UART-3 Transmit interrupt
INT_SOURCE_USART_4_ERROR	UART-4 Error interrupt
INT_SOURCE_USART_4_RECEIVE	UART-4 Receive interrupt
INT_SOURCE_USART_4_TRANSMIT	UART-4 Transmit interrupt
INT_SOURCE_USART_5_ERROR	UART-5 Error interrupt
INT_SOURCE_USART_5_RECEIVE	UART-5 Receive interrupt
INT_SOURCE_USART_5_TRANSMIT	UART-5 Transmit interrupt
INT_SOURCE_USART_6_ERROR	UART-6 Error interrupt
INT_SOURCE_USART_6_RECEIVE	UART-6 Receive interrupt
INT_SOURCE_USART_6_TRANSMIT	UART-6 Transmit interrupt
INT_SOURCE_I2C_1_ERROR	I2C-1 Bus Error Event interrupt
INT_SOURCE_I2C_1_SLAVE	I2C-1 Slave Event interrupt
INT_SOURCE_I2C_1_MASTER	I2C-1 Master Event interrupt
INT_SOURCE_I2C_2_ERROR	I2C-2 Bus Error Event interrupt
INT_SOURCE_I2C_2_SLAVE	I2C-2 Slave Event interrupt
INT_SOURCE_I2C_2_MASTER	I2C-2 Master Event interrupt
INT_SOURCE_I2C_3_ERROR	I2C-3 Bus Error Event interrupt
INT_SOURCE_I2C_3_SLAVE	I2C-3 Slave Event interrupt
INT_SOURCE_I2C_3_MASTER	I2C-3 Master Event interrupt
INT_SOURCE_I2C_4_ERROR	I2C-4 Bus Error Event interrupt
INT_SOURCE_I2C_4_SLAVE	I2C-4 Slave Event interrupt
INT_SOURCE_I2C_4_MASTER	I2C-4 Master Event interrupt
INT_SOURCE_I2C_5_ERROR	I2C-5 Bus Error Event interrupt

INT_SOURCE_I2C_5_SLAVE	I2C-5 Slave Event interrupt
INT_SOURCE_I2C_5_MASTER	I2C-5 Master Event interrupt
INT_SOURCE_CHANGE_NOTICE	Input Change Notice interrupt
INT_SOURCE_ADC_1	Analog-to-Digital Converter 1 interrupt
INT_SOURCE_PARALLEL_PORT	Parallel Master Port interrupt
INT_SOURCE_PARALLEL_PORT_ERROR	Parallel Master Port Error interrupt
INT_SOURCE_COMPARATOR_1	Comparator 1 interrupt
INT_SOURCE_COMPARATOR_2	Comparator 2 interrupt
INT_SOURCE_CLOCK_MONITOR	Fail-Safe Clock Monitor interrupt
INT_SOURCE_RTCC	Real-Time Clock and Calender interrupt
INT_SOURCE_DMA_0	Direct Memory Access Channel 0 interrupt
INT_SOURCE_DMA_1	Direct Memory Access Channel 1 interrupt
INT_SOURCE_DMA_2	Direct Memory Access Channel 2 interrupt
INT_SOURCE_DMA_3	Direct Memory Access Channel 3 interrupt
INT_SOURCE_DMA_4	Direct Memory Access Channel 4 interrupt
INT_SOURCE_DMA_5	Direct Memory Access Channel 5 interrupt
INT_SOURCE_DMA_6	Direct Memory Access Channel 6 interrupt
INT_SOURCE_DMA_7	Direct Memory Access Channel 7 interrupt
INT_SOURCE_FLASH_CONTROL	Flash Control Event interrupt
INT_SOURCE_USB_1	Universal Serial Bus 1 interrupt
INT_SOURCE_CAN_1	Controller Area Network 1 interrupt
INT_SOURCE_CAN_2	Controller Area Network 2 interrupt
INT_SOURCE_ETH_1	Ethernet interrupt

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.11.6.4.4 INT_SUBPRIORITY_LEVEL Enumeration**C**

```
typedef enum {
    INT_SUBPRIORITY_LEVEL0,
    INT_SUBPRIORITY_LEVEL1,
    INT_SUBPRIORITY_LEVEL2,
    INT_SUBPRIORITY_LEVEL3
} INT_SUBPRIORITY_LEVEL;
```

Description

Sub Priority Level

This enumeration lists the possible interrupt sub priority levels that can be selected for each source.

Members

Members	Description
INT_SUBPRIORITY_LEVEL0	Sub Priority 0
INT_SUBPRIORITY_LEVEL1	Sub Priority 1
INT_SUBPRIORITY_LEVEL2	Sub Priority 2
INT_SUBPRIORITY_LEVEL3	Sub Priority 3

Remarks

This enumeration (INT_SUBPRIORITY_LEVEL) is processor specific and is defined in the processor-specific header files (see processor.h).

5.6.11.6.4.5 INT_TRAP_SOURCE Enumeration**C**

```
typedef enum {
    INT_TRAP_ADDRESS_ERROR_LOAD,
    INT_TRAP_ADDRESS_ERROR_STORE,
    INT_TRAP_BUS_ERROR_INSTRUCTION,
    INT_TRAP_BUS_ERROR_DATA,
    INT_TRAP_SYSCALL,
    INT_TRAP_BREAKPOINT,
    INT_TRAP_RESERVED_INSTRUCTION,
    INT_TRAP_UNUSABLE_COPROCESSOR,
    INT_TRAP_ARITHMETIC_OVERFLOW,
    INT_TRAP_GENERAL
} INT_TRAP_SOURCE;
```

Description

Trap Source

This enumeration lists the names of the trap or exception sources.

Members

Members	Description
INT_TRAP_ADDRESS_ERROR_LOAD	Address error exception on a data load or instruction fetch
INT_TRAP_ADDRESS_ERROR_STORE	Address error exception on a data store
INT_TRAP_BUS_ERROR_INSTRUCTION	Bus error exception on an instruction fetch
INT_TRAP_BUS_ERROR_DATA	Bus error exception on a data load or store
INT_TRAP_SYSCALL	Exception do to a Syscall instruction
INT_TRAP_BREAKPOINT	Exception do to hitting a debug breakpoint
INT_TRAP_RESERVED_INSTRUCTION	Exception do to fetching a reserved instruction
INT_TRAP_UNUSABLE_COPROCESSOR	Exception do to fetching a reference to an unusable coprocessor
INT_TRAP_ARITHMETIC_OVERFLOW	Arithmetic Overflow exception
INT_TRAP_GENERAL	General exception trap

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.11.6.4.6 INT_VECTOR Enumeration**C**

```
typedef enum {
    INT_VECTOR_CT,
    INT_VECTOR_CS0,
    INT_VECTOR_CS1,
    INT_VECTOR_INT0,
    INT_VECTOR_T1,
    INT_VECTOR_IC1,
    INT_VECTOR_OC1,
    INT_VECTOR_INT1,
    INT_VECTOR_T2,
    INT_VECTOR_IC2,
```

```
INT_VECTOR_OC2,  
INT_VECTOR_INT2,  
INT_VECTOR_T3,  
INT_VECTOR_IC3,  
INT_VECTOR_OC3,  
INT_VECTOR_INT3,  
INT_VECTOR_T4,  
INT_VECTOR_IC4,  
INT_VECTOR_OC4,  
INT_VECTOR_INT4,  
INT_VECTOR_T5,  
INT_VECTOR_IC5,  
INT_VECTOR_OC5,  
INT_VECTOR_SPI1,  
INT_VECTOR_UART1,  
INT_VECTOR_SPI3,  
INT_VECTOR_I2C3,  
INT_VECTOR_I2C1,  
INT_VECTOR_CN,  
INT_VECTOR_AD1,  
INT_VECTOR_PMP,  
INT_VECTOR_CMP1,  
INT_VECTOR_CMP2,  
INT_VECTOR_UART3,  
INT_VECTOR_SPI2,  
INT_VECTOR_I2C4,  
INT_VECTOR_UART2,  
INT_VECTOR_SPI4,  
INT_VECTOR_I2C5,  
INT_VECTOR_I2C2,  
INT_VECTOR_FSCM,  
INT_VECTOR_RTCC,  
INT_VECTOR_DMA0,  
INT_VECTOR_DMA1,  
INT_VECTOR_DMA2,  
INT_VECTOR_DMA3,  
INT_VECTOR_DMA4,  
INT_VECTOR_DMA5,  
INT_VECTOR_DMA6,  
INT_VECTOR_DMA7,  
INT_VECTOR_FCE,  
INT_VECTOR_USB,  
INT_VECTOR_CAN1,  
INT_VECTOR_CAN2,  
INT_VECTOR_ETH,  
INT_VECTOR_UART4,  
INT_VECTOR_UART6,  
INT_VECTOR_UART5  
} INT_VECTOR;
```

Description

Interrupt Vectors

This enumeration lists the possible interrupt vectors.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.11.6.5 Feature Existence Functions

5.6.11.6.5.1 PLIB_INT_ExistsAlternateVectorTable Function

C

```
bool PLIB_INT_ExistsAlternateVectorTable(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the AlternateVectorTable feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_AlternateVectorTableSelect](#)
- [PLIB_INT_StandardVectorTableSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlternateVectorTable feature is supported on the device
- false - The AlternateVectorTable feature is not supported on the device

Remarks

None.

5.6.11.6.5.2 PLIB_INT_ExistsCPUCurrentPriorityLevel Function

C

```
bool PLIB_INT_ExistsCPUCurrentPriorityLevel(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the CPUCurrentPriorityLevel feature is available on the Interrupt Controller module. When this function returns true, this function is supported on the device:

- [PLIB_INT_CPUCurrentPriorityLevelGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CPUCurrentPriorityLevel feature is supported on the device
- false - The CPUCurrentPriorityLevel feature is not supported on the device

Remarks

None.

5.6.11.6.5.3 PLIB_INT_ExistsEnableControl Function

C

```
bool PLIB_INT_ExistsEnableControl(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_Enable](#)
- [PLIB_INT_Disable](#)
- [PLIB_INT_IsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Remarks

None.

5.6.11.6.5.4 PLIB_INT_ExistsExternalINTEdgeSelect Function

C

```
bool PLIB_INT_ExistsExternalINTEdgeSelect(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the ExternalINTEdgeSelect feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_ExternalRisingEdgeSelect](#)
- [PLIB_INT_ExternalFallingEdgeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the ExternalINTEdgeSelect feature:

- true - The ExternalINTEdgeSelect feature is supported on the device

- false - The ExternalINTEdgeSelect feature is not supported on the device

Remarks

None.

5.6.11.6.5.5 PLIB_INT_ExistsHighPriority Function**C**

```
bool PLIB_INT_ExistsHighPriority(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the HighPriority feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_PriorityHighEnable](#)
- [PLIB_INT_PriorityHighDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The HighPriority feature is supported on the device
- false - The HighPriority feature is not supported on the device

Remarks

None.

5.6.11.6.5.6 PLIB_INT_ExistsINTCPUPriority Function**C**

```
bool PLIB_INT_ExistsINTCPUPriority(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the INTCPUPriority feature is available on the Interrupt Controller module. When this function returns true, this function is supported on the device:

- [PLIB_INT_PriorityGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The INTCPUPriority feature is supported on the device

- false - The INTCPUriority feature is not supported on the device

Remarks

None.

5.6.11.6.5.7 PLIB_INT_ExistsINTCPUVector Function**C**

```
bool PLIB_INT_ExistsINTCPUVector(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the INTCPUVector feature is available on the Interrupt Controller module. When this function returns true, this function is supported on the device:

- [PLIB_INT_VectorGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The INTCPUVector feature is supported on the device
- false - The INTCPUVector feature is not supported on the device

Remarks

None.

5.6.11.6.5.8 PLIB_INT_ExistsINTNesting Function**C**

```
bool PLIB_INT_ExistsINTNesting(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the INTNesting feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_NestingEnable](#)
- [PLIB_INT_NestingDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The INTNesting feature is supported on the device

- false - The INTNesting feature is not supported on the device

Remarks

None.

5.6.11.6.5.9 PLIB_INT_ExistsLowPriority Function**C**

```
bool PLIB_INT_ExistsLowPriority(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the LowPriority feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_PriorityLowEnable](#)
- [PLIB_INT_PriorityLowDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LowPriority feature is supported on the device
- false - The LowPriority feature is not supported on the device

Remarks

None.

5.6.11.6.5.10 PLIB_INT_ExistsPeripheralControl Function**C**

```
bool PLIB_INT_ExistsPeripheralControl(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the PeripheralControl feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_PeripheralsEnable](#)
- [PLIB_INT_PeripheralsDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the PeripheralControl feature:

- true - The PeripheralControl feature is supported on the device
- false - The PeripheralControl feature is not supported on the device

Remarks

None.

5.6.11.6.5.11 PLIB_INT_ExistsPriority Function**C**

```
bool PLIB_INT_ExistsPriority(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the Priority feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_PriorityEnable](#)
- [PLIB_INT_PriorityDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Priority feature is supported on the device
- false - The Priority feature is not supported on the device

Remarks

None.

5.6.11.6.5.12 PLIB_INT_ExistsProximityTimerControl Function**C**

```
bool PLIB_INT_ExistsProximityTimerControl(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the ProximityTimerControl feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_ProximityTimerSet](#)
- [PLIB_INT_ProximityTimerGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ProximityTimerControl feature is supported on the device
- false - The ProximityTimerControl feature is not supported on the device

Remarks

None.

5.6.11.6.5.13 PLIB_INT_ExistsProximityTimerEnable Function**C**

```
bool PLIB_INT_ExistsProximityTimerEnable(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the ProximityTimerEnable feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_ProximityTimerEnable](#)
- [PLIB_INT_ProximityTimerDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ProximityTimerEnable feature is supported on the device
- false - The ProximityTimerEnable feature is not supported on the device

Remarks

None.

5.6.11.6.5.14 PLIB_INT_ExistsSingleVectorShadowSet Function**C**

```
bool PLIB_INT_ExistsSingleVectorShadowSet(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the SingleVectorShadowSet feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_SingleVectorShadowSetDisable](#)
- [PLIB_INT_SingleVectorShadowSetEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SingleVectorShadowSet feature is supported on the device
- false - The SingleVectorShadowSet feature is not supported on the device

Remarks

None.

5.6.11.6.5.15 PLIB_INT_ExistsSourceControl Function**C**

```
bool PLIB_INT_ExistsSourceControl(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the SourceControl feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_SourceEnable](#)
- [PLIB_INT_SourceDisable](#)
- [PLIB_INT_SourcesIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SourceControl feature is supported on the device
- false - The SourceControl feature is not supported on the device

Remarks

None.

5.6.11.6.5.16 PLIB_INT_ExistsSourceFlag Function**C**

```
bool PLIB_INT_ExistsSourceFlag(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the SourceFlag feature is available on the Interrupt Controller module. When this function returns

true, these functions are supported on the device:

- [PLIB_INT_SourceFlagGet](#)
- [PLIB_INT_SourceFlagSet](#)
- [PLIB_INT_SourceFlagClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SourceFlag feature is supported on the device
- false - The SourceFlag feature is not supported on the device

Remarks

None.

5.6.11.6.5.17 PLIB_INT_ExistsTrapSource Function

C

```
bool PLIB_INT_ExistsTrapSource(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the TrapSource feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_TrapSourceFlagGet](#)
- [PLIB_INT_TrapSourceFlagClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TrapSource feature is supported on the device
- false - The TrapSource feature is not supported on the device

Remarks

None.

5.6.11.6.5.18 PLIB_INT_ExistsVectorPriority Function

C

```
bool PLIB_INT_ExistsVectorPriority(  
    INT_MODULE_ID index
```



```
);
```

Description

This function identifies whether the VectorPriority feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_VectorPrioritySet](#)
- [PLIB_INT_VectorPriorityGet](#)
- [PLIB_INT_VectorSubPrioritySet](#)
- [PLIB_INT_VectorSubPriorityGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The VectorPriority feature is supported on the device
- false - The VectorPriority feature is not supported on the device

Remarks

None.

5.6.11.6.5.19 PLIB_INT_ExistsVectorSelect Function

C

```
bool PLIB_INT_ExistsVectorSelect(  
    INT_MODULE_ID index  
);
```

Description

This function identifies whether the VectorSelect feature is available on the Interrupt Controller module. When this function returns true, these functions are supported on the device:

- [PLIB_INT_MultiVectorSelect](#)
- [PLIB_INT_SingleVectorSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The VectorSelect feature is supported on the device
- false - The VectorSelect feature is not supported on the device

Remarks

None.

5.6.11.7 Files

Files

Name	Description
plib_int.h	Defines the Interrupt Peripheral Library interface

Description

5.6.11.7.1 plib_int.h

Interrupt Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Interrupt Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Interrupt Controller module.

Functions

	Name	Description
≡	PLIB_INT_AlternateVectorTableSelect	Enables the use of the alternate vector table.
≡	PLIB_INT_CPUCurrentPriorityLevelGet	Gets the interrupt priority level at which the CPU is currently operating.
≡	PLIB_INT_Disable	Disables the generation of interrupts to the CPU.
≡	PLIB_INT_DISIStatusGet	Returns the status of the DISI instruction status.
≡	PLIB_INT_Enable	Enables the generation of interrupts to the CPU.
≡	PLIB_INT_ExistsAlternateVectorTable	Identifies whether the AlternateVectorTable feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsCPUCurrentPriorityLevel	Identifies whether the CPUCurrentPriorityLevel feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsExternalINTEdgeSelect	Identifies whether the ExternalINTEdgeSelect feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsHighPriority	Identifies whether the HighPriority feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsINTCPUPriority	Identifies whether the INTCPUPriority feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsINTCPUVector	Identifies whether the INTCPUVector feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsINTNesting	Identifies whether the INTNesting feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsLowPriority	Identifies whether the LowPriority feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsPeripheralControl	Identifies whether the PeripheralControl feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsPriority	Identifies whether the Priority feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsProximityTimerControl	Identifies whether the ProximityTimerControl feature exists on the Interrupt Controller module.
≡	PLIB_INT_ExistsProximityTimerEnable	Identifies whether the ProximityTimerEnable feature exists on the Interrupt Controller module.

	PLIB_INT_ExistsSingleVectorShadowSet	Identifies whether the SingleVectorShadowSet feature exists on the Interrupt Controller module.
	PLIB_INT_ExistsSourceControl	Identifies whether the SourceControl feature exists on the Interrupt Controller module.
	PLIB_INT_ExistsSourceFlag	Identifies whether the SourceFlag feature exists on the Interrupt Controller module.
	PLIB_INT_ExistsTrapSource	Identifies whether the TrapSource feature exists on the Interrupt Controller module.
	PLIB_INT_ExistsVectorPriority	Identifies whether the VectorPriority feature exists on the Interrupt Controller module.
	PLIB_INT_ExistsVectorSelect	Identifies whether the VectorSelect feature exists on the Interrupt Controller module.
	PLIB_INT_ExternalFallingEdgeSelect	Selects the falling edge as the edge polarity of the external interrupt.
	PLIB_INT_ExternalRisingEdgeSelect	Selects the rising edge as the edge polarity of the external interrupt.
	PLIB_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.
	PLIB_INT_MultiVectorSelect	Configures the Interrupt Controller for Multiple Vector mode.
	PLIB_INT_NestingDisable	Disables interrupt nesting.
	PLIB_INT_NestingEnable	Enables interrupt nesting.
	PLIB_INT_PeripheralsDisable	Disables the generation of interrupts to the CPU by the peripherals.
	PLIB_INT_PeripheralsEnable	Enables the generation of interrupts to the CPU by the peripherals.
	PLIB_INT_PriorityDisable	Disables interrupt priority levels.
	PLIB_INT_PriorityEnable	Enables interrupt priority levels.
	PLIB_INT_PriorityGet	Returns the priority level of the latest interrupt presented to the CPU.
	PLIB_INT_PriorityHighDisable	Disables the generation of high-priority interrupts to the CPU.
	PLIB_INT_PriorityHighEnable	Enables the generation of high-priority interrupts to the CPU.
	PLIB_INT_PriorityLowDisable	Disables the generation of low-priority interrupts to the CPU.
	PLIB_INT_PriorityLowEnable	Enables generation of low-priority interrupts to the CPU.
	PLIB_INT_ProximityTimerDisable	Disables the interrupt temporal-proximity timer.
	PLIB_INT_ProximityTimerEnable	Enables the interrupt temporal-proximity timer and selects the priority levels that start the timer.
	PLIB_INT_ProximityTimerGet	Returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.
	PLIB_INT_ProximityTimerSet	Sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.
	PLIB_INT_SingleVectorSelect	Configures the Interrupt Controller for Single Vector mode.
	PLIB_INT_SingleVectorShadowSetDisable	Disables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SingleVectorShadowSetEnable	Enables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SourceDisable	Disables the interrupt source
	PLIB_INT_SourceEnable	Enables the interrupt source. This section provides information on initialization and interrupt source setup and handling.
	PLIB_INT_SourceFlagClear	Clears the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagGet	Returns the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagSet	Sets the status of the interrupt flag for the selected source.
	PLIB_INT_SourceIsEnabled	Gets the enable state of the interrupt source.
	PLIB_INT_StandardVectorTableSelect	Enables the use of the standard vector table.

	PLIB_INT_TrapSourceFlagClear	Clears the flag for the selected trap.
	PLIB_INT_TrapSourceFlagGet	Returns the status of the flag for the selected trap.
	PLIB_INT_VectorGet	Returns the interrupt vector that is presented to the CPU.
	PLIB_INT_VectorPriorityGet	Gets the priority of the interrupt vector.
	PLIB_INT_VectorPrioritySet	Sets the priority of the interrupt vector.
	PLIB_INT_VectorSubPriorityGet	Gets the sub-priority of the interrupt vector.
	PLIB_INT_VectorSubPrioritySet	Sets the sub-priority of the interrupt vector.

File Name

plib_int.h

Company

Microchip Technology Inc.

5.6.12 NVM Peripheral Library

5.6.12.1 Introduction

NVM Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Non-Volatile Memory (NVM) component (Flash) on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description**Flash Program Memory**

The Flash program memory is readable, writeable and erasable during normal operation over the entire operating voltage range.

A read from program memory is executed at one byte/word at a time depending on the width of the data bus.

A write to the program memory is executed in either blocks of specific sizes or a single word depending on the type of processor used.

An erase is performed in blocks. A bulk erase may be performed from user code depending on the type of processor supporting the operation.

Writing or erasing program memory will cease instruction fetches until the operation is complete thus restricting memory access and therefore preventing code execution. This is controlled by an internal programming timer.

There are three processor dependant methods for program memory modification.

- Run-Time Self Programming (RTSP)
- In-Circuit Serial Programming (ICSP)
- EJTAG programming

Note: This topic covers the RTSP techniques.

5.6.12.2 Release Notes

MPLAB Harmony Version: v0.70b **NVM Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.12.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.12.4 Using the Library

This topic describes the basic architecture of the NVM Peripheral Library and provides information and examples on how to use it.

Interface Header File: peripheral.h

The interface to the NVM library is defined in the "[plib_nvm.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the NVM library must include "peripheral.h".

Library File:

The NVM peripheral library is part of the processor-specific peripheral library archive (.a) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

5.6.12.4.1 Hardware Abstraction Model

This library provides a low-level abstraction of the NVM module on the Microchip family of microcontrollers with a convenient C

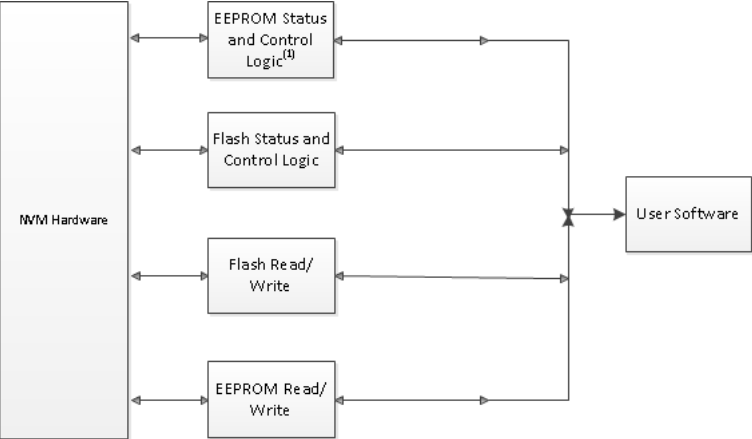
language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Abstraction Model

The NVM Peripheral Library provides interface routines to interact with the blocks shown in the diagram.

NVM Software Abstraction Block Diagram



The **Flash Status and Control logic** ensures that the Flash memory is configured appropriately for modification. It also provides the status of the different operations in progress as well as errors, if any. It also decides if the operation is for Flash program memory or the special device Configuration registers.

The **Flash Read/Write** block ensures that the user data is written to/read from the program memory holding latches. It provides a layer of abstraction over the sequence of processor specific instructions which are required to access this data. Depending on the processor type, either block or Word programming options are available.

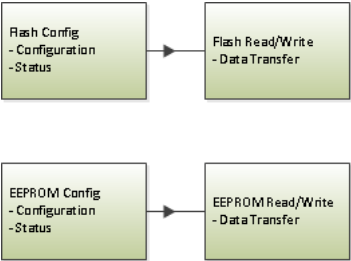
5.6.12.4.2 Library Usage Model

The topic describes the major components of the usage model.

Description

Each of the blocks in the diagram correspond to the library interface section.

Usage Model



Library Interface Section	Description
Flash Configuration	Provides setup, configuration and status control interface routines for Flash program memory.

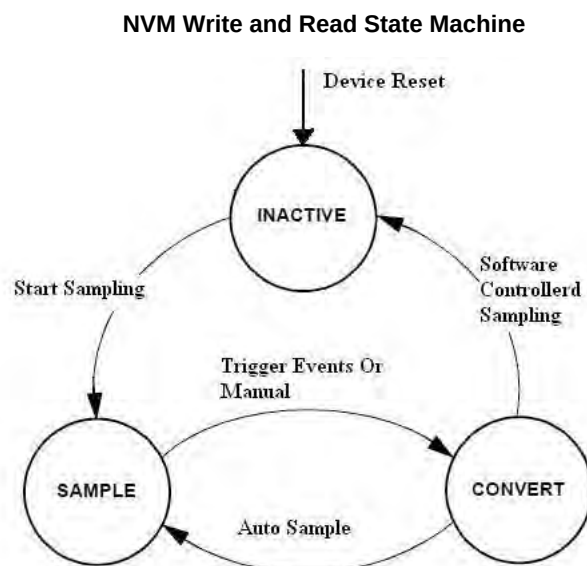
Flash Read/Write	Provides bitwise, pagewise and blockwise Memory access along with NVM keysequence interface routines.

5.6.12.4.3 How the Library Works

The usage model for this library is explained in the following sections.

5.6.12.4.3.1 State Machine

The following state machine is for the NVM Read and Write during the normal operation. This state machine is provided to give a general idea of the usage model of the peripheral library. Refer to the usage model for more detailed steps for the scenario that is being used.



State	Associated Function
Setup and Initialization	Refer to mode of NVM for detailed instructions of setup.
Select Type of memory	Once the NVM has been appropriately set up and initialized, the state machine waits for the application to select the type of memory to be accessed Flash program memory or configuration registers.
Configure and enable write/read	Configures the registers and enables the write/read operation.
Select Address	The application provides the write/read address from which to access memory.
Send Data/Read Data	When the application is polling it can fill data/ read data into the internal holding registers and initiate write via PLIB_NVM_FlashWriteStart(NVM_ID_0) .
Read Error/Write Status	The error/write status is available to the application via PLIB_NVM_FlashWriteCycleHasCompleted(NVM_ID_0) and other status APIs.

Note: Refer to the mode used for the NVM for the setup and initialization steps.

5.6.12.4.3.2 Flash Operations

Note: Flash program memory operations vary across microcontrollers. Please refer to the specific device data sheet or related family reference manual section to determine which features are supported by the device in use.

Flash Read Operation

Setup

1. Provide the Flash program memory from which the data memory has to be read via the API [PLIB_NVM_FlashRead\(\)](#). This address is then updated in the respective address registers or the Table Pointers depending on the device.
2. Ensure the alignment of the address as specified by the specific device data sheet.
3. The size of the data read depends upon the microcontroller in use. For specific details please refer to the Family Reference manual.

Example : Program Memory Read

```
// Determine the address from which to read
#define MY_FLASH_BASE_ADDRESS      0xA0007862 //Or any address in flash to read

uint32_t    address = MY_FLASH_BASE_ADDRESS;
// The Data Available at the memory location
size_t      DataToBeRead;

// where, MY_NVM_INSTANCE - is a specific instance of the hardware peripheral.
// where, address        - The address in the memory from which to read.
DataToBeRead = PLIB_NVM_FlashRead(NVM_ID_0, address);
```

5.6.12.4.3.3 Miscellaneous Functions

Note: Flash program memory operations vary across microcontrollers. Please refer to the specific device data sheet or related family reference manual to determine which features are supported by the device in use.

This section will detail out a few niche and processor specific APIs provided by this library.

Stop in Idle Mode

The [PLIB_NVM_StopInIdleEnable](#) and [PLIB_NVM_StopInIdleDisable](#) functions cater to this functionality. These functions ensure that Flash operations are discontinued in Idle mode.

Low Voltage Detection

The [PLIB_NVM_LowVoltageIsDetected](#) and [PLIB_NVM_LowVoltageEventIsActive](#) functions support low-voltage error detection and triggering of a low-voltage event.

5.6.12.4.4 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.12.4.5 Building the Library

This section lists the files that are available in the \src of the NVM driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.6.12.5 Library Interface

Configuration and Status Functions

	Name	Description
≡	PLIB_NVM_LowVoltageEventsActive	Provides low voltage detection status
≡	PLIB_NVM_LowVoltageIsDetected	Provides low voltage error detection status
≡	PLIB_NVM_MemoryModifyEnable	Allows write cycles to Flash/EEPROM
≡	PLIB_NVM_MemoryModifyInhibit	Inhibits write cycles to Flash/EEPROM
≡	PLIB_NVM_MemoryOperationSelect	Selects the operation to be performed on Flash/EEPROM memory.
≡	PLIB_NVM_StopInIdleDisable	Continues Flash operation when device enters idle mode.
≡	PLIB_NVM_StopInIdleEnable	Discontinues Flash operation when device enters idle mode.
≡	PLIB_NVM_WriteOperationHasTerminated	This routine provides the status of the Flash/EEPROM write operation or sequence.

Feature Existence Functions

	Name	Description
≡	PLIB_NVM_ExistsAccessEnable	Identifies whether the AccessEnableControl feature exists on the NVM module
≡	PLIB_NVM_ExistsAddressModifyControl	Identifies whether the AddressModifyControl feature exists on the NVM module
≡	PLIB_NVM_ExistsBootPageWriteProtect	Identifies whether the BootPageWriteProtect feature exists on the NVM module
≡	PLIB_NVM_ExistsEEPROMReadInitiate	Identifies whether the EEPROMReadInitiate feature exists on the NVM module
≡	PLIB_NVM_ExistsFlashBankRegionSelect	Identifies whether the FlashBankRegionSelect feature exists on the NVM module
≡	PLIB_NVM_ExistsFlashWPMemoryRangeProvide	Identifies whether the FlashWPMemoryRangeProvide feature exists on the NVM module
≡	PLIB_NVM_ExistsKeySequence	Identifies whether the KeySequence feature exists on the NVM module
≡	PLIB_NVM_ExistsLockBootSelect	Identifies whether the LockBootSelect feature exists on the NVM module
≡	PLIB_NVM_ExistsLockPFMSelect	Identifies whether the LockPFMSelect feature exists on the NVM module
≡	PLIB_NVM_ExistsLowVoltageError	Identifies whether the LowVoltageError feature exists on the NVM module
≡	PLIB_NVM_ExistsLowVoltageStatus	Identifies whether the LowVoltageStatus feature exists on the NVM module
≡	PLIB_NVM_ExistsMemoryModificationControl	Identifies whether the MemoryModificationControl feature exists on the NVM module
≡	PLIB_NVM_ExistsOperationMode	Identifies whether the OperationMode feature exists on the NVM module
≡	PLIB_NVM_ExistsProgramEraseOperation	Identifies whether the ProgramEraseOperation feature exists on the NVM module
≡	PLIB_NVM_ExistsProvideData	Identifies whether the ProvideData feature exists on the NVM module
≡	PLIB_NVM_ExistsProvideQuadData	Identifies whether the ProvideQuadData feature exists on the NVM module

	PLIB_NVM_ExistsSourceAddress	Identifies whether the SourceAddress feature exists on the NVM module
	PLIB_NVM_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the NVM module
	PLIB_NVM_ExistsWriteErrorStatus	Identifies whether the WriteErrorStatus feature exists on the NVM module
	PLIB_NVM_ExistsWriteOperation	Identifies whether the WriteOperation feature exists on the NVM module

Flash Memory Functions

	Name	Description
	PLIB_NVM_DataBlockSourceAddress	Takes the address parameter in the argument and loads the base address from which data has to be copied into flash.
	PLIB_NVM_FlashAccessEnable	Allows access to the Flash program memory
	PLIB_NVM_FlashAddressToModify	Takes the address parameter in the argument and loads the address which will be modified by the actual write operation to the appropriate pointer registers .
	PLIB_NVM_FlashEraseOperationSelect	For ERASE and FREE
	PLIB_NVM_FlashEraseStart	Initiates a Program memory erase/write cycle
	PLIB_NVM_FlashProvideData	Provides the user data to intermediate registers before written into flash
	PLIB_NVM_FlashProvideQuadData	Provides low voltage detection status
	PLIB_NVM_FlashRead	Takes the address parameter in the argument and loads the read address to the appropriate register and returns the value read.
	PLIB_NVM_FlashWriteCycleHasCompleted	This routine provides the status of the Flash/EEPROM write cycle.
	PLIB_NVM_FlashWriteKeySequence	Copies the mandatory KEY sequence into the respective registers.
	PLIB_NVM_FlashWriteOperationSelect	Performs erase operation on the memory row selected
	PLIB_NVM_FlashWriteProtectMemoryAreaRange	Provides low voltage detection status
	PLIB_NVM_FlashWriteStart	Initiates a Program memory erase/write cycle
	PLIB_NVM_IsProgramFlashMemoryLocked	Provides low voltage detection status
	PLIB_NVM_ProgramFlashBank1LowerRegion	Provides low voltage detection status
	PLIB_NVM_ProgramFlashBank2LowerRegion	Provides low voltage detection status

Other Functions

	Name	Description
	PLIB_NVM_BootPageWriteProtectionDisable	Provides low voltage detection status
	PLIB_NVM_BootPageWriteProtectionEnable	Provides low voltage detection status
	PLIB_NVM_IsBootMemoryLocked	Provides low voltage detection status
	PLIB_NVM_IsBootPageWriteProtected	Provides low voltage detection status
	PLIB_NVM_LockBootMemory	Provides low voltage detection status
	PLIB_NVM_LockProgramFlashMemory	Provides low voltage detection status

Description

This section describes the Application Programming Interface (API) functions of the NVM Peripheral Library.

Refer to each section for detailed descriptions.

5.6.12.5.1 Configuration and Status Functions

5.6.12.5.1.1 PLIB_NVM_LowVoltageEventIsActive Function

C

```
bool PLIB_NVM_LowVoltageEventIsActive(  
    NVM_MODULE_ID index  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.1.2 PLIB_NVM_LowVoltageIsDetected Function

C

```
bool PLIB_NVM_LowVoltageIsDetected(  
    NVM_MODULE_ID index  
);
```

Description

This routine provides detection of low voltage error status

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage detection and possible data corruption 0 - Voltage Level Acceptable for programming.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageIsDetected(MY_NVM_INSTANCE);
```

5.6.12.5.1.3 PLIB_NVM_MemoryModifyEnable Function**C**

```
void PLIB_NVM_MemoryModifyEnable(  
    NVM_MODULE_ID index  
);
```

Description

This routine allows write cycles to Flash/EEPROM

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsMemoryModificationControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_MemoryModifyEnable(MY_NVM_INSTANCE);
```

5.6.12.5.1.4 PLIB_NVM_MemoryModifyInhibit Function**C**

```
void PLIB_NVM_MemoryModifyInhibit(  
    NVM_MODULE_ID index  
);
```

Description

This routine Inhibits write cycles to Flash/EEPROM

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsMemoryModificationControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_MemoryModifyInhibit(MY_NVM_INSTANCE);
```

5.6.12.5.1.5 PLIB_NVM_MemoryOperationSelect Function**C**

```
void PLIB_NVM_MemoryOperationSelect(  
    NVM_MODULE_ID index,  
    NVM_OPERATION_MODE operationmode  
);
```

Description

This routine selects the operation to be performed on Flash/EEPROM memory.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsOperationMode](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_MemoryOperationSelect(MY_NVM_INSTANCE, NVM_MEMORY_ROW_PROGRAM_OPERATION);
```

5.6.12.5.1.6 PLIB_NVM_StopInIdleDisable Function**C**

```
void PLIB_NVM_StopInIdleDisable(  
    NVM_MODULE_ID index  
);
```

Description

This routine continues Flash operation when device enters idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_StopInIdleDisable(MY_NVM_INSTANCE, DataToWrite);
```

5.6.12.5.1.7 PLIB_NVM_StopInIdleEnable Function**C**

```
void PLIB_NVM_StopInIdleEnable(  
    NVM_MODULE_ID index  
);
```

Description

This routine discontinues Flash operation when device enters idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_StopInIdleEnable(MY_NVM_INSTANCE, DataToWrite);
```

5.6.12.5.1.8 PLIB_NVM_WriteOperationHasTerminated Function**C**

```
bool PLIB_NVM_WriteOperationHasTerminated(  
    NVM_MODULE_ID index  
);
```

Description

This routine provides the status of the Flash/EEPROM write operation or sequence which was initiated by a write/erase operation.

Preconditions

The Address row to be erased must be provided earlier using [PLIB_NVM_FlashAddressToModify\(\)](#). The module should be configured to access Flash memory using [PLIB_NVM_FlashMemoryAccessEnable\(\)](#). A Write cycle must have been initiated via [PLIB_NVM_EEPROMWriteInitiate\(\)](#).

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Write operation prematurely terminated. 0 - Write operation completed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsWriteErrorStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_WriteOperationHasTerminated(MY_NVM_INSTANCE);
```

5.6.12.5.2 Flash Memory Functions

5.6.12.5.2.1 PLIB_NVM_DataBlockSourceAddress Function

C

```
void PLIB_NVM_DataBlockSourceAddress(  
    NVM_MODULE_ID index,  
    uint32_t address  
);
```

Description

This routine takes the address parameter in the argument and loads the base address from which data has to be copied into flash. This is to copy a row of data directly into flash in one iteration without handling any intermediate holding registers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The starting address in the user data memory from which data will be written

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsSourceAddress](#) in your application to determine whether this feature is available.

Example

```
uint32_t address = MY_RAM_BASE_ADDRESS;  
PLIB_NVM_DataBlockSourceAddress(MY_NVM_INSTANCE, address);
```

5.6.12.5.2.2 PLIB_NVM_FlashAccessEnable Function

C

```
void PLIB_NVM_FlashAccessEnable(  
    NVM_MODULE_ID index  
);
```

Description

This routine allows access to the Flash program memory

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsAccessEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_FlashAccessEnable(MY_NVM_INSTANCE);
```

5.6.12.5.2.3 PLIB_NVM_FlashAddressToModify Function**C**

```
void PLIB_NVM_FlashAddressToModify(  
    NVM_MODULE_ID index,  
    uint32_t address  
);
```

Description

This routine takes the address parameter in the argument and loads the address which will be modified by the actual write operation. The write operation following this will write the user data into the program memory.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The starting address in the memory from which data will be written

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsAddressModifyControl](#) in your application to determine whether this feature is available.

Example

```
uint32_t address = MY_FLASH_BASE_ADDRESS;  
PLIB_NVM_FlashAddressToModify(MY_NVM_INSTANCE, address);
```

5.6.12.5.2.4 PLIB_NVM_FlashEraseOperationSelect Function**C**

```
void PLIB_NVM_FlashEraseOperationSelect(  
    NVM_MODULE_ID index  
);
```

Description

For ERASE and FREE

5.6.12.5.2.5 PLIB_NVM_FlashEraseStart Function

C

```
void PLIB_NVM_FlashEraseStart(  
    NVM_MODULE_ID index  
)  
;
```

Description

This routine initiates the program memory erase or write cycles

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsWriteOperation](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_FlashEraseStart(MY_NVM_INSTANCE, DataToWrite);
```

5.6.12.5.2.6 PLIB_NVM_FlashProvideData Function

C

```
void PLIB_NVM_FlashProvideData(  
    NVM_MODULE_ID index,  
    uint32_t data  
)  
;
```

Description

This routine provides the user data to intermediate registers before written into flash.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be written.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsProvideData](#) in your application to determine whether this feature is available.

Example

```
uint32_t DataToWrite;
PLIB_NVM_FlashProvideData(MY_NVM_INSTANCE, DataToWrite);
```

5.6.12.5.2.7 PLIB_NVM_FlashProvideQuadData Function**C**

```
void PLIB_NVM_FlashProvideQuadData(
    NVM_MODULE_ID index,
    uint32_t * data
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.2.8 PLIB_NVM_FlashRead Function**C**

```
uint32_t PLIB_NVM_FlashRead(
    NVM_MODULE_ID index,
    uint32_t address
);
```

Description

This routine takes the address parameter in the argument and loads the read address to the appropriate register. The read operation provides data from the above address.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The address in the memory from which to read

Returns

Data value read at the memory address.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsWriteOperation](#) in your application to determine whether this feature is available.

Example

```
uint32_t    DataToBeRead;
DataToBeRead = PLIB_NVM_FlashRead(MY_NVM_INSTANCE);
```

5.6.12.5.2.9 PLIB_NVM_FlashWriteCycleHasCompleted Function**C**

```
bool PLIB_NVM_FlashWriteCycleHasCompleted(
    NVM_MODULE_ID index
);
```

Description

This routine provides the status of the Flash/EEPROM write cycle which was initiated by a write/erase operation.

Preconditions

The Address row to be erased must be provided earlier using [PLIB_NVM_FlashAddressToModify\(\)](#). The module should be configured to access Flash memory using [PLIB_NVM_FlashMemoryAccessEnable\(\)](#). A Write cycle must have been initiated via [PLIB_NVM_EEPROMWriteInitiate\(\)](#).

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Write/Erase Cycle is incomplete. 0 - Write cycle completed

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsWriteOperation](#) in your application to determine whether this feature is available.

Example

```
bool status;
status = PLIB_NVM_FlashWriteCycleHasCompleted(MY_NVM_INSTANCE);
```

5.6.12.5.2.10 PLIB_NVM_FlashWriteKeySequence Function**C**

```
void PLIB_NVM_FlashWriteKeySequence(
    NVM_MODULE_ID index,
    uint32_t keysequence
);
```

Description

This routine copies the mandatory KEY sequence into the respective registers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

keysequence	Mandatory KEY sequence depending on the controller type

Returns

None.

Remarks

Without the KEY sequence write/erase operation will not be executed.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsKeySequence](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_FlashWriteKeySequence(MY_NVM_INSTANCE, keysequence);
```

5.6.12.5.2.11 PLIB_NVM_FlashWriteOperationSelect Function**C**

```
void PLIB_NVM_FlashWriteOperationSelect(  
    NVM_MODULE_ID index  
);
```

Description

This routine Performs erase operation on the flash memory row selected

Preconditions

The Address row to be erased must be provided earlier using [PLIB_NVM_FlashAddressToModify\(\)](#). The module should be configured to access Flash memory using [PLIB_NVM_FlashMemoryAccessEnable\(\)](#).

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsProgramEraseOperation](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_FlashWriteOperationSelect(MY_NVM_INSTANCE);
```

5.6.12.5.2.12 PLIB_NVM_FlashWriteProtectMemoryAreaRange Function**C**

```
void PLIB_NVM_FlashWriteProtectMemoryAreaRange(  
    NVM_MODULE_ID index,  
    uint32_t address  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.2.13 PLIB_NVM_FlashWriteStart Function**C**

```
void PLIB_NVM_FlashWriteStart(  
    NVM_MODULE_ID index  
);
```

Description

This routine initiates the program memory erase or write cycles

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsWriteOperation](#) in your application to determine whether this feature is available.

Example

```
PLIB_NVM_FlashWriteStart(MY_NVM_INSTANCE);
```

5.6.12.5.2.14 PLIB_NVM_IsProgramFlashMemoryLocked Function**C**

```
bool PLIB_NVM_IsProgramFlashMemoryLocked(  
    NVM_MODULE_ID index  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.2.15 PLIB_NVM_ProgramFlashBank1LowerRegion Function**C**

```
void PLIB_NVM_ProgramFlashBank1LowerRegion(  
    NVM_MODULE_ID index  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.2.16 PLIB_NVM_ProgramFlashBank2LowerRegion Function**C**

```
void PLIB_NVM_ProgramFlashBank2LowerRegion(  
    NVM_MODULE_ID index  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.3 Other Functions

5.6.12.5.3.1 PLIB_NVM_BootPageWriteProtectionDisable Function

C

```
void PLIB_NVM_BootPageWriteProtectionDisable(  
    NVM_MODULE_ID index,  
    NVM_BOOT_MEMORY_PAGE bootPage  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.3.2 PLIB_NVM_BootPageWriteProtectionEnable Function

C

```
void PLIB_NVM_BootPageWriteProtectionEnable(  
    NVM_MODULE_ID index,  
    NVM_BOOT_MEMORY_PAGE bootPage  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.3.3 PLIB_NVM_IsBootMemoryLocked Function**C**

```
bool PLIB_NVM_IsBootMemoryLocked(  
    NVM_MODULE_ID index,  
    NVM_BOOT_MEMORY_AREA memoryArea  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.3.4 PLIB_NVM_IsBootPageWriteProtected Function**C**

```
bool PLIB_NVM_IsBootPageWriteProtected(  
    NVM_MODULE_ID index,
```



```
    NVM_BOOT_MEMORY_PAGE bootPage  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.3.5 PLIB_NVM_LockBootMemory Function

C

```
void PLIB_NVM_LockBootMemory(  
    NVM_MODULE_ID index,  
    NVM_BOOT_MEMORY_AREA memoryArea  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.3.6 PLIB_NVM_LockProgramFlashMemory Function

C

```
void PLIB_NVM_LockProgramFlashMemory(  
    NVM_MODULE_ID index  
);
```

Description

This routine provides detection of low voltage event if any.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Low Voltage Event active. 0 - Low Voltage Event not Active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_NVM_ExistsLowVoltageStatus](#) in your application to determine whether this feature is available.

Example

```
bool status;  
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

5.6.12.5.4 Feature Existence Functions

5.6.12.5.4.1 PLIB_NVM_ExistsAccessEnable Function

C

```
bool PLIB_NVM_ExistsAccessEnable(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the AccessEnableControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_FlashAccessEnable](#)
- [PLIB_NVM_EEPROMAccessEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AccessEnableControl feature is supported on the device

- false - The AccessEnableControl feature is not supported on the device

Remarks

None.

5.6.12.5.4.2 PLIB_NVM_ExistsAddressModifyControl Function**C**

```
bool PLIB_NVM_ExistsAddressModifyControl(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the AddressModifyControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_FlashAddressToModify](#)
- [PLIB_NVM_EEPROMAddressToModify](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AddressModifyControl feature is supported on the device
- false - The AddressModifyControl feature is not supported on the device

Remarks

None.

5.6.12.5.4.3 PLIB_NVM_ExistsBootPageWriteProtect Function**C**

```
bool PLIB_NVM_ExistsBootPageWriteProtect(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the BootPageWriteProtect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_BootPageWriteProtectionEnable](#)
- [PLIB_NVM_BootPageWriteProtectionDisable](#)
- [PLIB_NVM_IsBootPageWriteProtected](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BootPageWriteProtect feature is supported on the device
- false - The BootPageWriteProtect feature is not supported on the device

Remarks

None.

5.6.12.5.4.4 PLIB_NVM_ExistsEEPROMReadInitiate Function**C**

```
bool PLIB_NVM_ExistsEEPROMReadInitiate(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the EEPROMReadInitiate feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMReadStart

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EEPROMReadInitiate feature is supported on the device
- false - The EEPROMReadInitiate feature is not supported on the device

Remarks

None.

5.6.12.5.4.5 PLIB_NVM_ExistsFlashBankRegionSelect Function**C**

```
bool PLIB_NVM_ExistsFlashBankRegionSelect(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the FlashBankRegionSelect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_ProgramFlashBank1LowerRegion](#)
- [PLIB_NVM_ProgramFlashBank2LowerRegion](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FlashBankRegionSelect feature is supported on the device
- false - The FlashBankRegionSelect feature is not supported on the device

Remarks

None.

5.6.12.5.4.6 PLIB_NVM_ExistsFlashWPMemoryRangeProvide Function**C**

```
bool PLIB_NVM_ExistsFlashWPMemoryRangeProvide(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the FlashWPMemoryRangeProvide feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_FlashWriteProtectMemoryAreaRange](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FlashWPMemoryRangeProvide feature is supported on the device
- false - The FlashWPMemoryRangeProvide feature is not supported on the device

Remarks

None.

5.6.12.5.4.7 PLIB_NVM_ExistsKeySequence Function**C**

```
bool PLIB_NVM_ExistsKeySequence(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the KeySequence feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_FlashWriteKeySequence](#)
- PLIB_NVM_EEPROMWriteKeySequence

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The KeySequence feature is supported on the device
- false - The KeySequence feature is not supported on the device

Remarks

None.

5.6.12.5.4.8 PLIB_NVM_ExistsLockBootSelect Function**C**

```
bool PLIB_NVM_ExistsLockBootSelect(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the LockBootSelect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_LockBootMemory](#)
- [PLIB_NVM_IsBootMemoryLocked](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LockBootSelect feature is supported on the device
- false - The LockBootSelect feature is not supported on the device

Remarks

None.

5.6.12.5.4.9 PLIB_NVM_ExistsLockPFMSelect Function**C**

```
bool PLIB_NVM_ExistsLockPFMSelect(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the LockPFMSelect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_LockProgramFlashMemory](#)
- [PLIB_NVM_IsProgramFlashMemoryLocked](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LockPFMSelect feature is supported on the device
- false - The LockPFMSelect feature is not supported on the device

Remarks

None.

5.6.12.5.4.10 PLIB_NVM_ExistsLowVoltageError Function

C

```
bool PLIB_NVM_ExistsLowVoltageError(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the LowVoltageStatus feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_LowVoltageIsDetected](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LowVoltageStatus feature is supported on the device
- false - The LowVoltageStatus feature is not supported on the device

Remarks

None.

5.6.12.5.4.11 PLIB_NVM_ExistsLowVoltageStatus Function

C

```
bool PLIB_NVM_ExistsLowVoltageStatus(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the LowVoltageStatus feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_LowVoltageEventsActive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LowVoltageStatus feature is supported on the device
- false - The LowVoltageStatus feature is not supported on the device

Remarks

None.

5.6.12.5.4.12 PLIB_NVM_ExistsMemoryModificationControl Function

C

```
bool PLIB_NVM_ExistsMemoryModificationControl(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the MemoryModificationControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_MemoryModifyEnable](#)
- [PLIB_NVM_MemoryModifyInhibit](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MemoryModificationControl feature is supported on the device
- false - The MemoryModificationControl feature is not supported on the device

Remarks

None.

5.6.12.5.4.13 PLIB_NVM_ExistsOperationMode Function

C

```
bool PLIB_NVM_ExistsOperationMode(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the OperationMode feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_MemoryOperationSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OperationMode feature is supported on the device
- false - The OperationMode feature is not supported on the device

Remarks

None.

5.6.12.5.4.14 PLIB_NVM_ExistsProgramEraseOperation Function**C**

```
bool PLIB_NVM_ExistsProgramEraseOperation(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the ProgramEraseOperation feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_FlashEraseOperationSelect](#)
- [PLIB_NVM_FlashWriteOperationSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ProgramEraseOperation feature is supported on the device
- false - The ProgramEraseOperation feature is not supported on the device

Remarks

None.

5.6.12.5.4.15 PLIB_NVM_ExistsProvideData Function**C**

```
bool PLIB_NVM_ExistsProvideData(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the ProvideData feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_FlashProvideData](#)
- [PLIB_NVM_EEPROMProvideData](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ProvideData feature is supported on the device
- false - The ProvideData feature is not supported on the device

Remarks

None.

5.6.12.5.4.16 PLIB_NVM_ExistsProvideQuadData Function

C

```
bool PLIB_NVM_ExistsProvideQuadData(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the ProvideQuadData feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_FlashProvideQuadData](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ProvideQuadData feature is supported on the device
- false - The ProvideQuadData feature is not supported on the device

Remarks

None.

5.6.12.5.4.17 PLIB_NVM_ExistsSourceAddress Function

C

```
bool PLIB_NVM_ExistsSourceAddress(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the SourceAddress feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_DataBlockSourceAddress](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SourceAddress feature is supported on the device
- false - The SourceAddress feature is not supported on the device

Remarks

None.

5.6.12.5.4.18 PLIB_NVM_ExistsStopInIdle Function**C**

```
bool PLIB_NVM_ExistsStopInIdle(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_StopInIdleEnable](#)
- [PLIB_NVM_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Remarks

None.

5.6.12.5.4.19 PLIB_NVM_ExistsWriteErrorStatus Function**C**

```
bool PLIB_NVM_ExistsWriteErrorStatus(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the WriteErrorStatus feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_WriteOperationHasTerminated](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WriteErrorStatus feature is supported on the device
- false - The WriteErrorStatus feature is not supported on the device

Remarks

None.

5.6.12.5.4.20 PLIB_NVM_ExistsWriteOperation Function

C

```
bool PLIB_NVM_ExistsWriteOperation(  
    NVM_MODULE_ID index  
);
```

Description

This function identifies whether the WriteOperation feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- [PLIB_NVM_FlashRead](#)
- PLIB_NVM_EEPROMRead
- [PLIB_NVM_FlashWriteStart](#)
- [PLIB_NVM_FlashEraseStart](#)
- PLIB_NVM_EEPROMWriteStart
- PLIB_NVM_EEPROMEraseStart
- [PLIB_NVM_FlashWriteCycleHasCompleted](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WriteOperation feature is supported on the device
- false - The WriteOperation feature is not supported on the device

Remarks

None.

5.6.12.6 Files

Files

Name	Description
plib_nvm.h	NVM PLIB Interface Header for NVM common definitions

Description

5.6.12.6.1 plib_nvm.h

NVM Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the NVM PLIB for all families of Microchip microcontrollers..The definitions in this file are common to NVM peripheral.

Functions

	Name	Description
≡	PLIB_NVM_BootPageWriteProtectionDisable	Provides low voltage detection status
≡	PLIB_NVM_BootPageWriteProtectionEnable	Provides low voltage detection status
≡	PLIB_NVM_DataBlockSourceAddress	Takes the address parameter in the argument and loads the base address from which data has to be copied into flash.
≡	PLIB_NVM_ExistsAccessEnable	Identifies whether the AccessEnableControl feature exists on the NVM module
≡	PLIB_NVM_ExistsAddressModifyControl	Identifies whether the AddressModifyControl feature exists on the NVM module
≡	PLIB_NVM_ExistsBootPageWriteProtect	Identifies whether the BootPageWriteProtect feature exists on the NVM module
≡	PLIB_NVM_ExistsEEPROMReadInitiate	Identifies whether the EEPROMReadInitiate feature exists on the NVM module
≡	PLIB_NVM_ExistsFlashBankRegionSelect	Identifies whether the FlashBankRegionSelect feature exists on the NVM module
≡	PLIB_NVM_ExistsFlashWPMemoryRangeProvide	Identifies whether the FlashWPMemoryRangeProvide feature exists on the NVM module
≡	PLIB_NVM_ExistsKeySequence	Identifies whether the KeySequence feature exists on the NVM module
≡	PLIB_NVM_ExistsLockBootSelect	Identifies whether the LockBootSelect feature exists on the NVM module
≡	PLIB_NVM_ExistsLockPFMSelect	Identifies whether the LockPFMSelect feature exists on the NVM module
≡	PLIB_NVM_ExistsLowVoltageError	Identifies whether the LowVoltageError feature exists on the NVM module
≡	PLIB_NVM_ExistsLowVoltageStatus	Identifies whether the LowVoltageStatus feature exists on the NVM module
≡	PLIB_NVM_ExistsMemoryModificationControl	Identifies whether the MemoryModificationControl feature exists on the NVM module
≡	PLIB_NVM_ExistsOperationMode	Identifies whether the OperationMode feature exists on the NVM module

	PLIB_NVM_ExistsProgramEraseOperation	Identifies whether the ProgramEraseOperation feature exists on the NVM module
	PLIB_NVM_ExistsProvideData	Identifies whether the ProvideData feature exists on the NVM module
	PLIB_NVM_ExistsProvideQuadData	Identifies whether the ProvideQuadData feature exists on the NVM module
	PLIB_NVM_ExistsSourceAddress	Identifies whether the SourceAddress feature exists on the NVM module
	PLIB_NVM_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the NVM module
	PLIB_NVM_ExistsWriteErrorStatus	Identifies whether the WriteErrorStatus feature exists on the NVM module
	PLIB_NVM_ExistsWriteOperation	Identifies whether the WriteOperation feature exists on the NVM module
	PLIB_NVM_FlashAccessEnable	Allows access to the Flash program memory
	PLIB_NVM_FlashAddressToModify	Takes the address parameter in the argument and loads the address which will be modified by the actual write operation to the appropriate pointer registers .
	PLIB_NVM_FlashEraseOperationSelect	For ERASE and FREE
	PLIB_NVM_FlashEraseStart	Initiates a Program memory erase/write cycle
	PLIB_NVM_FlashProvideData	Provides the user data to intermediate registers before written into flash
	PLIB_NVM_FlashProvideQuadData	Provides low voltage detection status
	PLIB_NVM_FlashRead	Takes the address parameter in the argument and loads the read address to the appropriate register and returns the value read.
	PLIB_NVM_FlashWriteCycleHasCompleted	This routine provides the status of the Flash/EEPROM write cycle.
	PLIB_NVM_FlashWriteKeySequence	Copies the mandatory KEY sequence into the respective registers.
	PLIB_NVM_FlashWriteOperationSelect	Performs erase operation on the memory row selected
	PLIB_NVM_FlashWriteProtectMemoryAreaRange	Provides low voltage detection status
	PLIB_NVM_FlashWriteStart	Initiates a Program memory erase/write cycle
	PLIB_NVM_IsBootMemoryLocked	Provides low voltage detection status
	PLIB_NVM_IsBootPageWriteProtected	Provides low voltage detection status
	PLIB_NVM_IsProgramFlashMemoryLocked	Provides low voltage detection status
	PLIB_NVM_LockBootMemory	Provides low voltage detection status
	PLIB_NVM_LockProgramFlashMemory	Provides low voltage detection status
	PLIB_NVM_LowVoltageEventIsActive	Provides low voltage detection status
	PLIB_NVM_LowVoltageIsDetected	Provides low voltage error detection status
	PLIB_NVM_MemoryModifyEnable	Allows write cycles to Flash/EEPROM
	PLIB_NVM_MemoryModifyInhibit	Inhibits write cycles to Flash/EEPROM
	PLIB_NVM_MemoryOperationSelect	Selects the operation to be performed on Flash/EEPROM memory.
	PLIB_NVM_ProgramFlashBank1LowerRegion	Provides low voltage detection status
	PLIB_NVM_ProgramFlashBank2LowerRegion	Provides low voltage detection status
	PLIB_NVM_StopInIdleDisable	Continues Flash operation when device enters idle mode.
	PLIB_NVM_StopInIdleEnable	Discontinues Flash operation when device enters idle mode.

	PLIB_NVM_WriteOperationHasTerminated	This routine provides the status of the Flash/EEPROM write operation or sequence.

5.6.13 Output Compare Peripheral Library

5.6.13.1 Introduction

Output Compare Peripheral Library for Microchip Microcontrollers

This library provides a low-level abstraction of the Output Compare Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Output Compare module is primarily used to generate a single pulse or a series of pulses in response to selected time base events.

Output Compare module operation essentially requires a timer and one or two compare values. Users have an option to select either a 32-bit timer or a 16-bit timer. The timer can be programmed to count at a desired frequency and count up to a desired period value. Details regarding timer operation and setup can be found in the Timer Peripheral Library. The Output Compare module compares the value of the timer with the compare values depending on the selected operating mode. When a compare match occurs between a timer value and compare values, the Output Compare module outputs a change of state on its output pins in accordance with the chosen Output Compare compare mode logic. Either a single pulse or a sequence of pulses may be generated.

Some of the key operating modes of the OC module are:

- **Single Compare Set High/Low modes:** In these compare modes, a compare match between the timer and the buffer (primary compare value) sets the output High/Low. The output remains at the same state after compare match event unless the module is disabled or the mode is changed.
- **Single Compare Toggle mode:** Output toggles state at every compare match event between the timer and the buffer (primary compare value)
- **Dual compare Single/Continuous Pulse modes:** These modes require two compare values. Leading edge of output pulse is generated during compare match of the incrementing timer and the buffer (primary compare value). Trailing edge of output pulse results when a compare match occurs between the incrementing timer and the pulse width value (secondary compare value). The output may be a single pulse or a sequence of pulses.
- **Pulse-Width Modulation (PWM) modes:** In this mode, output goes high when a compare match occurs between timer and the pulse width value (duty cycle). The output is reset back to its initial state when the timer resets after attaining its maximum count.

The Output Compare module also provides programmable interrupt generation on a compare match event. In PWM mode, hardware-based Fault detection and automatic output disable features are provided.

Figure 1 shows a block diagram of the Output Compare module. A compare match between the timer value and the compare values generates a pulse at the output.

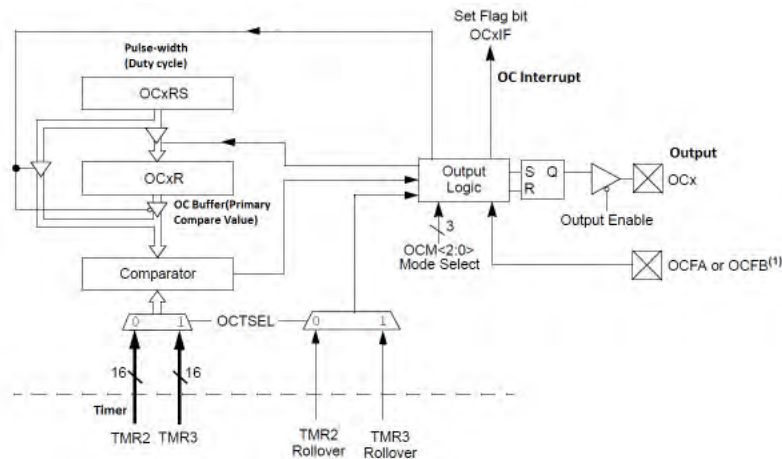


Figure 1: Output Compare Module Block Diagram

Figure 2 shows a block diagram of the Output Compare module with dedicated timers, present on some devices. It facilitates the use of multiple Output Compare modules operating synchronously or the use of an asynchronous trigger to generate a pulse.

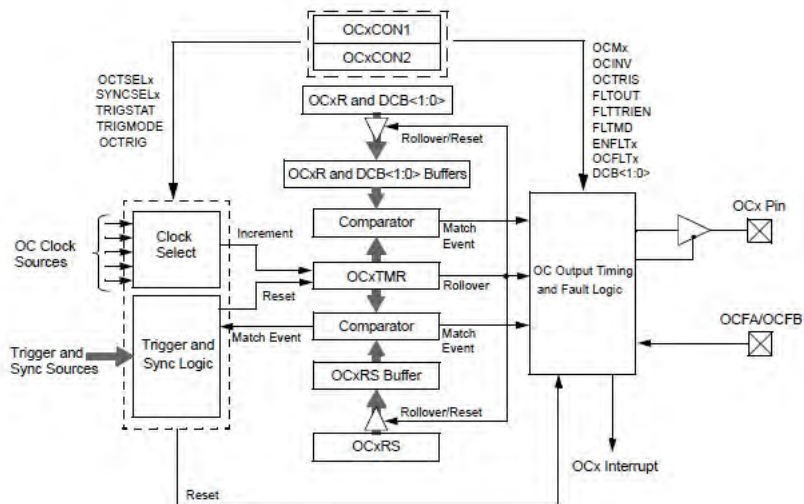


Figure 2: Output Compare Module Block Diagram with Dedicated Timers

5.6.13.2 Release Notes

MPLAB Harmony Version: v0.70b **Output Compare Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.13.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.13.4 Using the Library

This topic describes the basic architecture of the Output Compare Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_oc.h](#)

The interface to the Output Compare Peripheral library is defined in the "[plib_oc.h](#)" header file. This file is included by the "peripheral.h" file. Any C language source (.c) file that uses the Output Compare Peripheral Library should include "peripheral.h".

Library File:

The Output Compare Peripheral library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the Peripheral interacts with the framework.

5.6.13.4.1 Hardware Abstraction Model

This library provides the low-level abstraction of the Output Compare module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all the functionality of the available Output Compare modules on the device. Refer to the specific device data sheet or related family reference manual section to determine the set of functions that are supported for each Output Compare module on your device.

Description

The Output Compare Peripheral Library is used to simplify low-level access to the Output Compare module without having the need to directly interact the module's registers. The function names are generic and lead to easier access to the Output Compare module on different devices.

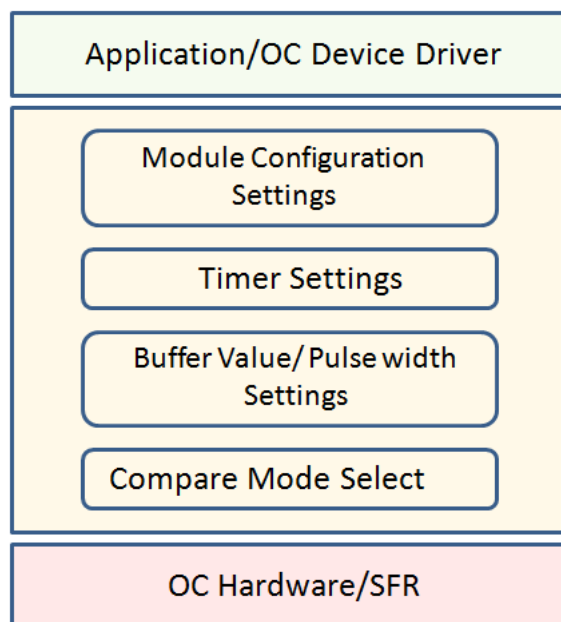


Figure 3: Output Compare module Software Abstraction Block Diagram

The Output Compare module can be described from a software standpoint as having functions to configure the module itself, to select the timer, set the buffer and pulse width values, and select the mode of operation. The desired nature of the output pulse is thus obtained.

A 16-bit or a 32-bit timer can be selected for the output compare time base. In some devices, the Output Compare module can be synchronized to an external input source. The Output Compare module can also operate asynchronously, based on a trigger by an input source. The Output Compare module entails the use of some functions dealing with Timer set up. These details can be found in Timer Peripheral Library.

Functions have been provided to set buffer values and pulse-width values. Pulse-width values are used only in Dual compare and PWM modes. In PWM mode, the buffer value provides the initial duty cycle for the first time period, while all later time cycles employ the 'pulse-width' value as the duty cycle value.

Compare modes:

- Single Compare Level Mode: Sets the output of Output Compare module at either 'High' or 'Low' at a compare match event
- Single Compare Toggle Mode: Toggles the output of OC module at each compare match event. This mode will produce continuous pulses.
- Dual Compare Mode: Output Compare module output is driven 'High' on compare match with buffer value and driven 'Low' on a compare match with the pulse-width value. The output can be either a single pulse or a continuous stream of pulses.
- PWM Mode: In Edge-aligned PWM mode, the Output Compare module output is driven 'High' whenever the timer resets, and is driven 'Low' on a compare match with the pulse-width value. Faults may or may not be enabled. In Center-aligned PWM mode, the Output Compare module output is driven 'High' on a compare match with the buffer value and driven 'Low' on a compare match with the pulse-width value.

Note: In PWM mode, it is vital that the user selects the mode of operation before selecting a Fault input.

5.6.13.4.2 How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

All of the interface functions in this library are classified according to:

- General Setup: These routines deal with general setup of Output Compare module
- Compare/PWM Mode: These routines facilitate the selection of a Compare/PWM mode and setting up the compare values
- Timer: These functions deal with configuring the timer, selection of the clock source, and setting up a synchronous or asynchronous mode of operation for the OC module
- Power-Saving Modes:
 - Sleep Mode: The Output Compare module output is driven to the same state as it was in prior to the device entering Sleep mode. In PWM Fault mode, Fault detection is active. A Fault forces output of the OC module to tri-state or to a fixed predetermined state.
 - Idle Mode: The Output Compare module can be configured to suspend its operations or continue its operations. If the Output Compare module suspends its operations, it has the same behavior as it does in Sleep mode.
- Faults: These functions are used to select Fault inputs and identify occurrences of Faults when the Output Compare module functions in PWM mode
- Exists: These functions notify whether or not a particular feature is present on a device

The following sections provide examples that depict the use of interface functions to perform general tasks such as initialization and set up of the Output Compare module and setting up the Output Compare module in different compare modes.

5.6.13.4.2.1 Single Compare Set High Mode

The Single Compare Set High mode sets the output of the Output Compare module 'High' at a compare match event. This section illustrates the steps required to configure the Output Compare module in Single Compare Set High mode.

Example:

```
/* This example illustrates setting up Output Compare Module in Single Compare Set High mode */

/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for the API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);

/* Select Timer2 as timebase */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);

/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode */
PLIB_OC_ModeSelect(MY_OC_ID, OC_SET_HIGH_SINGLE_PULSE_MODE);

/* Set buffer size to 16-bits */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set buffer value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/* Configure interrupts associated with Output Compare module. Refer to Timer Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable the Output Compare module */
PLIB_OC_Enable(MY_OC_ID);
```

5.6.13.4.2.2 Single Compare Toggle Mode

The Single Compare Toggle mode toggles the output of the Output Compare module at each compare match event. This section illustrates the steps required to configure the Output Compare module in Single Compare Toggle mode.

Example:

```
/* This example illustrates setting up Output Compare Module in Single Compare Toggle mode */  
  
/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for the API */  
  
/* Disable OC module */  
PLIB_OC_Disable(MY_OC_ID);  
  
/* Select Timer2 as timebase */  
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);  
  
/* Set period of time base. Refer to Timer Peripheral Library for the API */  
  
/* Select compare mode */  
PLIB_OC_ModeSelect(MY_OC_ID, OC_TOGGLE_CONTINUOUS_PULSE_MODE);  
  
/* Set buffer size to 16-bits */  
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);  
  
/* Set Buffer Value */  
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);  
  
/* Configure interrupts associated with Output Compare module. Refer to Interrupts Peripheral Library for the API */  
  
/* Enable Timer2. Refer to Timer Peripheral Library for the API */  
  
/* Enable Output Compare module */  
PLIB_OC_Enable(MY_OC_ID);
```

5.6.13.4.2.3 Dual Compare Continuous Mode

In this mode, the Output Compare module output is driven 'High' on a compare match with the buffer value and driven 'Low' on a compare match with the pulse-width value. A continuous stream of pulses is generated. This section illustrates the steps required to configure the Output Compare module in Dual Compare Continuous Pulse mode.

Example:

```
/* This example illustrates setting up Output Compare Module in Dual Compare Continuous Pulse mode */  
  
/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for the API */  
  
/* Disable OC module */  
PLIB_OC_Disable(MY_OC_ID);  
  
/* Select Timer2 as timebase */  
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);  
  
/* Set period of time base. Refer to Timer Peripheral Library for the API */  
  
/* Select compare mode */  
PLIB_OC_ModeSelect(MY_OC_ID, OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE);  
  
/* Set buffer size to 16-bits */  
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);
```

```

/* Set buffer(primary compare) value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/*Set pulse width(secondary compare) value */
PLIB_OC_PulseWidth16BitSet(MY_OC_ID, 0x0FFF);

/* Configure interrupts associated with Output Compare module. Refer to
Interrupts Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable the Output Compare module */
PLIB_OC_Enable(MY_OC_ID);

```

5.6.13.4.2.4 PWM Mode with Enabled Faults

This section illustrates the steps required to configure the Output Compare module in PWM mode.

Note: In PWM mode, it is vital that the user selects the mode of operation before selecting a Fault input.

Example:

```

/* This example illustrates setting up Output Compare Module in Dual Compare Continuous Pulse
mode */

/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for the API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);

/* Select Timer2 as timebase */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);

/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode. PWM Edge aligned mode is selected */
PLIB_OC_ModeSelect(MY_OC_ID, OC_COMPARE_PWM_EDGE_ALIGNED_MODE );

/* Select Fault input. In Output Compare modules that do not have dedicated timers,
fault selection is preset in hardware */
PLIB_OC_FaultInputSelect(MY_OC_ID, OC_FAULT_PRESET);

/* Set buffer size to 16-bits. Refer to Timer Peripheral Library for the API */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set buffer (initial duty cycle) Value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/*Set pulse width (Duty cycle) value */
PLIB_OC_PulseWidth16BitSet(MY_OC_ID, 0x0FFF);

/* Configure interrupts associated with Output Compare module. Refer to
Interrupts Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable OC module */
PLIB_OC_Enable(MY_OC_ID);

/* Check for PWM Fault */
while(!PLIB_OC_FaultHasOccurred)
{
    /* If no PWM fault, continue normal operation*/
}

```

5.6.13.4.3 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.13.5 Library Interface

Data Types and Constants

	Name	Description
	<u>OC_16BIT_TIMERS</u>	Lists different 16 bit time bases for OC module
	<u>OC_BUFFER_SIZE</u>	Lists different buffer sizes for compare value
	<u>OC_CLOCK_RESOLUTION</u>	Lists different PWM resolutions for PWM mode and OC output falling edge delay for compare modes
	<u>OC_COMPARE_MODES</u>	Lists different compare modes for OC module
	<u>OC_FAULT_MODES</u>	Lists modes to clear fault mode
	<u>OC_FAULT_OUT</u>	Provides state of OC output pin when a fault occurs
	<u>OC_FAULT_TRISTATE</u>	Lists modes to clear fault mode
	<u>OC_FAULTS</u>	Lists different fault sources for OC module
	<u>OC_MODULE_ID</u>	OC_MODULE_ID This enumeration defines number of modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.
	<u>OC_SYNC_MODES</u>	Lists different synchronization modes for OC module
	<u>OC_SYNC_SOURCES</u>	Lists different synchronization sources for OC module
	<u>OC_TRIGGER_STATUS_MODES</u>	Lists different trigger status reset modes for OC module

Compare Mode Functions

	Name	Description
	<u>PLIB_OC_Buffer16BitSet</u>	Sets a 16-bit primary compare value for compare operations.
	<u>PLIB_OC_Buffer32BitSet</u>	Sets a 32-bit primary compare value for compare operations.
	<u>PLIB_OC_BufferSizeSelect</u>	Sets the buffer size and pulse width to 16-bits or 32-bits.
	<u>PLIB_OC_PulseWidth16BitSet</u>	Sets a 16-bit pulse width for OC module output.
	<u>PLIB_OC_PulseWidth32BitSet</u>	Sets a 32-bit pulse width for OC module output.
	<u>PLIB_OC_PWMdutyCycleResolutionSet</u>	Sets the PWM duty cycle resolution and delays the falling edge of OC output in compare modes.

Faults Functions

	Name	Description
	<u>PLIB_OC_FaultHasOccurred</u>	Checks if a PWM fault has occurred.
	<u>PLIB_OC_FaultInputSelect</u>	Selects a Fault input for the OC module.
	<u>PLIB_OC_FaultModeSelect</u>	Selects an automatic or software-based mode to clear the Fault.
	<u>PLIB_OC_FaultOutSelect</u>	OC output pin is driven to a known state on a Fault.
	<u>PLIB_OC_FaultTristateSelect</u>	The OC module output can be tristated or driven to a known state when a Fault occurs.

Feature Existence Functions

	Name	Description
	PLIB_OC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the OC module.
	PLIB_OC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the OC module.
	PLIB_OC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the OC module.
	PLIB_OC_ExistsCompareModeSelect	Identifies whether the CompareModeSelect feature exists on the OC module.
	PLIB_OC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the OC module.
	PLIB_OC_ExistsFaultInput	Identifies whether the FaultInput feature exists on the OC module.
	PLIB_OC_ExistsFaultModeSelect	Identifies whether the FaultModeSelect feature exists on the OC module.
	PLIB_OC_ExistsFaultOutSelect	Identifies whether the FaultOut feature exists on the OC module.
	PLIB_OC_ExistsFaultStatus	Identifies whether the FaultStatus feature exists on the OC module.
	PLIB_OC_ExistsFaultTristateSelect	Identifies whether the FaultTristateSelect feature exists on the OC module.
	PLIB_OC_ExistsPolarityInvert	Identifies whether the PolarityInvert feature exists on the OC module.
	PLIB_OC_ExistsPulseWidth	Identifies whether the PulseWidth feature exists on the OC module.
	PLIB_OC_ExistsPWMDutyCycleResolutionControl	Identifies whether the PWMDutyCycleResolution feature exists on the OC module.
	PLIB_OC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the OC module.
	PLIB_OC_ExistsSyncModeSelect	Identifies whether the SyncModeSelect feature exists on the OC module.
	PLIB_OC_ExistsSyncSourceSelect	Identifies whether the SyncSourceSelect feature exists on the OC module.
	PLIB_OC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the OC module.
	PLIB_OC_ExistsTimerTriggered	Identifies whether the TimerTriggered feature exists on the OC module.
	PLIB_OC_ExistsTriggerControl	Identifies whether the TriggerControl feature exists on the OC module.
	PLIB_OC_ExistsTriggerStatusModeSelect	Identifies whether the TriggerStatusModeSelect feature exists on the OC module.

General Setup Functions

	Name	Description
	PLIB_OC_Disable	Disable the OC module
	PLIB_OC_Enable	Enables the OC module.
	PLIB_OC_ModeSelect	Selects the compare mode for the OC module.
	PLIB_OC_PolarityInvertedDisable	Disables OC module output inversion polarity.
	PLIB_OC_PolarityInvertedEnable	Output compare polarity is inverted.

Power-Saving Modes Functions

	Name	Description
≡	PLIB_OC_StopInIdleDisable	OC module continues operating when the device enters Idle mode
≡	PLIB_OC_StopInIdleEnable	Discontinues OC module operation when the device enters Idle mode

Timer Functions

	Name	Description
≡	PLIB_OC_IsTriggered	Checks whether or not a timer source has been triggered.
≡	PLIB_OC_SyncModeSelect	Selects the synchronization mode for the OC module.
≡	PLIB_OC_SyncSourceSelect	Selects the input source for synchronous or asynchronous (triggered) operation of the OC module.
≡	PLIB_OC_TimerSelect	Selects a clock source for the OC module.
≡	PLIB_OC_TriggerClear	Clears the trigger status for the OC module.
≡	PLIB_OC_TriggerSet	Sets the trigger status for the OC module.
≡	PLIB_OC_TriggerStatusModeSelect	Selects the trigger status reset mode.
≡	PLIB_OC_AlternateClockDisable	Selects Timer 2/3 instead of the alternate clock source.
≡	PLIB_OC_AlternateClockEnable	Selects the alternate clock source.

Description**5.6.13.5.1 General Setup Functions****5.6.13.5.1.1 PLIB_OC_Disable Function****C**

```
void PLIB_OC_Disable(
    OC_MODULE_ID index
);
```

Description

This function disables the OC module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the OC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
```



```
PLIB_OC_Disable(MY_OC_ID);
```

5.6.13.5.1.2 PLIB_OC_Enable Function

C

```
void PLIB_OC_Enable(  
    OC_MODULE_ID index  
);
```

Description

This function enables the OC module.

Preconditions

The module should be appropriately configured before being enabled.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1  
  
//Do all the other configurations before enabling.  
  
PLIB_OC_Enable(MY_OC_ID);
```

5.6.13.5.1.3 PLIB_OC_ModeSelect Function

C

```
void PLIB_OC_ModeSelect(  
    OC_MODULE_ID index,  
    OC_COMPARE_MODES cmpMode  
);
```

Description

This function selects the compare mode for the OC module.

Preconditions

The OC module must be turned off before a new mode is selected. The OC module is turned off via `PLIB_OC_ModeSelect(MY_OC_ID,OC_COMPARE_TURN_OFF_MODE)`

Parameters

Parameters	Description
index	Identifies the desired OC module
cmpMode	Identifies the compare mode for OC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsCompareModeSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
// Dual compare continuous pulse mode is selected
PLIB_OC_ModeSelect(MY_OC_ID, OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE );
```

5.6.13.5.1.4 PLIB_OC_PolarityInvertedDisable Function**C**

```
void PLIB_OC_PolarityInvertedDisable(
    OC_MODULE_ID index
);
```

Description

This function disables polarity inversion of OC module output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsPolarityInvert](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
// Polarity of OC module output is Non-inverted
PLIB_OC_PolarityInvertedDisable(OC_MODULE_ID index);
```

5.6.13.5.1.5 PLIB_OC_PolarityInvertedEnable Function**C**

```
void PLIB_OC_PolarityInvertedEnable(
    OC_MODULE_ID index
);
```

Description

This function inverts the polarity of the OC module output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsPolarityInvert](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
// Invert polarity of OC module output
PLIB_OC_PolarityInvertedEnable(MY_OC_ID);
```

5.6.13.5.2 Compare Mode Functions

5.6.13.5.2.1 PLIB_OC_Buffer16BitSet Function

C

```
void PLIB_OC_Buffer16BitSet(
    OC_MODULE_ID index,
    uint16_t val16Bit
);
```

Description

This function sets a 16-bit primary compare value for compare operations in all modes except PWM modes.

Preconditions

The PWM mode of operation should not be selected. The buffer size should be set to 16-bits by the [PLIB_OC_BufferSizeSelect\(\)](#) function.

Parameters

Parameters	Description
index	Identifies the desired OC module
val16Bit	Sets a 16-bit primary compare value

Returns

None

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsBufferValue](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);
```

5.6.13.5.2.2 PLIB_OC_Buffer32BitSet Function

C

```
void PLIB_OC_Buffer32BitSet(
    OC_MODULE_ID index,
    uint32_t val32Bit
);
```

Description

This function sets a 32-bit primary compare value for compare operations in all modes except PWM modes.

Preconditions

The PWM mode of operation should not be selected. The buffer size should be set to 32-bits by the [PLIB_OC_BufferSizeSelect\(\)](#) function.

Parameters

Parameters	Description
index	Identifies the desired OC module
val32Bit	Sets a 32-bit primary compare value

Returns

None

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsBufferValue](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_Buffer32BitSet(MY_OC_ID, 0x000000FF);
```

5.6.13.5.2.3 PLIB_OC_BufferSizeSelect Function**C**

```
void PLIB_OC_BufferSizeSelect(
    OC_MODULE_ID index,
    OC_BUFFER_SIZE size
);
```

Description

This function sets the size of the buffer and pulse width to 16-bits or 32-bits. The choice is made based on whether a 16-bit timer or a 32-bit timer is selected.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module
size	Identifies the size of compare value

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsBufferSize](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
// Buffer size and pulse width size are set to 32-bits
```

```
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_32BIT);
```

5.6.13.5.2.4 PLIB_OC_PulseWidth16BitSet Function

C

```
void PLIB_OC_PulseWidth16BitSet(  
    OC_MODULE_ID index,  
    uint16_t pulsewidth  
);
```

Description

This function sets a 16-bit pulse width for the OC module in dual compare modes. A dual compare mode can be selected using the [PLIB_OC_ModeSelect\(\)](#) function. Secondary compare match event (pulse width) decides the trailing (falling) edge of the OC module output.

Preconditions

Dual compare operation should be selected. The buffer size should be set to 16-bits by the [PLIB_OC_BufferSizeSelect\(\)](#) function.

Parameters

Parameters	Description
index	Identifies the desired OC module
pulseWidth	Pulse width value

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsPulseWidth](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1  
  
PLIB_OC_PulseWidth16BitSet(MY_OC_ID, 0x0FFF);
```

5.6.13.5.2.5 PLIB_OC_PulseWidth32BitSet Function

C

```
void PLIB_OC_PulseWidth32BitSet(  
    OC_MODULE_ID index,  
    uint32_t pulsewidth  
);
```

Description

This function sets a 32-bit pulse width for OC module in dual compare modes. A dual compare mode can be selected using [PLIB_OC_ModeSelect\(\)](#) routine. Secondary compare match event (pulse width) decides the trailing(falling) edge of OC module output.

Preconditions

Dual compare operation should be selected. The buffer size should be set to 32-bits by the [PLIB_OC_BufferSizeSelect\(\)](#) function.

Parameters

Parameters	Description
index	Identifies the desired OC module
pulseWidth	Pulse width value

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsPulseWidth](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_PulseWidth32BitSet(MY_OC_ID, 0x00000FFF);
```

5.6.13.5.2.6 PLIB_OC_PWMDutyCycleResolutionSet Function**C**

```
void PLIB_OC_PWMDutyCycleResolutionSet(
    OC_MODULE_ID index,
    OC_CLOCK_RESOLUTION clkResolution
);
```

Description

This function sets the PWM duty cycle resolution in PWM mode and delays the falling edge of OC output in compare modes.

Preconditions

Dual compare mode must be selected for this function.

Parameters

Parameters	Description
index	Identifies the desired OC module
clkResolution	Select clock resolution

Returns

None

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsPWMDutyCycleResolutionControl](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_PWMDutyCycleResolutionSet(MY_OC_ID, OC_CLOCK_P1);
```

5.6.13.5.3 Timer Functions**5.6.13.5.3.1 PLIB_OC_IsTriggered Function****C**

```
bool PLIB_OC_IsTriggered(
    OC_MODULE_ID index
);
```

Description

This function checks whether Output Compare timer has been triggered or not. This function returns 'true' if OC timer is triggered

and running and 'false' if the OC timer is cleared and stopped.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

bool - true - OC timer is triggered and running

- false - OC timer is stopped and held in clear

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsTimerTriggered](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
if(!PLIB_OC_IsTriggered(MY_OC_ID))
{
    // Do some operation
};
```

5.6.13.5.3.2 PLIB_OC_SyncModeSelect Function

C

```
void PLIB_OC_SyncModeSelect(
    OC_MODULE_ID index,
    OC_SYNC_MODES mode
);
```

Description

This function selects the synchronization mode for Othe OC module. The Synchronous or Asynchronous mode of operation can be selected.

Preconditions

If Asynchronous mode is selected and an OC trigger is selected, that OC module must be disabled first before enabling the trigger functionality.

Parameters

Parameters	Description
index	Identifies the desired OC module
mode	Identifies the mode (Synchronous or Asynchronous)

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsSyncModeSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
// Synchronous mode of operation is selected.
```

```
PLIB_OC_SyncModeSelect(MY_OC_ID, OC_MODE_SYNC);
```

5.6.13.5.3.3 PLIB_OC_SyncSourceSelect Function

C

```
void PLIB_OC_SyncSourceSelect(  
    OC_MODULE_ID index,  
    OC_SYNC_SOURCES src  
);
```

Description

This function selects the input source for synchronous or asynchronous (triggered) operation of the OC module.

Preconditions

If an OC trigger is selected, that OC module must be disabled before it is configured to be used as a trigger to the next OC module.

Parameters

Parameters	Description
index	Identifies the desired OC module
src	Identifies the input source

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsSyncSourceSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1  
//Input Capture is selected as the input source  
PLIB_OC_SyncSourceSelect(MY_OC_ID, OC_SYNC_SOURCE_IC6);
```

5.6.13.5.3.4 PLIB_OC_TimerSelect Function

C

```
void PLIB_OC_TimerSelect(  
    OC_MODULE_ID index,  
    OC_16BIT_TIMERS tmr  
);
```

Description

This function selects a clock source for the OC module if the 16-bit time base is set.

Preconditions

The 16-bit time base needs to be set.

Parameters

Parameters	Description
index	Identifies the desired OC module
tmr	Identifies the timer

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsTimerSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
// 16-bit Timer2 is selected
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);
```

5.6.13.5.3.5 PLIB_OC_TriggerClear Function**C**

```
void PLIB_OC_TriggerClear(
    OC_MODULE_ID index
);
```

Description

This function clears the trigger status for the OC module.

Preconditions

Dual compare mode must be selected for this function.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

None

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsTriggerControl](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_TriggerClear(MY_OC_ID);
```

5.6.13.5.3.6 PLIB_OC_TriggerSet Function**C**

```
void PLIB_OC_TriggerSet(
    OC_MODULE_ID index
);
```

Description

This function sets the trigger status for the OC module.

Preconditions

Dual compare mode must be selected for this function.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsTriggerControl](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_TriggerSet(MY_OC_ID);
```

5.6.13.5.3.7 PLIB_OC_TriggerStatusModeSelect Function**C**

```
void PLIB_OC_TriggerStatusModeSelect(
    OC_MODULE_ID index,
    OC_TRIGGER_STATUS_MODES modes
);
```

Description

This function lets the user to reset the trigger status via software or a compare match.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module
modes	Identifies the trigger status reset mode

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsTriggerStatusModeSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
//Trigger status is reset by compare match
PLIB_OC_TriggerStatusModeSelect(MY_OC_ID, OC_TRIGGER_STATUS_MATCH);
```

5.6.13.5.3.8 PLIB_OC_AlternateClockDisable Function**C**

```
void PLIB_OC_AlternateClockDisable(
    OC_MODULE_ID index
);
```

Description

Selects Timer 2/3.instead of the alternate clock source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

None.

Remarks

The feature is not supported on all devices. Check your device's data sheet. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the [OC_MODULE_ID](#) passed in the call.

Example

```
// Call system service to unlock oscillator
#define MY_OC_ID OC_ID_1
PLIB_OC_AlternateClockDisable( MY_OC_ID );
```

5.6.13.5.3.9 PLIB_OC_AlternateClockEnable Function**C**

```
void PLIB_OC_AlternateClockEnable(
    OC_MODULE_ID index
);
```

Description

Selects the alternate clock source instead of Timer 2/3.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

None.

Remarks

The feature is not supported on all devices. Check your device's data sheet. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the [OC_MODULE_ID](#) passed in the call.

Example

```
// Call system service to unlock oscillator
#define MY_OC_ID OC_ID_1
PLIB_OC_AlternateClockEnable( MY_OC_ID );
```

5.6.13.5.4 Power-Saving Modes Functions**5.6.13.5.4.1 PLIB_OC_StopInIdleDisable Function****C**

```
void PLIB_OC_StopInIdleDisable(
```

```
    OC_MODULE_ID index
);
```

Description

The function continues OC module operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

None

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_StopInIdleDisable(MY_OC_ID);
```

5.6.13.5.4.2 PLIB_OC_StopInIdleEnable Function

C

```
void PLIB_OC_StopInIdleEnable(
    OC_MODULE_ID index
);
```

Description

This function discontinues OC module operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_StopInIdleEnable(MY_OC_ID);
```

5.6.13.5.5 Faults Functions

5.6.13.5.5.1 PLIB_OC_FaultHasOccurred Function

C

```
bool PLIB_OC_FaultHasOccurred(  
    OC_MODULE_ID index  
);
```

Description

This function returns 'true' if a PWM Fault has occurred and 'false' if no Fault condition exists.

Preconditions

This function should be used only in Edge or Center-Aligned PWM mode set by the [PLIB_OC_ModeSelect\(\)](#) function.

Parameters

Parameters	Description
index	Identifies the desired OC module

Returns

bool - true - PWM Fault has occurred

- false - No PWM fault

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsFaultStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1  
if(!PLIB_OC_FaultHasOccurred(MY_OC_ID))  
{  
    // Do some operation  
};
```

5.6.13.5.5.2 PLIB_OC_FaultInputSelect Function

C

```
void PLIB_OC_FaultInputSelect(  
    OC_MODULE_ID index,  
    OC_FAULTS flt  
);
```

Description

This function selects a Fault input for the OC module.

Preconditions

Set the operation mode using [PLIB_OC_ModeSelect\(\)](#) before selecting the Fault input.

Parameters

Parameters	Description
index	Identifies the desired OC module
flt	Identifies the OC fault input

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsFaultInput](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
// Fault B is enabled for Output Compare Module
PLIB_OC_FaultInputSelect(MY_OC_ID, OC_FAULT_B);
```

5.6.13.5.5.3 PLIB_OC_FaultModeSelect Function**C**

```
void PLIB_OC_FaultModeSelect(
    OC_MODULE_ID index,
    OC_FAULT_MODES modes
);
```

Description

This function selects a mode to clear the Fault mode. The Fault mode can be reset automatically or by software once the Fault source ceases to exist.

Preconditions

Set the operation mode using [PLIB_OC_ModeSelect\(\)](#) routine before selecting the fault input.

Parameters

Parameters	Description
index	Identifies the desired OC module
modes	Identifies the mode to reset Fault status

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsFaultModeSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1
// Fault mode is maintained even after fault ceases. Fault mode is reset in software.
PLIB_OC_FaultModeSelect(MY_OC_ID, OC_SOFTWARE_CLEAR);
```

5.6.13.5.5.4 PLIB_OC_FaultOutSelect Function**C**

```
void PLIB_OC_FaultOutSelect(
    OC_MODULE_ID index,
    OC_FAULT_OUT fltState
);
```

Description

This function sets OC output to a known state on occurrence of a Fault in Edge or Center-Aligned PWM modes.

Preconditions

Edge or Center-Aligned PWM mode with Faults enabled should be selected. OC module output tristate should be disabled on occurrence of a Fault via [PLIB_OC_TriStateDisable\(\)](#).

Parameters

Parameters	Description
index	Identifies the desired OC module
fltState	Identifies the high or low state to which the output is driven on a fault

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsFaultOutSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

//PWM output is set to High on occurrence of fault

PLIB_OC_FaultOutSelect(MY_OC_ID, OC_HIGH);
```

5.6.13.5.5.5 PLIB_OC_FaultTristateSelect Function**C**

```
void PLIB_OC_FaultTristateSelect(
    OC_MODULE_ID index,
    OC_FAULT_TRISTATE tristateSel
);
```

Description

This function is used to enable or disable tristating of OC module outputs when a Fault occurs.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired OC module
tristateSel	Enable/Disable Tristate

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OC_ExistsFaultTristateSelect](#) in your application to determine whether this feature is available.

Example

```
#define MY_OC_ID OC_ID_1

//Tristate OC module output

PLIB_OC_FaultTristateSelect(MY_OC_ID, OC_ENABLE_TRISTATE);
```

5.6.13.5.6 Data Types and Constants

5.6.13.5.6.1 OC_16BIT_TIMERS Enumeration

C

```
typedef enum {
    OC_TIMER_16BIT_TMR2,
    OC_TIMER_16BIT_TMR3,
    OC_TIMER_16BIT_TMR4,
    OC_TIMER_16BIT_TMR5,
    OC_TIMER_16BIT_TMR1,
    OC_TIMER_PTG,
    OC_TIMER_PERIPHERAL_CLK
} OC_16BIT_TIMERS;
```

Description

OC 16 bit Timer Select

This enumeration lists different 16 bit time bases for OC module. The time base is set by [PLIB_OC_TimerSelect\(\)](#) routine.

Members

Members	Description
OC_TIMER_16BIT_TMR2	Clock source of Timer2 is the clock source of OC module
OC_TIMER_16BIT_TMR3	Clock source of Timer3 is the clock source of OC module
OC_TIMER_16BIT_TMR4	Clock source of Timer4 is the clock source of OC module
OC_TIMER_16BIT_TMR5	Clock source of Timer5 is the clock source of OC module
OC_TIMER_16BIT_TMR1	Clock source of Timer1 is the clock source of OC module
OC_TIMER_PTG	Peripheral Trigger Generator output is the clock source for OC module
OC_TIMER_PERIPHERAL_CLK	Peripheral clock is the clock source for OC module

5.6.13.5.6.2 OC_BUFFER_SIZE Enumeration

C

```
typedef enum {
    OC_BUFFER_SIZE_16BIT,
    OC_BUFFER_SIZE_32BIT
} OC_BUFFER_SIZE;
```

Description

OC Buffer Size

This enumeration lists different buffer sizes for compare value. and duty cycle value.

Members

Members	Description
OC_BUFFER_SIZE_16BIT	Buffer size is 16-bits. Duty cycle and compare values will have 16-bit values.
OC_BUFFER_SIZE_32BIT	Buffer size is 32-bits. Duty cycle and compare values will have 32-bit values.

5.6.13.5.6.3 OC_CLOCK_RESOLUTION Enumeration

C

```
typedef enum {
    OC_CLOCK_P1,
    OC_CLOCK_P2,
    OC_CLOCK_P3,
    OC_CLOCK_P4
} OC_CLOCK_RESOLUTION;
```


Description

OC PWM Clock Resolution

Lists different PWM resolutions in PWM mode and falling edge delay of Output Compare module output in compare modes.

Members

Members	Description
OC_CLOCK_P1	OC output falling edge transitionson rising edge of P1 clock
OC_CLOCK_P2	OC output falling edge transitionson rising edge of P2 clock
OC_CLOCK_P3	OC output falling edge transitionson rising edge of P3 clock
OC_CLOCK_P4	OC output falling edge transitionson rising edge of P4 clock

5.6.13.5.6.4 OC_COMPARE_MODES Enumeration

C

```
typedef enum {
    OC_COMPARE_PWM_CENTER_ALIGNED_MODE,
    OC_COMPARE_PWM_EDGE_ALIGNED_MODE,
    OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE,
    OC_DUAL_COMPARE_SINGLE_PULSE_MODE,
    OC_TOGGLE_CONTINUOUS_PULSE_MODE,
    OC_SET_LOW_SINGLE_PULSE_MODE,
    OC_SET_HIGH_SINGLE_PULSE_MODE,
    OC_COMPARE_TURN_OFF_MODE
} OC_COMPARE_MODES;
```

Description

OC Compare Modes

This enumeration lists different compare modes for OC module. The compare mode is set by [PLIB_OC_ModeSelect\(\)](#) routine.

Members

Members	Description
OC_COMPARE_PWM_CENTER_ALIGNED_MODE	PWM center aligned mode: OC module output is a center aligned PWM signal
OC_COMPARE_PWM_EDGE_ALIGNED_MODE	PWM edge aligned mode: OC module output is an edge aligned PWM signal
OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE	Dual Compare, Continuous Pulse mode: OC module output is driven high on compare match with primary compare value and driven low on compare match with secondary compare value. A continuous stream of pulses is generated unless the compare mode is changed or the module is disabled. If the secondary compare value is greater than time base period value, secondary compare match does not occur. As a consequence, OC module output stays high.
OC_DUAL_COMPARE_SINGLE_PULSE_MODE	Dual Compare, Single Pulse mode: OC module output is driven high on compare match with primary compare value and driven low on compare match with secondary compare value. If the secondary compare value is greater than time base period value, secondary compare match does not occur. As a consequence, OC module output stays high until a mode change is made or module is disabled
OC_TOGGLE_CONTINUOUS_PULSE_MODE	Single Compare Toggle mode: OC module output is initialized to Low. OC module output toggles at every compare match event with primary compare value with a single peripheral bus clock cycle delay. This scheme generates a square wave with 50% duty cycle. An interrupt is generated each time the output toggles.

OC_SET_LOW_SINGLE_PULSE_MODE	Single Compare Set Low mode: A compare match event with primary compare value will set the output of OC module 'Low' with a single peripheral bus clock cycle delay. Output stays Low unless OC module is disabled or a new compare mode is chosen. An interrupt is generated at compare match event. OC module output is initially forced High.
OC_SET_HIGH_SINGLE_PULSE_MODE	Single Compare Set High mode: A compare match event with primary compare value will set the output of OC module 'High' with a single peripheral bus clock cycle delay. Output stays High unless OC module is disabled or a new compare mode is chosen. An interrupt is generated at compare match event. OC module output is initially forced Low.
OC_COMPARE_TURN_OFF_MODE	Turn OFF mode: OC module is disabled but still draws current. This mode is used to temporarily turn OFF the OC module before a new compare mode is selected

5.6.13.5.6.5 OC_FAULT_MODES Enumeration

C

```
typedef enum {
    OC_SOFTWARE_CLEAR,
    OC_AUTOMATIC_CLEAR
} OC_FAULT_MODES;
```

Description

OC Fault Clear Modes

This enumeration lists modes to clear fault mode after the fault source is removed.

Members

Members	Description
OC_SOFTWARE_CLEAR	Fault mode needs to be cleared in software after the fault source is removed.
OC_AUTOMATIC_CLEAR	Fault mode is cleared automatically once fault source is removed.

5.6.13.5.6.6 OC_FAULT_OUT Enumeration

C

```
typedef enum {
    OC_FAULT_LOW,
    OC_FAULT_HIGH
} OC_FAULT_OUT;
```

Description

OC Fault Output State

This enumeration lists different states of OC output pin when a fault occurs.

Members

Members	Description
OC_FAULT_LOW	OC output is driven low when fault occurs.
OC_FAULT_HIGH	OC output is driven high when fault occurs.

5.6.13.5.6.7 OC_FAULT_TRISTATE Enumeration

C

```
typedef enum {
    OC_DISABLE_TRISTATE,
    OC_ENABLE_TRISTATE
} OC_FAULT_TRISTATE;
```

Description

OC Fault Output Tristate select

This enumeration lists modes to clear fault mode after the fault source is removed.

Members

Members	Description
OC_DISABLE_TRISTATE	OC module outputs are set in a known state on the occurrence of a fault
OC_ENABLE_TRISTATE	OC module outputs are tristated on the occurrence of a fault

5.6.13.5.6.8 OC_FAULTS Enumeration

C

```
typedef enum {
    OC_FAULT_A,
    OC_FAULT_B,
    OC_FAULT_C,
    OC_FAULT_PRESET,
    OC_FAULT_DISABLE
} OC_FAULTS;
```

Description

OC Fault Select

This enumeration lists different fault sources for OC module. The fault source is selected by `PLIB_OC_CompareClockSelect()` routine.

Members

Members	Description
OC_FAULT_A	Fault A input
OC_FAULT_B	Fault B input
OC_FAULT_C	Fault C input
OC_FAULT_PRESET	Preset hardware fault input. This value is used for OC modules that have a preset hardware configured fault input selection.
OC_FAULT_DISABLE	Faults are disabled. PWM operation without faults.

5.6.13.5.6.9 OC_MODULE_ID Enumeration

C

```
typedef enum {
    OC_ID_1,
    OC_ID_2,
    OC_ID_3,
    OC_NUMBER_OF_MODULES
} OC_MODULE_ID;
```

Description

OC_MODULE_ID

This enumeration defines number of modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers.

Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.

Members

Members	Description
OC_NUMBER_OF_MODULES	The total number of modules available.

5.6.13.5.6.10 OC_SYNC_MODES Enumeration

C

```
typedef enum {
    OC_MODE_TRIGGER,
    OC_MODE_SYNC
} OC_SYNC_MODES;
```

Description

OC Synchronization modes Status

This enumeration lists different synchronization modes for OC module timer. The timer can be made to operate synchronously with an external input or asynchronously on receiving an input trigger.

Members

Members	Description
OC_MODE_TRIGGER	OC module is asynchronously triggered by a chosen trigger input source
OC_MODE_SYNC	OC module is synchronized to the input source

5.6.13.5.6.11 OC_SYNC_SOURCES Enumeration

C

```
typedef enum {
    OC_SYNC_SOURCE_NONE,
    OC_SYNC_SOURCE_OC1,
    OC_SYNC_SOURCE_OC2,
    OC_SYNC_SOURCE_OC3,
    OC_SYNC_SOURCE_OC4,
    OC_SYNC_SOURCE_OC5,
    OC_SYNC_SOURCE_OC6,
    OC_SYNC_SOURCE_OC7,
    OC_SYNC_SOURCE_OC8,
    OC_SYNC_SOURCE_OC9,
    OC_SYNC_SOURCE_PTG,
    OC_SYNC_SOURCE_TMR1,
    OC_SYNC_SOURCE_TMR2,
    OC_SYNC_SOURCE_TMR3,
    OC_SYNC_SOURCE_TMR4,
    OC_SYNC_SOURCE_TMR5,
    OC_SYNC_SOURCE_IC1,
    OC_SYNC_SOURCE_IC2,
    OC_SYNC_SOURCE_IC3,
    OC_SYNC_SOURCE_IC4,
    OC_SYNC_SOURCE_IC5,
    OC_SYNC_SOURCE_IC6,
    OC_SYNC_SOURCE_IC7,
```

```

OC_SYNC_SOURCE_IC8,
OC_SYNC_SOURCE_CMP1,
OC_SYNC_SOURCE_CMP2,
OC_SYNC_SOURCE_CMP3,
OC_SYNC_SOURCE_AD,
OC_SYNC_SOURCE_CTMU,
OC_SYNC_SOURCE_INT1,
OC_SYNC_SOURCE_INT2
} OC_SYNC_SOURCES;

```

Description

OC Synchronization Sources

This enumeration lists all the input sources for synchronous and asynchronously triggered operation. OC module timer can be synchronized to an input source or operate asynchronously on the receipt on an input trigger. The Input source can be selected by [PLIB_OC_SyncSourceSelect\(\)](#) routine.

Members

Members	Description
OC_SYNC_SOURCE_NONE	No synchronization and trigger input
OC_SYNC_SOURCE_OC1	Synchronization and trigger input is Output Compare1
OC_SYNC_SOURCE_OC2	Synchronization and trigger input is Output Compare2
OC_SYNC_SOURCE_OC3	Synchronization and trigger input is Output Compare3
OC_SYNC_SOURCE_OC4	Synchronization and trigger input is Output Compare4
OC_SYNC_SOURCE_OC5	Synchronization and trigger input is Output Compare5
OC_SYNC_SOURCE_OC6	Synchronization and trigger input is Output Compare6
OC_SYNC_SOURCE_OC7	Synchronization and trigger input is Output Compare7
OC_SYNC_SOURCE_OC8	Synchronization and trigger input is Output Compare8
OC_SYNC_SOURCE_OC9	Synchronization and trigger input is Output Compare9
OC_SYNC_SOURCE_PTG	Synchronization and trigger input is Peripheral Trigger Generator Module
OC_SYNC_SOURCE_TMR1	Synchronization and trigger input is Timer1
OC_SYNC_SOURCE_TMR2	Synchronization and trigger input is Timer2
OC_SYNC_SOURCE_TMR3	Synchronization and trigger input is Timer3
OC_SYNC_SOURCE_TMR4	Synchronization and trigger input is Timer4
OC_SYNC_SOURCE_TMR5	Synchronization and trigger input is Timer5
OC_SYNC_SOURCE_IC1	Synchronization and trigger input is Input Capture1
OC_SYNC_SOURCE_IC2	Synchronization and trigger input is Input Capture2
OC_SYNC_SOURCE_IC3	Synchronization and trigger input is Input Capture3
OC_SYNC_SOURCE_IC4	Synchronization and trigger input is Input Capture4
OC_SYNC_SOURCE_IC5	Synchronization and trigger input is Input Capture5
OC_SYNC_SOURCE_IC6	Synchronization and trigger input is Input Capture6
OC_SYNC_SOURCE_IC7	Synchronization and trigger input is Input Capture7
OC_SYNC_SOURCE_IC8	Synchronization and trigger input is Input Capture8
OC_SYNC_SOURCE_CMP1	Trigger input is Comparator1. This input is not used for Synchronization
OC_SYNC_SOURCE_CMP2	Trigger input is Comparator2. This input is not used for Synchronization
OC_SYNC_SOURCE_CMP3	Trigger input is Comparator3. This input is not used for Synchronization
OC_SYNC_SOURCE_AD	Trigger input is ADC. This input is not used for Synchronization
OC_SYNC_SOURCE_CTMU	Trigger input is Charge Time Measurement Unit. This input is not used for Synchronization
OC_SYNC_SOURCE_INT1	Synchronization and trigger input is External Interrupt1

OC_SYNC_SOURCE_INT2	Synchronization and trigger input is External Interrupt2

5.6.13.5.6.12 OC_TRIGGER_STATUS_MODES Enumeration

C

```
typedef enum {
    OC_TRIGGER_STATUS_MATCH,
    OC_TRIGGER_STATUS_SOFTWARE
} OC_TRIGGER_STATUS_MODES;
```

Description

OC Trigger Status

This enumeration lists different trigger status reset modes. Trigger status can be reset in software or by a compare match between timer value and pulse width value.

Members

Members	Description
OC_TRIGGER_STATUS_MATCH	Trigger status is cleared on compare match between timer value and pulse width value
OC_TRIGGER_STATUS_SOFTWARE	Trigger status is cleared in software

5.6.13.5.7 Feature Existence Functions

5.6.13.5.7.1 PLIB_OC_ExistsAlternateClock Function

C

```
bool PLIB_OC_ExistsAlternateClock(
    OC_MODULE_ID index
);
```

Description

This function identifies whether the AlternateClock feature is available on the OC module. When this function returns true, these functions are supported on the device:

- [PLIB_OC_AlternateClockEnable](#)
- [PLIB_OC_AlternateClockDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlternateClock feature is supported on the device
- false - The AlternateClock feature is not supported on the device

Remarks

None.

5.6.13.5.7.2 PLIB_OC_ExistsBufferSize Function

C

```
bool PLIB_OC_ExistsBufferSize(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the BufferSize feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_BufferSizeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the BufferSize feature is supported on the device
- false - If the BufferSize feature is not supported on the device

Remarks

None.

5.6.13.5.7.3 PLIB_OC_ExistsBufferValue Function

C

```
bool PLIB_OC_ExistsBufferValue(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the BufferValue feature is available on the OC module. When this function returns true, these functions are supported on the device:

- [PLIB_OC_Buffer32BitSet](#)
- [PLIB_OC_Buffer16BitSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the BufferValue feature is supported on the device
- false - If the BufferValue feature is not supported on the device

Remarks

None.

5.6.13.5.7.4 PLIB_OC_ExistsCompareModeSelect Function

C

```
bool PLIB_OC_ExistsCompareModeSelect(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the CompareModeSelect feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_ModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the CompareModeSelect feature is supported on the device
- false - If the CompareModeSelect feature is not supported on the device

Remarks

None.

5.6.13.5.7.5 PLIB_OC_ExistsEnableControl Function

C

```
bool PLIB_OC_ExistsEnableControl(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the OC module. When this function returns true, these functions are supported on the device:

- [PLIB_OC_Enable](#)
- [PLIB_OC_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the EnableControl feature is supported on the device
- false - If the EnableControl feature is not supported on the device

Remarks

None.

5.6.13.5.7.6 PLIB_OC_ExistsFaultInput Function

C

```
bool PLIB_OC_ExistsFaultInput(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the FaultInput feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_FaultInputSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the FaultInput feature is supported on the device
- false - If the FaultInput feature is not supported on the device

Remarks

None.

5.6.13.5.7.7 PLIB_OC_ExistsFaultModeSelect Function

C

```
bool PLIB_OC_ExistsFaultModeSelect(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the FaultModeSelect feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_FaultModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the FaultModeSelect feature is supported on the device
- false - If the FaultModeSelect feature is not supported on the device

Remarks

None.

5.6.13.5.7.8 PLIB_OC_ExistsFaultOutSelect Function

C

```
bool PLIB_OC_ExistsFaultOutSelect(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the FaultOut feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_FaultOutSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the FaultOut feature is supported on the device
- false - If the FaultOut feature is not supported on the device

Remarks

None.

5.6.13.5.7.9 PLIB_OC_ExistsFaultStatus Function

C

```
bool PLIB_OC_ExistsFaultStatus(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the FaultStatus feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_FaultHasOccurred](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the FaultStatus feature is supported on the device
- false - If the FaultStatus feature is not supported on the device

Remarks

None.

5.6.13.5.7.10 PLIB_OC_ExistsFaultTristateSelect Function

C

```
bool PLIB_OC_ExistsFaultTristateSelect(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the FaultTristateSelect feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_FaultTristateSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the FaultTristateSelect feature is supported on the device
- false - If the FaultTristateSelect feature is not supported on the device

Remarks

None.

5.6.13.5.7.11 PLIB_OC_ExistsPolarityInvert Function

C

```
bool PLIB_OC_ExistsPolarityInvert(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the PolarityInvert feature is available on the OC module. When this function returns true, these functions are supported on the device:

- [PLIB_OC_PolarityInvertedEnable](#)
- [PLIB_OC_PolarityInvertedDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PolarityInvert feature is supported on the device
- false - If the PolarityInvert feature is not supported on the device

Remarks

None.

5.6.13.5.7.12 PLIB_OC_ExistsPulseWidth Function

C

```
bool PLIB_OC_ExistsPulseWidth(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the PulseWidth feature is available on the OC module. When this function returns true, these functions are supported on the device:

- [PLIB_OC_PulseWidth32BitSet](#)
- [PLIB_OC_PulseWidth16BitSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PulseWidth feature is supported on the device
- false - If the PulseWidth feature is not supported on the device

Remarks

None.

5.6.13.5.7.13 PLIB_OC_ExistsPWMDutyCycleResolutionControl Function

C

```
bool PLIB_OC_ExistsPWMDutyCycleResolutionControl(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the PWMDutyCycleResolution feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_PWMDutyCycleResolutionSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PWMDutyCycleResolution feature is supported on the device
- false - If the PWMDutyCycleResolution feature is not supported on the device

Remarks

None.

5.6.13.5.7.14 PLIB_OC_ExistsStopInIdle Function

C

```
bool PLIB_OC_ExistsStopInIdle(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the OC module. When this function returns true, these functions are supported on the device:

- [PLIB_OC_StopInIdleEnable](#)
- [PLIB_OC_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the StopInIdle feature is supported on the device
- false - If the StopInIdle feature is not supported on the device

Remarks

None.

5.6.13.5.7.15 PLIB_OC_ExistsSyncModeSelect Function

C

```
bool PLIB_OC_ExistsSyncModeSelect(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the SyncModeSelect feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_SyncModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SyncModeSelect feature is supported on the device
- false - If the SyncModeSelect feature is not supported on the device

Remarks

None.

5.6.13.5.7.16 PLIB_OC_ExistsSyncSourceSelect Function

C

```
bool PLIB_OC_ExistsSyncSourceSelect(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the SyncSourceSelect feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_SyncSourceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SyncSourceSelect feature is supported on the device
- false - If the SyncSourceSelect feature is not supported on the device

Remarks

None.

5.6.13.5.7.17 PLIB_OC_ExistsTimerSelect Function

C

```
bool PLIB_OC_ExistsTimerSelect(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the TimerSelect feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_TimerSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TimerSelect feature is supported on the device
- false - If the TimerSelect feature is not supported on the device

Remarks

None.

5.6.13.5.7.18 PLIB_OC_ExistsTimerTriggered Function

C

```
bool PLIB_OC_ExistsTimerTriggered(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the TimerTriggered feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_IsTriggered](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TimerTriggered feature is supported on the device
- false - If the TimerTriggered feature is not supported on the device

Remarks

None.

5.6.13.5.7.19 PLIB_OC_ExistsTriggerControl Function

C

```
bool PLIB_OC_ExistsTriggerControl(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the TriggerControl feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_TriggerSet](#)
- [PLIB_OC_TriggerClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TriggerControl feature is supported on the device
- false - If the TriggerControl feature is not supported on the device

Remarks

None.

5.6.13.5.7.20 PLIB_OC_ExistsTriggerStatusModeSelect Function

C

```
bool PLIB_OC_ExistsTriggerStatusModeSelect(  
    OC_MODULE_ID index  
);
```

Description

This function identifies whether the TriggerStatusModeSelect feature is available on the OC module. When this function returns true, this function is supported on the device:

- [PLIB_OC_TriggerStatusModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the TriggerStatusModeSelect feature is supported on the device
- false - If the TriggerStatusModeSelect feature is not supported on the device

Remarks

None.

5.6.13.6 Files

Files

Name	Description
plib_oc.h	This file contains the interface definition for the Output Compare (OC) peripheral library.

Description

5.6.13.6.1 plib_oc.h

Output Compare Module Peripheral Library Interface Header

This library provides a low-level abstraction of the Output Compare module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

Functions

	Name	Description
≡	PLIB_OC_AlternateClockDisable	Selects Timer 2/3.instead of the alternate clock source.
≡	PLIB_OC_AlternateClockEnable	Selects the alternate clock source.
≡	PLIB_OC_Buffer16BitSet	Sets a 16-bit primary compare value for compare operations.

	PLIB_OC_Buffer32BitSet	Sets a 32-bit primary compare value for compare operations.
	PLIB_OC_BufferSizeSelect	Sets the buffer size and pulse width to 16-bits or 32-bits.
	PLIB_OC_Disable	Disable the OC module
	PLIB_OC_Enable	Enables the OC module.
	PLIB_OC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the OC module
	PLIB_OC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the OC module.
	PLIB_OC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the OC module.
	PLIB_OC_ExistsCompareModeSelect	Identifies whether the CompareModeSelect feature exists on the OC module.
	PLIB_OC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the OC module.
	PLIB_OC_ExistsFaultInput	Identifies whether the FaultInput feature exists on the OC module.
	PLIB_OC_ExistsFaultModeSelect	Identifies whether the FaultModeSelect feature exists on the OC module.
	PLIB_OC_ExistsFaultOutSelect	Identifies whether the FaultOut feature exists on the OC module.
	PLIB_OC_ExistsFaultStatus	Identifies whether the FaultStatus feature exists on the OC module.
	PLIB_OC_ExistsFaultTristateSelect	Identifies whether the FaultTristateSelect feature exists on the OC module.
	PLIB_OC_ExistsPolarityInvert	Identifies whether the PolarityInvert feature exists on the OC module.
	PLIB_OC_ExistsPulseWidth	Identifies whether the PulseWidth feature exists on the OC module.
	PLIB_OC_ExistsPWMDutyCycleResolutionControl	Identifies whether the PWMDutyCycleResolution feature exists on the OC module.
	PLIB_OC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the OC module.
	PLIB_OC_ExistsSyncModeSelect	Identifies whether the SyncModeSelect feature exists on the OC module.
	PLIB_OC_ExistsSyncSourceSelect	Identifies whether the SyncSourceSelect feature exists on the OC module.
	PLIB_OC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the OC module.
	PLIB_OC_ExistsTimerTriggered	Identifies whether the TimerTriggered feature exists on the OC module.
	PLIB_OC_ExistsTriggerControl	Identifies whether the TriggerControl feature exists on the OC module.
	PLIB_OC_ExistsTriggerStatusModeSelect	Identifies whether the TriggerStatusModeSelect feature exists on the OC module.
	PLIB_OC_FaultHasOccurred	Checks if a PWM fault has occurred.
	PLIB_OC_FaultInputSelect	Selects a Fault input for the OC module.
	PLIB_OC_FaultModeSelect	Selects an automatic or software-based mode to clear the Fault.
	PLIB_OC_FaultOutSelect	OC output pin is driven to a known state on a Fault.
	PLIB_OC_FaultTristateSelect	The OC module output can be tristated or driven to a known state when a Fault occurs.

	PLIB_OC_IsTriggered	Checks whether or not a timer source has been triggered.
	PLIB_OC_ModeSelect	Selects the compare mode for the OC module.
	PLIB_OC_PolarityInvertedDisable	Disables OC module output inversion polarity.
	PLIB_OC_PolarityInvertedEnable	Output compare polarity is inverted.
	PLIB_OC_PulseWidth16BitSet	Sets a 16-bit pulse width for OC module output.
	PLIB_OC_PulseWidth32BitSet	Sets a 32-bit pulse width for OC module output.
	PLIB_OC_PWMdutyCycleResolutionSet	Sets the PWM duty cycle resolution and delays the falling edge of OC output in compare modes.
	PLIB_OC_StopInIdleDisable	OC module continues operating when the device enters Idle mode
	PLIB_OC_StopInIdleEnable	Discontinues OC module operation when the device enters Idle mode
	PLIB_OC_SyncModeSelect	Selects the synchronization mode for the OC module.
	PLIB_OC_SyncSourceSelect	Selects the input source for synchronous or asynchronous (triggered) operation of the OC module.
	PLIB_OC_TimerSelect	Selects a clock source for the OC module.
	PLIB_OC_TriggerClear	Clears the trigger status for the OC module.
	PLIB_OC_TriggerSet	Sets the trigger status for the OC module.
	PLIB_OC_TriggerStatusModeSelect	Selects the trigger status reset mode.

File Name

plib_oc.h

Company

Microchip Technology Inc.

5.6.14 Oscillator Peripheral Library

5.6.14.1 Introduction

Oscillator Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Oscillator module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description**Oscillator Overview**

The Oscillator is the heart of the microcontroller and provides the clock on which the core and the peripherals run.

For all of the oscillators present on a Microchip microcontroller, two kinds of configurations exist:

- Through Configuration bits
- At execution time

The first is through 'Configuration bits', which is a one-time configuration done during the programming of the device. These one-time configurations are programmed in the code memory.

The other is 'Execution time' configuration, which deals with features that are allowed to be changed during run-time. This peripheral library provides functions which deal only with the 'execution time' configurable features of the Oscillator module.

Certain Oscillator module features can only be selected through 'Configuration bits'. However, most of the features can be altered during run-time by using the functions provided in this library.

5.6.14.2 Release Notes

MPLAB Harmony Version: v0.70b **Oscillator Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.14.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.14.4 Using the Library

Using the Library

This topic describes the basic architecture of the Oscillator Peripheral Library and provides information and examples on its use.

Interface Header File: plib_osc.h

The interface to the Oscillator Peripheral Library is defined in the "plib_osc.h" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the Oscillator Peripheral Library must include "peripheral.h".

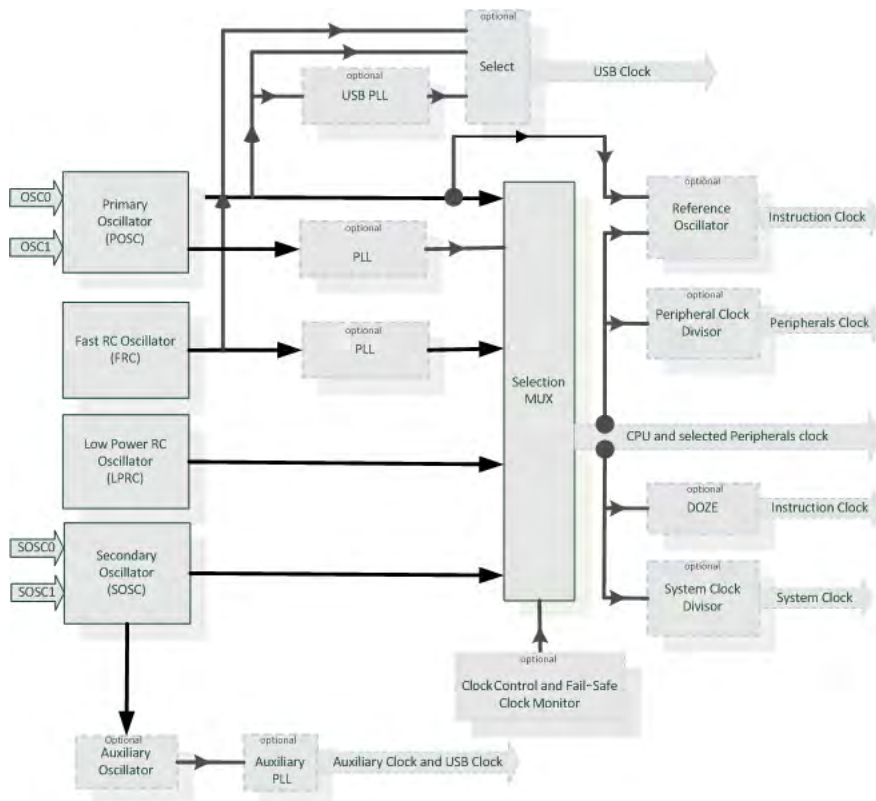
5.6.14.4.1 Hardware Abstraction Model

This library provides a low-level abstraction of the Oscillator module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model

The Oscillator hardware model is shown in the following figure. The blocks represented in dashed lines may not be present in all microcontrollers. Refer to the **"Oscillator"** chapter in the specific device data sheet to determine availability.



To understand the Oscillator module and how each feature is mapped in this library, it is important to understand the following terminology.

Clock Source

A clock source is hardware that generates oscillations, which may be internal or external.

Divisor and Multiplier/PLL

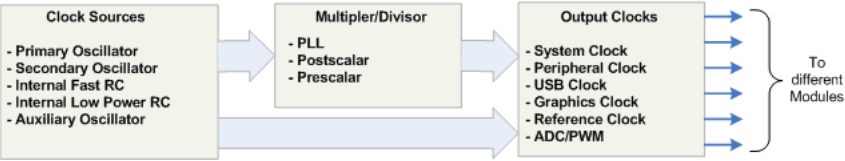
These are hardware modules that can scale the clock. The rate at which the scaling is done may be fixed or configurable.

Clock Outputs

Clock outputs means output lines from the Oscillator module, which may route to different modules of the device or to the CPU (i.e., the system clock).

The following diagram provides a simplified explanation and the relationship between the previously mentioned terms. In most cases, there are multiple clock source options available for each of the clock outputs. However, not all the clock sources are

available for all the output clocks. Scaling is an optional feature in most of the cases.

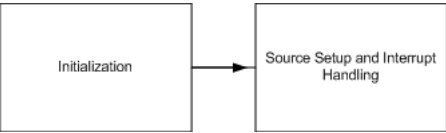


5.6.14.4.2 Library Usage Model

The usage model for the Oscillator Peripheral Library is described in this section.

Description

The following diagram shows the major components of the usage model:



Each of the blocks in the diagram above correspond to the library interface section.

Library Interface Section	Description

5.6.14.4.3 How the Library Works

The library can be used to control the Oscillator module. The usage model is explained in this section.

5.6.14.4.3.1 Oscillator Selection and Switching

Note: The device Configuration options change with the microcontroller family. Refer to the **"Oscillator"** chapter in the specific device data sheet for possible device configurations.

On devices that allow it, the oscillator selection can be overridden at run-time using the Oscillator Peripheral Library, as shown in the following code example:

```
//Do the configuration bit settings
OSC_SYS_TYPE currOsc;
OSC_SYS_TYPE newOsc=OSC_PRIMARY;

//get the current oscillator
currOsc = PLIB_OSC_CurrentSysClockGet(OSC_ID_0);

//check if the current oscillator is same as new oscillator
if(currOsc != newOsc)
{
    //set the new oscillator request to Fast RC oscillator
    PLIB_OSC_SysClockSelect ( OSC_ID_0, newOsc );
}
```

Example: Oscillator Selection Change During Run-time

In the event the clock switch did not complete, the clock switch logic can be reset by calling the following API:

```
bool oscSwitch_st;

oscSwitch_st = PLIB_OSC_ClockSwitchingIsComplete(OSC_ID_0);

if(!oscSwitch_st)
{
```

```
    PLIB_OSC_ClockSwitchingAbort(OSC_ID_0);
}
```

Note: This function takes care of the register unlock, which is required as per the device data sheet. Therefore, the user does not need to explicitly perform the unlock sequence.

Example: Oscillator Switch Abort

This will abandon the clock switch process, stop and reset the oscillator start-up timer (if applicable), and stop the PLL (if applicable). The clock switch process can be aborted at any time. A clock switch that is already in progress can also be aborted by performing a second clock switch.

5.6.14.4.3.2 Clock Sources

This section explains the available clock sources and their setup.

The following clock sources are available in Microchip microcontrollers:

- Primary Oscillator (Posc)
- Secondary Oscillator (Sosc)
- Internal Fast RC (FRC) Oscillator
- Internal Low-Power RC (LPRC) Oscillator

Primary Oscillator (Posc)

The Primary Oscillator has several operating modes (refer to the "**Oscillator**" chapter in the specific device data sheet for exact operating modes). The selection of the operating mode is done using the device Configuration registers during device programming. During run-time this can be changed using the oscillator switch option. However, changing the operating mode from High Gain (HC) to External Clock (EC) or External Resonator(XT) is not possible.

If a PLL is available, the PLL input divider can only be programmed during device programming. However, the PLL output divider and the PLL multiplier can be changed during run-time using the PLL functions described in the PLL section.

Secondary Oscillator (Sosc)

The optional Secondary Oscillator is designed specifically for low-power operation with an external crystal. The Secondary Oscillator is enabled in hardware by the device Configuration bits. Once it is enabled in hardware, it can be switched on during software run-time.

```
bool oscSecondary_st;

oscSecondary_st = PLIB_OSC_SecondaryIsEnabled(OSC_ID_0);
if(oscSecondary_st)
{
    PLIB_OSC_SecondaryEnable(OSC_ID_0);
}
//Do something
PLIB_OSC_SecondaryDisable(OSC_ID_0);
```

Internal Fast RC (FRC) Oscillator

The FRC Oscillator is a fast user-trimmable internal RC oscillator with a user-selectable input divider, PLL multiplier, and output divider. See the "**Oscillator**" chapter in the specific device data sheet for more information on the FRC oscillator.

Once the FRC oscillator is selected using the device Configuration registers or an oscillator switch is initiated selecting the FRC as a new oscillator, the FRC Oscillator PLL divider can be changed during run-time.

```
OSC_FRC_DIV FRCDivisor;

//Get the current divider value for FRC
```

```
FRCDivisor = PLIB_OSC_FRCDivisorGet(OSC_ID_0);

if(FRCDivisor != OSC_FRC_DIV_4)
{
    PLIB_OSC_FRCDivisorSelect ( OSC_ID_0, OSC_FRC_DIV_4 );
}
```

Internal Low-Power RC (LPRC) Oscillator

The LPRC Oscillator is separate from the FRC Oscillator. It oscillates at a nominal frequency of 31.25 kHz (this value is device-dependent). The LPRC Oscillator can act as the clock source for the Power-up Timer (PWRT), Watchdog Timer (WDT), Fail-Safe Clock Monitor (FSCM), and PLL reference circuits. It can also be used to provide a low-frequency clock source option for the device in those applications where power consumption is critical and timing accuracy is not required.

The LPRC remains enabled after power on in the following conditions:

- Fail-safe clock monitoring is enabled
- Watchdog Timer is enabled
- LPRC is selected as the system clock

PLL Configuration

```
OSC_SYSPLL_OUT_DIV PLLOutDiv;
OSC_SYSPLL_MULTIPLIER pll_multiply;

pll_multiply = PLIB_OSC_SysPLLMultiplierGet(OSC_ID_0);
if(pll_multiply != 15)
{
    PLIB_OSC_SysPLLMultiplierSelect(OSC_ID_0, 15);
}
PLLOutDiv = PLIB_OSC_SysPLLOutputDivisorGet(OSC_ID_0);

if(PLLOutDiv != OSC_SYSPLL_OUT_DIV_8)
{
    PLIB_OSC_SysPLLOutputDivisorSelect(OSC_ID_0, OSC_SYSPLL_OUT_DIV_8);
}
```

5.6.14.4.3.3 Fail-Safe Clock Monitor

Fail-Safe Clock Monitor (FSCM)

The Fail-Safe Clock Monitor (FSCM) is designed to allow continued device operation if the current oscillator fails. It is intended for use with the Primary Oscillator (Posc) and automatically switches to the FRC oscillator if a Posc failure is detected. The switch to the FRC oscillator allows continued device operation and the ability to retry the Posc or to execute the appropriate for a clock failure.

The FSCM can be enabled in the device configuration during the programming of the device. During run-time, the clock failure status can be obtained as follows:

```
bool clockFail;

//Returns true if clock failure is detected.
clockFail = PLIB_OSC_ClockHasFailed(OSC_ID_0);
```

5.6.14.4.3.4 Internal FRC Oscillator Tuning

Tuning the Oscillator

Oscillator tuning will help compensate for temperature effects on the FRC frequency over a wide range of temperatures. The tuning step size is an approximation, the application is supposed to try different values to achieve the best result. In some of the devices there are different tuning modes available.

Direct Number Method

FRC tuning is based on the number specified by the [PLIB_OSC_FRCTuningSelect](#) function.

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_USING_NUMBER);
```

```
PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x11);
```

Sequential Dithering

To get the sequential dithering working, the client needs to set eight sequencer values. Value 0 is set using the [PLIB_OSC_FRCTuningSelect](#) function. The other seven values (values 1 through 7) are set using the [PLIB_OSC_FRCTuningSequenceValueSet](#) function.

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_SEQ_DITHER);

PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x11);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_1,
    OSC_FRC_TUNE_MINUS_2_25_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_2,
    OSC_FRC_TUNE_MINUS_1_5_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_3,
    OSC_FRC_TUNE_MINUS_0_375_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_4,
    OSC_FRC_TUNE_PLUS_0_43_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_5,
    OSC_FRC_TUNE_PLUS_1_29_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_6,
    OSC_FRC_TUNE_PLUS_2_54_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_7,
    OSC_FRC_TUNE_MINUS_3_PERCENT);
```

```
//Configure PWM (period, pulse width and turn on the module)
```

Pseudo-Random Number

Select the tuning mode. Then, configure the PWM module and set the period and pulse width. The Oscillator system generates a 4-bit number based on a pseudo-random number generation algorithm. It then uses this value to tune the FRC oscillator. The module will generate different frequencies corresponding to the generated pseudo-random numbers every eighth PWM cycle.

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_PSEUDO_RANDOM);
```

```
//15 bit Linear Feedback Shift register value
PLIB_OSC_LinearFeedbackShiftRegSet(OSC_ID_0, 0x7FFF);
```

```
//Configure PWM (period, pulse width and turn on the module)
```

Note: This function takes care of the register unlock, which is required as per the device data sheet. Therefore, the user does not need to explicitly perform the unlock sequence.

5.6.14.5 Configuring the Library

The library is configured for the supported oscillator modules when the processor is chosen in the MPLAB IDE.

5.6.14.6 Library Interface

Data Types and Constants

	Name	Description
	<u>OSC_PB_CLOCK_DIV_TYPE</u>	Type of the oscillator PB Clock divisor value.
	<u>OSC_REF_DIVISOR_TYPE</u>	Reference oscillator divisor type.
	<u>OSC_REFERENCE_MAX_DIV</u>	Defines the reference clock output divisor maximum value.
	<u>OSC_SYSPLL_MULTIPLIER_TYPE</u>	Type of the oscillator system PLL multiplier value.
	<u>OSC_AUX_CLOCK_SOURCE</u>	Lists the possible Auxiliary PLL input divider values.
	<u>OSC_AUX_MODE</u>	Lists the possible Auxiliary Oscillator mode values.
	<u>OSC_AUXPLL_IN_DIV</u>	Lists the possible Auxiliary PLL input divider values.
	<u>OSC_AUXPLL_MULTIPLIER</u>	Lists the possible Auxiliary PLL multiplier
	<u>OSC_AUXPLL_OUT_DIV</u>	Lists the possible Auxiliary PLL output divider values.
	<u>OSC_DOZE_RATIO</u>	Lists the possible doze values.
	<u>OSC_FRC_DIV</u>	Lists the possible Fast RC (FRC) Oscillator divider values.
	<u>OSC_FRC_TUNE</u>	Lists the oscillator tuning values.
	<u>OSC_GFX_CLOCK</u>	Lists the possible Graphics (display) module clock values.
	<u>OSC_LFSR_TYPE</u>	Lists the possible Auxiliary PLL input divider values.
	<u>OSC_MODULE_ID</u>	Possible instances of the OSC module.
	<u>OSC_OPERATION_ON_WAIT</u>	Lists the possible status on a WAIT instruction.
	<u>OSC_PERIPHERAL_BUS</u>	Lists the possible Peripheral buses
	<u>OSC_PLL_SELECT</u>	Lists the PLLs in the Oscillator module.
	<u>OSC_PLLAUX_CLOCK_SOURCE</u>	Lists the possible Auxiliary PLL clock source.
	<u>OSC_REF_BASECLOCK</u>	Lists the possible base clock values for the reference oscillator.
	<u>OSC_REF_DIV</u>	Lists the possible reference oscillator divisor values.
	<u>OSC_REFERENCE</u>	Lists the possible reference oscillator
	<u>OSC_SYS_CLOCK_DIV</u>	Lists the possible system clock divisor bits.
	<u>OSC_SYS_TYPE</u>	Lists the possible oscillator type values.
	<u>OSC_SYSPLL_FREQ_RANGE</u>	Lists the possible PLL frequency range
	<u>OSC_SYSPLL_IN_CLK_SOURCE</u>	Lists the possible input clock source for PLL module
	<u>OSC_SYSPLL_IN_DIV</u>	Lists the possible PLL input divider values.
	<u>OSC_SYSPLL_OUT_DIV</u>	Lists the possible PLL output divider values.
	<u>OSC_TUNING_MODE</u>	Lists the oscillator tuning modes.
	<u>OSC_TUNING_SEQUENCERS</u>	Lists the oscillator sequencer entries.
	<u>OSC_USBCLOCK_SOURCE</u>	Lists the possible USB clock sources.

Auxiliary Oscillator Setup Functions

	Name	Description
	<u>PLIB_OSC_AuxClockSourceGet</u>	Gets the clock source for the auxiliary clock.
	<u>PLIB_OSC_AuxClockSourceSet</u>	Sets the clock source for the auxiliary clock divisor.
	<u>PLIB_OSC_AuxModeGet</u>	Gets the selected Auxiliary Oscillator mode.
	<u>PLIB_OSC_AuxModeSelect</u>	Selects the Auxiliary Oscillator mode.

Clock Fail Monitoring Functions

	Name	Description
	PLIB_OSC_ClockHasFailed	Returns 'true' if the clock fails.

Doze Setup Functions

	Name	Description
	PLIB_OSC_DozeModeDisable	Disables Doze mode.
	PLIB_OSC_DozeModeEnable	Enables Doze mode.
	PLIB_OSC_DozeRatioGet	Gets the ratio of the processor clock to the peripheral clock.
	PLIB_OSC_DozeRatioSelect	Selects the Doze ratio of the processor clock to the peripheral clock.
	PLIB_OSC_DozeRecoverOnInterruptDisable	Disables the recovery from Doze mode upon an interrupt.
	PLIB_OSC_DozeRecoverOnInterruptEnable	Enables recovery from Doze mode upon an interrupt.
	PLIB_OSC_DozeRecoverOnInterruptIsEnabled	Returns 'true' if recover from Doze mode on interrupt is enabled.

Fast RC Oscillator Setup Functions

	Name	Description
	PLIB_OSC_FRCDitherEnable	Enables the FRC Oscillator clock dithering.
	PLIB_OSC_FRCDivisorGet	Gets the FRC clock divisor.
	PLIB_OSC_FRCDivisorSelect	Sets the FRC clock divisor to the specified value.
	PLIB_OSC_FRCTuningModeGet	Gets the FRC oscillator tuning mode.
	PLIB_OSC_FRCTuningModeSet	Selects the FRC oscillator tuning mode.
	PLIB_OSC_FRCTuningSelect	Sets the FRC tuning value.
	PLIB_OSC_FRCTuningSequenceValueGet	Gets the value set in the FRC tune sequencer.
	PLIB_OSC_FRCTuningSequenceValueSet	Sets the value in the FRC tune sequencer.
	PLIB_OSC_LinearFeedbackShiftRegGet	Gets the value from linear feedback shift register.
	PLIB_OSC_LinearFeedbackShiftRegSet	Sets the linear feedback shift register.

Feature Existence Functions

	Name	Description
	PLIB_OSC_ExistsClockFail	Identifies whether the ClockFail feature exists on the OSC module
	PLIB_OSC_ExistsFRCDivisor	Identifies whether the FRCDivisor feature exists on the OSC module
	PLIB_OSC_ExistsFRCTuning	Identifies whether the FRCTuning feature exists on the OSC module
	PLIB_OSC_ExistsOnWaitAction	Identifies whether the OnWaitAction feature exists on the OSC module
	PLIB_OSC_ExistsOscCurrentGet	Identifies whether the OscCurrentGet feature exists on the OSC module
	PLIB_OSC_ExistsOscSelect	Identifies whether the OscSelect feature exists on the OSC module
	PLIB_OSC_ExistsOscSwitchInit	Identifies whether the OscSwitchInit feature exists on the OSC module
	PLIB_OSC_ExistsPBClockDivisor	Identifies whether the PBClockDivisor feature exists on the OSC module
	PLIB_OSC_ExistsPBClockOutputEnable	Identifies whether the PBClockOutputEnable feature exists on the OSC module

	PLIB_OSC_ExistsPBClockReady	Identifies whether the PBClockReady feature exists on the OSC module
	PLIB_OSC_ExistsPLLClockLock	Identifies whether the PLLClockLock feature exists on the OSC module
	PLIB_OSC_ExistsPLLLockStatus	Identifies whether the PLLLockStatus feature exists on the OSC module
	PLIB_OSC_ExistsReferenceOscBaseClock	Identifies whether the ReferenceOscBaseClock feature exists on the OSC module
	PLIB_OSC_ExistsReferenceOscChange	Identifies whether the ReferenceOscChange feature exists on the OSC module
	PLIB_OSC_ExistsReferenceOscChangeActive	Identifies whether the ReferenceOscChangeActive feature exists on the OSC module
	PLIB_OSC_ExistsReferenceOscDivisor	Identifies whether the ReferenceOscDivisor feature exists on the OSC module
	PLIB_OSC_ExistsReferenceOscEnable	Identifies whether the ReferenceOscEnable feature exists on the OSC module
	PLIB_OSC_ExistsReferenceOscStopInIdleEnable	Identifies whether the ReferenceOscStopInIdleEnable feature exists on the OSC module
	PLIB_OSC_ExistsReferenceOscStopInSleep	Identifies whether the ReferenceOscStopInSleep feature exists on the OSC module
	PLIB_OSC_ExistsReferenceOscTrim	Identifies whether the ReferenceOscTrim feature exists on the OSC module
	PLIB_OSC_ExistsReferenceOutputEnable	Identifies whether the ReferenceOutputEnable feature exists on the OSC module
	PLIB_OSC_ExistsSecondaryEnable	Identifies whether the SecondaryEnable feature exists on the OSC module
	PLIB_OSC_ExistsSecondaryReady	Identifies whether the SecondaryReady feature exists on the OSC module
	PLIB_OSC_ExistsSysPLLFrequencyRange	Identifies whether the PLLFrequencyRange feature exists on the OSC module
	PLIB_OSC_ExistsSysPLLInputClockSource	Identifies whether the PLLInputClockSource feature exists on the OSC module
	PLIB_OSC_ExistsSysPLLInputDivisor	Identifies whether the PLLInputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsSysPLLMultiplier	Identifies whether the PLLMultiplier feature exists on the OSC module
	PLIB_OSC_ExistsSysPLLOutputDivisor	Identifies whether the PLLOutputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsUsbClockSource	Identifies whether the UsbClockSource feature exists on the OSC module

General Setup Functions

	Name	Description
	PLIB_OSC_OnWaitActionGet	Gets the configured operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_OnWaitActionSet	Selects the operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_StartupTimerHasExpired	Returns 'true' if the oscillator start-up time-out timer has expired.

Oscillator Switch Setup Functions

	Name	Description
	PLIB_OSC_ClockSwitchingAbort	Aborts an oscillator switch.

	PLIB_OSC_ClockSwitchingIsComplete	Gets the oscillator switch progress status.
	PLIB_OSC_CurrentSysClockGet	Gets the current oscillator selected.
	PLIB_OSC_SysClockSelect	Selects the new oscillator.

Peripheral Clock Setup Functions

	Name	Description
	PLIB_OSC_PBClockDivisorGet	Gets the peripheral bus clock divisor.
	PLIB_OSC_PBClockDivisorIsReady	Checks whether the peripheral bus clock divisor is ready to be written.
	PLIB_OSC_PBClockDivisorSet	Sets the peripheral bus clock divisor to the specified value.
	PLIB_OSC_PBOutputClockDisable	Disables the peripheral bus output clock
	PLIB_OSC_PBOutputClockEnable	Enables the peripheral bus output clock
	PLIB_OSC_PBOutputClockIsEnabled	Checks whether the peripheral bus clock output is enabled or not

PLL Setup Functions

	Name	Description
	PLIB_OSC_PLLAuxClockSourceGet	Returns the clock source selected for the Auxiliary PLL.
	PLIB_OSC_PLLAuxClockSourceSelect	Sets the Auxiliary PLL clock source.
	PLIB_OSC_PLLAuxInputDivisorGet	Gets the input divisor for the Auxiliary PLL.
	PLIB_OSC_PLLAuxInputDivisorSelect	Sets the input divisor for the Auxiliary PLL to the specified value.
	PLIB_OSC_PLLAuxIsLocked	Returns whether the Auxiliary PLL is locked.
	PLIB_OSC_PLLAuxMultiplierGet	Gets the Auxiliary PLL multiplier.
	PLIB_OSC_PLLAuxMultiplierSelect	Sets the Auxiliary PLL multiplier to the specified value.
	PLIB_OSC_PLLAuxOutputDivisorGet	Gets the output divisor for the Auxiliary PLL.
	PLIB_OSC_PLLAuxOutputDivisorSet	Sets the output divisor for the Auxiliary PLL to the specified value.
	PLIB_OSC_PLLClockIsLocked	Gets the lock status for the clock and PLL selections.
	PLIB_OSC_PLLClockLock	Locks the clock and PLL selections.
	PLIB_OSC_PLLClockUnlock	Unlocks the clock and PLL selections.
	PLIB_OSC_PLLEnable	Disables the specified PLL.
	PLIB_OSC_PLLIsEnabled	Enables the specified PLL.
	PLIB_OSC_PLLIsLocked	Returns 'true' if the specified PLL is enabled.
	PLIB_OSC_PLLIsLocked	Returns 'true' if the selected PLL is locked.
	PLIB_OSC_SysPLLFrequencyRangeGet	Returns the frequency range for PLL
	PLIB_OSC_SysPLLFrequencyRangeSet	Sets the frequency range for PLL
	PLIB_OSC_SysPLLInputClockSourceGet	Returns the input clock source for the PLL module
	PLIB_OSC_SysPLLInputClockSourceSet	Sets the input clock source for the PLL module
	PLIB_OSC_SysPLLInputDivisorGet	Gets the input divisor for the PLL.
	PLIB_OSC_SysPLLInputDivisorSet	Sets the input divider for the PLL to the specified value.
	PLIB_OSC_SysPLLMultiplierGet	Gets the PLL multiplier.
	PLIB_OSC_SysPLLMultiplierSelect	Sets the PLL multiplier to the specified value.
	PLIB_OSC_SysPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_SysPLLOutputDivisorSet	Sets the output divider for the PLL to the specified value.

Primary Oscillator Setup Functions

	Name	Description
	PLIB_OSC_PrimaryOscInSleepModeDisable	The Primary Oscillator is disabled during Sleep mode.
	PLIB_OSC_PrimaryOscInSleepModeEnable	The Primary Oscillator continues to operate during Sleep mode.

	<u>PLIB_OSC_PrimaryOscInSleepModelsEnabled</u>	Returns 'true' if the Primary Oscillator is disabled during Sleep mode.

Reference Oscillator Setup Functions

	Name	Description
	<u>PLIB_OSC_ReferenceOscBaseClockSelect</u>	Sets the base clock for the reference oscillator.
	<u>PLIB_OSC_ReferenceOscDisable</u>	Disables the reference oscillator output.
	<u>PLIB_OSC_ReferenceOscDivisorValueSet</u>	Selects the reference oscillator divisor value.
	<u>PLIB_OSC_ReferenceOscEnable</u>	Enables the reference oscillator.
	<u>PLIB_OSC_ReferenceOscIsEnabled</u>	Gets the enable status of the reference oscillator output.
	<u>PLIB_OSC_ReferenceOscSourceChangeIsActive</u>	Returns 'true' if a reference oscillator source change request is active.
	<u>PLIB_OSC_ReferenceOscStopInIdleDisable</u>	Enables the reference oscillator in Idle mode.
	<u>PLIB_OSC_ReferenceOscStopInIdleEnable</u>	Configures the reference oscillator to stop operating in Idle mode.
	<u>PLIB_OSC_ReferenceOscStopInIdleIsEnabled</u>	Returns 'true' if the reference oscillator is disabled in Idle mode.
	<u>PLIB_OSC_ReferenceOscStopInSleepDisable</u>	Enables the reference oscillator in Sleep mode.
	<u>PLIB_OSC_ReferenceOscStopInSleepEnable</u>	Configures the reference oscillator to stop operating in Sleep mode.
	<u>PLIB_OSC_ReferenceOscStopInSleepIsEnabled</u>	Returns 'true' if the reference oscillator is disabled in Sleep mode.
	<u>PLIB_OSC_ReferenceOscSwitchIsComplete</u>	Returns 'true' if the reference oscillator base clock switching is complete.
	<u>PLIB_OSC_ReferenceOscTrimSet</u>	Sets the reference oscillator divisor trim value.
	<u>PLIB_OSC_ReferenceOutputDisable</u>	Disables the reference oscillator output.
	<u>PLIB_OSC_ReferenceOutputEnable</u>	Enables the reference oscillator output.
	<u>PLIB_OSC_ReferenceOutputIsEnabled</u>	Returns 'true' if the reference oscillator output is enabled.

Secondary Oscillator Setup Functions

	Name	Description
	<u>PLIB_OSC_SecondaryDisable</u>	Disables the Secondary Oscillator.
	<u>PLIB_OSC_SecondaryEnable</u>	Enables the Secondary Oscillator.
	<u>PLIB_OSC_SecondaryIsEnabled</u>	Returns 'true' if the Secondary Oscillator is enabled.
	<u>PLIB_OSC_SecondaryIsReady</u>	Returns 'true' if the Secondary Oscillator is ready.

USB and Display Clock Setup Functions

	Name	Description
	<u>PLIB_OSC_GraphicsClockDivisorGet</u>	Gets the Graphics module clock divisor.
	<u>PLIB_OSC_GraphicsClockDivisorSet</u>	Sets the Graphics module clock divisor.
	<u>PLIB_OSC_GraphicsClockSourceGet</u>	Gets the Graphics Controller module clock frequency value.
	<u>PLIB_OSC_GraphicsClockSourceSelect</u>	Sets the Graphics Controller module clock.
	<u>PLIB_OSC_UsbClockSourceGet</u>	Gets the USB module clock source.
	<u>PLIB_OSC_UsbClockSourceSelect</u>	Sets the USB module clock source.

Description

This section lists and describes the functions, data types, and constants provided in the Oscillator Peripheral Library.

5.6.14.6.1 General Setup Functions

5.6.14.6.1.1 PLIB_OSC_OnWaitActionGet Function

C

```
OSC_OPERATION_ON_WAIT PLIB_OSC_OnWaitActionGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the configured operation that is to be performed when a WAIT instruction is executed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

On a WAIT action, one of the possible values of [OSC_OPERATION_ON_WAIT](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsOnWaitAction](#) in your application to determine whether this feature is available.

Example

```
if (PLIB_OSC_OnWaitActionGet(OSC_ID_0) == OSC_ON_WAIT_SLEEP)  
{  
    //Do some action  
}
```

5.6.14.6.1.2 PLIB_OSC_OnWaitActionSet Function

C

```
void PLIB_OSC_OnWaitActionSet(  
    OSC_MODULE_ID index,  
    OSC_OPERATION_ON_WAIT onWait  
);
```

Description

This function selects the operation to be performed when a WAIT instruction is executed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
onWait	Operation to be performed when a WAIT instruction is executed. One of the possible values of OSC_OPERATION_ON_WAIT .

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_OSC_ExistsOnWaitAction](#) in your application to determine whether this feature is available.

If this API is not called, the device will enter Idle mode on execution of a WAIT instruction.

Example

```
PLIB_OSC_OnWaitActionSet(OSC_ID_0, OSC_ON_WAIT_SLEEP);
```

5.6.14.6.1.3 PLIB_OSC_StartupTimerHasExpired Function

C

```
bool PLIB_OSC_StartupTimerHasExpired(  
    OSC_MODULE_ID index  
);
```

Description

This function returns 'true' if the oscillator start-up time-out timer has expired.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

- true - Oscillator start-up time-out timer has expired
- false - Oscillator start-up time-out timer is running

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
bool startUpTimer;  
  
startUpTimer = PLIB_OSC_StartupTimerHasExpired(OSC_ID_0);
```

5.6.14.6.2 Primary Oscillator Setup Functions

5.6.14.6.2.1 PLIB_OSC_PrimaryOscInSleepModeDisable Function

C

```
void PLIB_OSC_PrimaryOscInSleepModeDisable(  
    OSC_MODULE_ID index  
);
```

Description

This function disables the Primary Oscillator during Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_PrimaryOscInSleepModeDisable(OSC_ID_0);
```

5.6.14.6.2.2 PLIB_OSC_PrimaryOscInSleepModeEnable Function**C**

```
void PLIB_OSC_PrimaryOscInSleepModeEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function enables the Primary Oscillator during Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_PrimaryOscInSleepModeEnable(OSC_ID_0);
```

5.6.14.6.2.3 PLIB_OSC_PrimaryOscInSleepModeIsEnabled Function**C**

```
bool PLIB_OSC_PrimaryOscInSleepModeIsEnabled(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the enable status of the Primary Oscillator during Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

- true - Primary Oscillator is enabled during Sleep mode

- false - Primary Oscillator is disabled during Sleep mode

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
bool priOscStatus;  
  
priOscStatus = PLIB_OSC_PrimaryOscInSleepModeIsEnabled(OSC_ID_0);
```

5.6.14.6.3 Secondary Oscillator Setup Functions

5.6.14.6.3.1 PLIB_OSC_SecondaryDisable Function

C

```
void PLIB_OSC_SecondaryDisable(  
    OSC_MODULE_ID index  
);
```

Description

This function disables the Secondary Oscillator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSecondaryEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_SecondaryDisable(OSC_ID_0);
```

5.6.14.6.3.2 PLIB_OSC_SecondaryEnable Function

C

```
void PLIB_OSC_SecondaryEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function enables the Secondary Oscillator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSecondaryEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_SecondaryEnable(OSC_ID_0);
```

5.6.14.6.3.3 PLIB_OSC_SecondaryIsEnabled Function**C**

```
bool PLIB_OSC_SecondaryIsEnabled(  
    OSC_MODULE_ID index  
);
```

Description

This function returns 'true' if the Secondary Oscillator is enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

- true - Secondary Oscillator is enabled
- false - Secondary Oscillator is disabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSecondaryEnable](#) in your application to determine whether this feature is available.

Example

```
bool secOscEnable;  
  
secOscEnable = PLIB_OSC_SecondaryIsEnabled(OSC_ID_0);
```

5.6.14.6.3.4 PLIB_OSC_SecondaryIsReady Function**C**

```
bool PLIB_OSC_SecondaryIsReady(  
    OSC_MODULE_ID index  
);
```

Description

This function returns the ready status of the Secondary Oscillator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

- true - Indicates that the Secondary Oscillator is running and is stable
- false - Secondary Oscillator is either turned off or is still warming up

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSecondaryReady](#) in your application to determine whether this feature is available.

Example

```
bool secOscReady;  
  
secOscReady = PLIB_OSC_SecondaryIsReady(OSC_ID_0);
```

5.6.14.6.4 Auxiliary Oscillator Setup Functions

5.6.14.6.4.1 PLIB_OSC_AuxClockSourceGet Function

C

```
OSC_AUX_CLOCK_SOURCE PLIB_OSC_AuxClockSourceGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the clock source for the auxiliary clock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values from [OSC_AUX_CLOCK_SOURCE](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
OSC_AUX_CLOCK_SOURCE auxClockSource;  
  
auxClockSource = PLIB_OSC_AuxClockSourceGet(OSC_ID_0);
```

5.6.14.6.4.2 PLIB_OSC_AuxClockSourceSet Function

C

```
void PLIB_OSC_AuxClockSourceSet(  
    OSC_MODULE_ID index,  
    OSC_AUX_CLOCK_SOURCE auxClockSource  
);
```

Description

This function sets the clock source for the auxiliary clock divisor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
auxClockSource	Clock source for the auxiliary clock divisor. One of the possible values from OSC_AUX_CLOCK_SOURCE .

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_AuxClockSourceSet( OSC_ID_0, OSC_AUX_CLOCK_PRIMARY );
```

5.6.14.6.4.3 PLIB_OSC_AuxModeGet Function**C**

```
OSC_AUX_MODE PLIB_OSC_AuxModeGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the Auxiliary Oscillator mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values of [OSC_AUX_MODE](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
OSC_AUX_MODE auxOscMode;  
  
auxOscMode = PLIB_OSC_AuxModeGet( OSC_ID_0 );
```

5.6.14.6.4.4 PLIB_OSC_AuxModeSelect Function**C**

```
void PLIB_OSC_AuxModeSelect(  
    OSC_MODULE_ID index,  
    OSC_AUX_MODE AuxOscMode
```

```
);
```

Description

This function selects the Auxiliary Oscillator mode based on the input.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
auxOscMode	One of the possible values of OSC_AUX_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_AuxModeSelect(OSC_ID_0, OSC_AUX_MODE_HS);
```

5.6.14.6.5 Reference Oscillator Setup Functions

5.6.14.6.5.1 PLIB_OSC_ReferenceOscBaseClockSelect Function

C

```
void PLIB_OSC_ReferenceOscBaseClockSelect(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc,  
    OSC_REF_BASECLOCK refOscBaseClock  
);
```

Description

This function sets the base clock for the reference oscillator. There are multiple clock sources by which the user can configure the module to output to the pin. Users can check the accuracy of the clock by probing the pin. Use the [PLIB_OSC_ReferenceOscDivisorValueSet](#) function to divide the clock if it is a very high value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator
refOscBaseClk	One of the possible values of OSC_REF_BASECLOCK

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscBaseClock](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_ReferenceOscBaseClockSelect(OSC_ID_0, OSC_REFERENCE_1, OSC_REF_BASECLOCK_PBCLK);
```

5.6.14.6.5.2 PLIB_OSC_ReferenceOscDisable Function**C**

```
void PLIB_OSC_ReferenceOscDisable(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc  
);
```

Description

This function disables output from the reference oscillator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_ReferenceOscDisable(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.3 PLIB_OSC_ReferenceOscDivisorValueSet Function**C**

```
void PLIB_OSC_ReferenceOscDivisorValueSet(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc,  
    OSC_REF_DIVISOR_TYPE refOscValue  
);
```

Description

This function selects the reference oscillator divisor value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator
refOscDiv	One of the possible values of OSC_REF_DIV

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscDivisor](#) in your application to determine whether this feature is available.

The value entered may not be the actual divisor. Please refer to the specific device data sheet to determine the actual divisor corresponding to the value entered.

Example

```
// Select the clock source.
PLIB_OSC_ReferenceOscBaseClockSelect(OSC_ID_0, OSC_REFERENCE_1, OSC_REF_BASECLOCK_PBCLK);

PLIB_OSC_ReferenceOscDivisorValueSet(OSC_ID_0, OSC_REFERENCE_1, 128);

PLIB_OSC_ReferenceOscEnable(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.4 PLIB_OSC_ReferenceOscEnable Function

C

```
void PLIB_OSC_ReferenceOscEnable(
    OSC_MODULE_ID index,
    OSC_REFERENCE referenceOsc
);
```

Description

This function enables the reference oscillator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscEnable](#) in your application to determine whether this feature is available.

If the device has a reference clock output enable control, calling this function may not give the reference clock output. Use the [PLIB_OSC_ReferenceOutputEnable](#) function to enable the output.

Example

```
PLIB_OSC_ReferenceOscEnable(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.5 PLIB_OSC_ReferenceOscIsEnabled Function

C

```
bool PLIB_OSC_ReferenceOscIsEnabled(
    OSC_MODULE_ID index,
    OSC_REFERENCE referenceOsc
);
```

```
);
```

Description

This function gets the enable status of the reference oscillator output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

- true - Reference oscillator is enabled
- false - Reference oscillator is disabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscEnable](#) in your application to determine whether this feature is available.

Example

```
if(PLIB_OSC_ReferenceOscIsEnabled(OSC_ID_0, OSC_REFERENCE_1))
{
    //Do some action
}
```

5.6.14.6.5.6 PLIB_OSC_ReferenceOscSourceChangeIsActive Function**C**

```
bool PLIB_OSC_ReferenceOscSourceChangeIsActive(
    OSC_MODULE_ID index,
    OSC_REFERENCE referenceOsc
);
```

Description

This function returns 'true' if the reference oscillator source change is in progress. The software is not allowed to give a new source change request.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

- true - Reference oscillator change request is active
- false - Reference oscillator change request is not active

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscChangeActive](#) in your application to determine whether this feature is available.

Example

```
if (!PLIB_OSC_ReferenceOscSourceChangeIsActive(OSC_ID_0, OSC_REFERENCE_1))
{
    //Allowed to change the reference clock source
}
```

5.6.14.6.5.7 PLIB_OSC_ReferenceOscStopInIdleDisable Function**C**

```
void PLIB_OSC_ReferenceOscStopInIdleDisable(
    OSC_MODULE_ID index,
    OSC_REFERENCE referenceOsc
);
```

Description

This function enables the reference oscillator in Idle mode. The reference oscillator continues to run in Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscStopInIdleEnable](#) in your application to determine whether this feature is available.

The default state of the device is the reference oscillator is enabled in Idle mode.

Example

```
PLIB_OSC_ReferenceOscStopInSleepDisable(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.8 PLIB_OSC_ReferenceOscStopInIdleEnable Function**C**

```
void PLIB_OSC_ReferenceOscStopInIdleEnable(
    OSC_MODULE_ID index,
    OSC_REFERENCE referenceOsc
);
```

Description

This function configures the reference oscillator to stop operating in Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscStopInIdleEnable](#) in your application to determine whether this feature is available.

The default state of the device is reference oscillator enabled in Idle mode. Therefore, calling this function is necessary only if the [PLIB_OSC_ReferenceOscStopInIdleDisable](#) function was previously called, and the software wants to enable oscillator operation in Sleep mode.

Example

```
PLIB_OSC_ReferenceOscStopInIdleEnable(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.9 PLIB_OSC_ReferenceOscStopInIdleIsEnabled Function**C**

```
bool PLIB_OSC_ReferenceOscStopInIdleIsEnabled(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc  
);
```

Description

This function returns 'true' if the reference oscillator is disabled in Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

- true - Reference oscillator is disabled in Idle mode
- false - Reference oscillator is enabled in Idle mode

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscStopInIdleEnable](#) in your application to determine whether this feature is available.

Example

```
bool refOscIdle;  
  
refOscIdle = PLIB_OSC_ReferenceOscStopInIdleIsEnabled(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.10 PLIB_OSC_ReferenceOscStopInSleepDisable Function**C**

```
void PLIB_OSC_ReferenceOscStopInSleepDisable(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc  
);
```

Description

This function enables the reference oscillator in Sleep mode. The reference oscillator continues to run in Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscStopInSleep](#) in your application to determine whether this feature is available.

The reference clock output will be stopped in Sleep mode if the base clock selected is 'System Clock' or 'Peripheral Clock' regardless of this function.

The default state of the device is for the reference oscillator to be enabled in Sleep mode. Therefore, calling this function is necessary only if the [PLIB_OSC_ReferenceOscStopInSleepEnable](#) function was previously called, and the software wants to enable oscillator operation in Sleep mode.

Example

```
PLIB_OSC_ReferenceOscStopInSleepDisable(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.11 PLIB_OSC_ReferenceOscStopInSleepEnable Function**C**

```
void PLIB_OSC_ReferenceOscStopInSleepEnable(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc  
);
```

Description

This function configures the reference oscillator to stop operating in Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscStopInSleep](#) in your application to determine whether this feature is available.

The default state of the device is for the reference oscillator to be enabled in Sleep mode.

Example

```
PLIB_OSC_ReferenceOscStopInSleepEnable(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.12 PLIB_OSC_ReferenceOscStopInSleepIsEnabled Function**C**

```
bool PLIB_OSC_ReferenceOscStopInSleepIsEnabled(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc  
);
```

Description

This function returns 'true' if the reference oscillator is disabled in Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

- true - Reference oscillator is disabled in Sleep mode
- false - Reference oscillator is enabled in Sleep mode

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscStopInSleep](#) in your application to determine whether this feature is available.

Example

```
bool refOscSleep;  
  
refOscSleep = PLIB_OSC_ReferenceOscStopInSleepIsEnabled(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.13 PLIB_OSC_ReferenceOscSwitchIsComplete Function**C**

```
bool PLIB_OSC_ReferenceOscSwitchIsComplete(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc  
);
```

Description

This function returns 'true' if the reference oscillator base clock switching is complete.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

- true - Reference clock base clock switching is complete
- false - Reference clock base clock switching is not complete; switching is not started

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscChange](#) in your application to determine whether this feature is available.

Example

```
bool refOscIdle;  
  
refOscIdle = PLIB_OSC_ReferenceOscSwitchIsComplete(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.14 PLIB_OSC_ReferenceOscTrimSet Function**C**

```
void PLIB_OSC_ReferenceOscTrimSet(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc,  
    OSC_REF_TRIM_TYPE trimValue  
);
```

Description

This function selects the reference oscillator divisor trim value. The value selected divided by OSC_REF_TRIM_MAX_VALUE will be added to the oscillator divisor value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator
trimValue	Reference oscillator trim value

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOscTrim](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_ReferenceOscTrimSet(OSC_ID_0, OSC_REFERENCE_1, 50);
```

5.6.14.6.5.15 PLIB_OSC_ReferenceOutputDisable Function**C**

```
void PLIB_OSC_ReferenceOutputDisable(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc  
);
```

Description

This function disables the reference oscillator output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOutputEnable](#) in your application to determine whether this feature is available.

The default state of the device is reference oscillator output disabled.

Example

```
PLIB_OSC_ReferenceOutputDisable(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.16 PLIB_OSC_ReferenceOutputEnable Function**C**

```
void PLIB_OSC_ReferenceOutputEnable(  
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc  
)
```

Description

This function enables the reference oscillator output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOutputEnable](#) in your application to determine whether this feature is available.

The default state of the device is reference oscillator output disabled.

Example

```
PLIB_OSC_ReferenceOutputEnable(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.5.17 PLIB_OSC_ReferenceOutputIsEnabled Function**C**

```
bool PLIB_OSC_ReferenceOutputIsEnabled(  

```

```
    OSC_MODULE_ID index,  
    OSC_REFERENCE referenceOsc  
);
```

Description

This function returns 'true' if the reference oscillator output is enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Returns

- true - Reference oscillator output is enabled
- false - Reference oscillator output is disabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsReferenceOutputEnable](#) in your application to determine whether this feature is available.

Example

```
bool refOscIdle;  
  
refOscIdle = PLIB_OSC_ReferenceOutputIsEnabled(OSC_ID_0, OSC_REFERENCE_1);
```

5.6.14.6.6 Fast RC Oscillator Setup Functions

5.6.14.6.6.1 PLIB_OSC_FRCDitherEnable Function

C

```
void PLIB_OSC_FRCDitherEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function enables the FRC Oscillator clock dithering.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_FRCDitherEnable(OSC_ID_0);
```

5.6.14.6.6.2 PLIB_OSC_FRCDivisorGet Function**C**

```
OSC_FRC_DIV PLIB_OSC_FRCDivisorGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the FRC clock divisor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values from [OSC_FRC_DIV](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsFRCDivisor](#) in your application to determine whether this feature is available.

Example

```
OSC_FRC_DIV divisorFRC;  
  
divisorFRC = PLIB_OSC_FRCDivisorGet(OSC_ID_0);
```

5.6.14.6.6.3 PLIB_OSC_FRCDivisorSelect Function**C**

```
void PLIB_OSC_FRCDivisorSelect(  
    OSC_MODULE_ID index,  
    OSC_FRC_DIV divisorFRC  
);
```

Description

This function sets the FRC clock divisor to the specified value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
divisorFRC	One of the possible values from OSC_FRC_DIV

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsFRCDivisor](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_FRCDivisorSelect ( OSC_ID_0, OSC_FRC_DIV_4 );
```

5.6.14.6.6.4 PLIB_OSC_FRCTuningModeGet Function**C**

```
OSC_TUNING_MODE PLIB_OSC_FRCTuningModeGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the FRC oscillator tuning mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible value from [OSC_TUNING_MODE](#) enum.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
if (OSC_TUNING_USING_NUMBER == PLIB_OSC_FRCTuningModeGet(OSC_ID_0))  
{  
    //Do some action  
}
```

5.6.14.6.6.5 PLIB_OSC_FRCTuningModeSet Function**C**

```
void PLIB_OSC_FRCTuningModeSet(  
    OSC_MODULE_ID index,  
    OSC_TUNING_MODE tuningMode  
);
```

Description

This function selects the FRC oscillator tuning mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
tuningMode	One of the possible value from OSC_TUNING_MODE .

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsFRCTuning](#) in your application to determine whether this feature is available.

This function takes care of the register unlock, which is required per the device data sheet. Therefore, the user need not explicitly perform the unlock sequence.

Example

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_USING_NUMBER);
```

5.6.14.6.6 PLIB_OSC_FRCTuningSelect Function**C**

```
void PLIB_OSC_FRCTuningSelect(  
    OSC_MODULE_ID index,  
    OSC_FRC_TUNE_TYPE tuningValue  
);
```

Description

This function tunes the FRC oscillator to the value specified. The application is supposed to try different values and find the best one. If the device has different tuning modes, this function will be used differently. See the example provided for the [PLIB_OSC_FRCTuningSequenceValueSet](#) function for details.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
tuningValue	Tuning value. One of the possible values from OSC_FRC_TUNE_TYPE.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsFRCTuning](#) in your application to determine whether this feature is available.

This function takes care of the register unlock, which is required per the device data sheet. Therefore, the user need not explicitly perform the unlock sequence.

Example

```
PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x05);
```

5.6.14.6.6.7 PLIB_OSC_FRCTuningSequenceValueGet Function**C**

```
OSC_FRC_TUNE_TYPE PLIB_OSC_FRCTuningSequenceValueGet(  
    OSC_MODULE_ID index,  
    OSC_TUNING_SEQUENCERS oscSequencer  
);
```

Description

This function gets the value set in the selected FRC tune sequencer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
oscSequencer	Value to be set in the sequencer

Returns

Value in the sequencer specified. One of the possible value from OSC_FRC_TUNE_TYPE.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

This API takes care of the register unlock, which is required per the device datasheet. Therefore, the need not explicitly perform the unlock sequence.

Example

```
OSC_FRC_TUNE_TYPE seqValue;  
  
seqValue = PLIB_OSC_FRCTuningSequenceValueGet(OSC_ID_0,  
        OSC_TUNING_SEQUENCER_1);
```

5.6.14.6.6.8 PLIB_OSC_FRCTuningSequenceValueSet Function**C**

```
void PLIB_OSC_FRCTuningSequenceValueSet(  
    OSC_MODULE_ID index,  
    OSC_TUNING_SEQUENCERS oscSequencer,  
    OSC_FRC_TUNE_TYPE seqValue  
);
```

Description

This function sets the value in the FRC tune sequencer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
oscSequencer	Select the sequencer
seqValue	Value to be set in the sequencer. One of the possible values from OSC_FRC_TUNE_TYPE.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_SEQ_DITHER);

PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x01);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_1,
    OSC_FRC_TUNE_MINUS_2_25_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_2,
    OSC_FRC_TUNE_MINUS_1_5_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_3,
    OSC_FRC_TUNE_MINUS_0_375_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_4,
    OSC_FRC_TUNE_PLUS_0_43_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_5,
    OSC_FRC_TUNE_PLUS_1_29_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_6,
    OSC_FRC_TUNE_PLUS_2_54_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_7,
    OSC_FRC_TUNE_MINUS_3_PERCENT);

//Configure PWM (period, pulse width and turn on the module)
```

5.6.14.6.6.9 PLIB_OSC_LinearFeedbackShiftRegGet Function

C

```
OSC_LFSR_TYPE PLIB_OSC_LinearFeedbackShiftRegGet(
    OSC_MODULE_ID index
);
```

Description

This function gets the value from the linear feedback shift register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

OSC_LFSR_TYPE - Linear Feedback shift register

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
OSC_LFSR_TYPE linearFbValue;

linearFbValue = PLIB_OSC_LinearFeedbackShiftRegGet(OSC_ID_0);
```

5.6.14.6.6.10 PLIB_OSC_LinearFeedbackShiftRegSet Function

C

```
void PLIB_OSC_LinearFeedbackShiftRegSet(  
    OSC_MODULE_ID index,  
    OSC_LFSR_TYPE linearFbValue  
);
```

Description

This function sets the linear feedback shift register. When the FRC oscillator tuning mode is configured for pseudo-random number-based dithering, this function is used to provide the seed number for the pseudo-random number generation algorithm.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
linearFbValue	The pseudo-random FRC trim value

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

This function takes care of the register unlock, which is required per the device data sheet. Therefore, the user need not explicitly perform the unlock sequence.

Example

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_PSEUDO_RANDOM);  
  
//15 bit Linear Feedback shift register value  
  
PLIB_OSC_LinearFeedbackShiftRegSet(OSC_ID_0, 0x7F);
```

5.6.14.6.7 Oscillator Switch Setup Functions

5.6.14.6.7.1 PLIB_OSC_ClockSwitchingAbort Function

C

```
void PLIB_OSC_ClockSwitchingAbort(  
    OSC_MODULE_ID index  
);
```

Description

This function aborts the oscillator switch to the selection specified by the new oscillator selection bits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsOscSwitchInit](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_ClockSwitchingAbort(OSC_ID_0);
```

5.6.14.6.7.2 PLIB_OSC_ClockSwitchingIsComplete Function**C**

```
bool PLIB_OSC_ClockSwitchingIsComplete(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the status of the oscillator switch progress.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

true - Oscillator switch is complete false - Oscillator switch is in progress

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsOscSwitchInit](#) in your application to determine whether this feature is available.

This function takes care of the register unlock, which is required per the device datasheet. Therefore, the user need not explicitly perform the unlock sequence.

Example

```
PLIB_OSC_SysClockSelect(OSC_ID_0, OSC_PRIMARY);  
  
while(!PLIB_OSC_ClockSwitchingIsComplete(OSC_ID_0));
```

5.6.14.6.7.3 PLIB_OSC_CurrentSysClockGet Function**C**

```
OSC_SYS_TYPE PLIB_OSC_CurrentSysClockGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the current oscillator. If the application hasn't changed the oscillator selection, this will be same as the oscillator selected through the Configuration bits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values from [OSC_SYS_TYPE](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsOscCurrentGet](#) in your application to determine whether this feature is available.

Example

```
OSC_SYS_TYPE oscCurrent;  
  
oscCurrent = PLIB_OSC_CurrentSysClockGet(OSC_ID_0);
```

5.6.14.6.7.4 PLIB_OSC_SysClockSelect Function**C**

```
void PLIB_OSC_SysClockSelect(  
    OSC_MODULE_ID index,  
    OSC_SYS_TYPE newOsc  
)
```

Description

This function selects the new oscillator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
newOsc	One of the possible values from OSC_SYS_TYPE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsOscSelect](#) in your application to determine whether this feature is available.

This function takes care of the register unlock, which is required per the device datasheet. Therefore, the user need not explicitly perform the unlock sequence.

This function adds the necessary delay (NOP instructions) after switching the oscillator. Therefore, the user need not add any delay as specified in the device data sheet.

Example

```
PLIB_OSC_SysClockSelect(OSC_ID_0, OSC_PRIMARY);
```

5.6.14.6.8 Doze Setup Functions

5.6.14.6.8.1 PLIB_OSC_DozeModeDisable Function

C

```
void PLIB_OSC_DozeModeDisable(  
    OSC_MODULE_ID index  
);
```

Description

This function disables Doze mode. The CPU peripheral clock ratio is set to 1:1.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

This function adds the necessary delay (NOP instructions), which must be added immediately before entering Doze mode and immediately after exiting Doze mode.

Example

```
PLIB_OSC_DozeModeDisable(OSC_ID_0);
```

5.6.14.6.8.2 PLIB_OSC_DozeModeEnable Function

C

```
void PLIB_OSC_DozeModeEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function enables Doze mode. This is a method to reduce power consumption By reducing the processor clock. Use the [PLIB_OSC_DozeRatioSelect](#) function to specify the CPU peripheral clock ratio.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

This function adds the necessary delay (NOP instructions), which must be added immediately before entering Doze mode and immediately after exiting Doze mode.

Example

```
PLIB_OSC_DozeModeEnable(OSC_ID_0);
```

5.6.14.6.8.3 PLIB_OSC_DozeRatioGet Function**C**

```
OSC_DOZE_RATIO PLIB_OSC_DozeRatioGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the Doze ratio, which is the ratio between the processor clock and the peripheral clock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values of [OSC_DOZE_RATIO](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
OSC_DOZE_RATIO dozeRatio;  
  
dozeRatio = PLIB_OSC_DozeRatioGet(OSC_ID_0);
```

5.6.14.6.8.4 PLIB_OSC_DozeRatioSelect Function**C**

```
void PLIB_OSC_DozeRatioSelect(  
    OSC_MODULE_ID index,  
    OSC_DOZE_RATIO dozeRatio  
);
```

Description

This function selects the Doze ratio, which is the ratio between the processor clock and the peripheral clock. By default, the ratio between the processor clock and the peripheral clock is 1:1.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
dozeRatio	One of the possible values of OSC_DOZE_RATIO

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_DozeRatioSelect(OSC_ID_0, OSC_DOZE_RATIO_4);
```

5.6.14.6.8.5 PLIB_OSC_DozeRecoverOnInterruptDisable Function**C**

```
void PLIB_OSC_DozeRecoverOnInterruptDisable(  
    OSC_MODULE_ID index  
);
```

Description

This function disables feature of recovering from Doze mode upon an interrupt. Interrupts will not affect the power-saving Doze mode enable.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_DozeRecoverOnInterruptDisable(OSC_ID_0);
```

5.6.14.6.8.6 PLIB_OSC_DozeRecoverOnInterruptEnable Function**C**

```
void PLIB_OSC_DozeRecoverOnInterruptEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function enables the feature of recovering from Doze mode upon an interrupt. Interrupts clear the power-saving Doze mode enable and reset the CPU peripheral clock ratio to 1:1.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_DozeRecoverOnInterruptEnable(OSC_ID_0);
```

5.6.14.6.8.7 PLIB_OSC_DozeRecoverOnInterruptIsEnabled Function**C**

```
bool PLIB_OSC_DozeRecoverOnInterruptIsEnabled(  
    OSC_MODULE_ID index  
);
```

Description

This function returns 'true' if recover from Doze mode on interrupt is enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

- true - Recover from Doze mode on interrupt is enabled
- false - Recover from Doze mode on interrupt is disabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
bool recoverOnInt;  
  
recoverOnInt = PLIB_OSC_DozeRecoverOnInterruptIsEnabled(OSC_ID_0);
```

5.6.14.6.9 USB and Display Clock Setup Functions**5.6.14.6.9.1 PLIB_OSC_GraphicsClockDivisorGet Function****C**

```
uint8_t PLIB_OSC_GraphicsClockDivisorGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the Graphics module clock divisor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

Graphics module clock divisor.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
uint8_t    displayClockDiv;

displayClockDiv = PLIB_OSC_GraphicsClockDivisorGet(OSC_ID_0);
```

5.6.14.6.9.2 PLIB_OSC_GraphicsClockDivisorSet Function**C**

```
void PLIB_OSC_GraphicsClockDivisorSet(
    OSC_MODULE_ID index,
    uint8_t clockDivisor
);
```

Description

This function sets the Graphics module clock divisor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
clockDivisor	Graphics module clock divisor

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_GraphicsClockDivisorSet(OSC_ID_0, 127);
```

5.6.14.6.9.3 PLIB_OSC_GraphicsClockSourceGet Function**C**

```
OSC_GFX_CLOCK PLIB_OSC_GraphicsClockSourceGet(
    OSC_MODULE_ID index
);
```

Description

This function gets the clock frequency used for the Graphics Controller module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
OSC_GFX_CLOCK gfxClock;  
  
gfxClock = PLIB_OSC_GraphicsClockSourceGet (OSC_ID_0);
```

5.6.14.6.9.4 PLIB_OSC_GraphicsClockSourceSelect Function

C

```
void PLIB_OSC_GraphicsClockSourceSelect(  
    OSC_MODULE_ID index,  
    OSC_GFX_CLOCK gfxClock  
);
```

Description

This function sets the clock frequency for the Graphics Controller module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
gfxClock	One of the possible values of OSC_GFX_CLOCK

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_GraphicsClockSourceSelect (OSC_ID_0, OSC_GFX_CLOCK_96M);
```

5.6.14.6.9.5 PLIB_OSC_UsbClockSourceGet Function

C

```
OSC_USBCLOCK_SOURCE PLIB_OSC_UsbClockSourceGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the USB module clock source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

USB module clock source. One of the possible values from [OSC_USBCLOCK_SOURCE](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsUsbClockSource](#) in your application to determine whether this feature is available.

Example

```
if (SYS_OSC_USBCLK_FRC == PLIB_OSC_UsbClockSourceGet(OSC_ID_0))
{
    //Do some action
}
```

5.6.14.6.9.6 PLIB_OSC_UsbClockSourceSelect Function

C

```
void PLIB_OSC_UsbClockSourceSelect(
    OSC_MODULE_ID index,
    OSC_USBCLOCK_SOURCE usbClock
);
```

Description

This function sets the USB module clock source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
usbClock	Select the USB module clock source. One of the possible values from OSC_USBCLOCK_SOURCE .

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsUsbClockSource](#) in your application to determine whether this feature is available.

Before placing the USB module in Suspend mode, use this function to enable the FRC clock.

Example

```
PLIB_OSC_UsbClockSourceSelect(OSC_ID_0, SYS_OSC_USBCLK_FRC);
```

5.6.14.6.10 PLL Setup Functions

5.6.14.6.10.1 PLIB_OSC_PLLAuxClockSourceGet Function

C

```
OSC_PLLAUX_CLOCK_SOURCE PLIB_OSC_PLLAuxClockSourceGet(
    OSC_MODULE_ID index
);
```

Description

This function returns the clock source selected for the Auxiliary PLL.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

The source for the auxiliary clock. One of the possible values from [OSC_PLLAUX_CLOCK_SOURCE](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
if(OSC_PLLAUX_CLOCK == PLIB_OSC_PLLAuxClockSourceGet(OSC_ID_0))
{
    //Take some action
}
```

5.6.14.6.10.2 PLIB_OSC_PLLAuxClockSourceSelect Function**C**

```
void PLIB_OSC_PLLAuxClockSourceSelect(
    OSC_MODULE_ID index,
    OSC_PLLAUX_CLOCK_SOURCE auxPLLClkSrc
);
```

Description

This function sets the Auxiliary PLL clock source to either the Primary oscillator or the Auxiliary oscillator depending on the input.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
auxPLLClkSrc	The source for the auxiliary clock. One of the possible values from OSC_PLLAUX_CLOCK_SOURCE .

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_PLLAuxClockSourceSelect(OSC_ID_0, OSC_PLLAUX_CLOCK);
```

5.6.14.6.10.3 PLIB_OSC_PLLAuxInputDivisorGet Function**C**

```
OSC_AUXPLL_IN_DIV PLIB_OSC_PLLAuxInputDivisorGet(
```

```
    OSC_MODULE_ID index  
);
```

Description

This function gets the input divisor for the Auxiliary PLL.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values from [OSC_AUXPLL_IN_DIV](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
OSC_AUXPLL_IN_DIV auxPLLInDiv;  
  
auxPLLInDiv = PLIB_OSC_PLLAuxInputDivisorGet(OSC_ID_0);
```

5.6.14.6.10.4 PLIB_OSC_PLLAuxInputDivisorSelect Function

C

```
void PLIB_OSC_PLLAuxInputDivisorSelect(  
    OSC_MODULE_ID index,  
    OSC_AUXPLL_IN_DIV auxPLLInDiv  
);
```

Description

This function sets the input divisor for the Auxiliary PLL to the specified value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
auxPLLInDiv	One of the possible values from OSC_AUXPLL_IN_DIV

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_PLLAuxInputDivisorSelect(OSC_ID_0, OSC_AUXPLL_IN_DIV_2);
```

5.6.14.6.10.5 PLIB_OSC_PLLAuxIsLocked Function

C

```
bool PLIB_OSC_PLLAuxIsLocked(  

```



```
    OSC_MODULE_ID index  
);
```

Description

This function returns 'true' if the Auxiliary PLL is locked.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

- true - Auxiliary PLL is in lock
- false - Auxiliary PLL is not in lock

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
bool lockAuxPLL;  
  
lockAuxPLL = PLIB_OSC_PLLAuxIsLocked(OSC_ID_0);
```

5.6.14.6.10.6 PLIB_OSC_PLLAuxMultiplierGet Function

C

```
OSC_AUXPLL_MULTIPLIER PLIB_OSC_PLLAuxMultiplierGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the Auxiliary PLL multiplier.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values from OSC_PLL_MUL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
OSC_AUXPLL_MULTIPLIER auxpllMultiply;  
  
auxpllMultiply = PLIB_OSC_PLLAuxMultiplierGet(OSC_ID_0);
```

5.6.14.6.10.7 PLIB_OSC_PLLAuxMultiplierSelect Function

C

```
void PLIB_OSC_PLLAuxMultiplierSelect(  
    OSC_MODULE_ID index,  
    OSC_AUXPLL_MULTIPLIER auxPllMultiplier  
);
```

Description

This function sets the Auxiliary PLL multiplier to the specified value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
auxPllMultiplier	One of the possible values from OSC_AUXPLL_MULTIPLIER

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_PLLAuxMultiplierSelect (OSC_ID_0, OSC_AUXPLL_MULTIPLIER_4);
```

5.6.14.6.10.8 PLIB_OSC_PLLAuxOutputDivisorGet Function

C

```
OSC_AUXPLL_OUT_DIV PLIB_OSC_PLLAuxOutputDivisorGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the output divisor for the Auxiliary PLL.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values from [OSC_AUXPLL_OUT_DIV](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
OSC_AUXPLL_OUT_DIV auxOutputDiv;  
  
auxOutputDiv = PLIB_OSC_PLLAuxOutputDivisorGet(OSC_ID_0);
```

5.6.14.6.10.9 PLIB_OSC_PLLAuxOutputDivisorSet Function

C

```
void PLIB_OSC_PLLAuxOutputDivisorSet(  
    OSC_MODULE_ID index,  
    OSC_AUXPLL_OUT_DIV auxOutputDiv  
);
```

Description

This function sets the output divisor for the Auxiliary PLL to the specified value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
auxOutputDiv	One of the possible values from OSC_AUXPLL_OUT_DIV

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_PLLAuxOutputDivisorSet(OSC_ID_0, OSC_AUXPLL_OUT_DIV_2);
```

5.6.14.6.10.10 PLIB_OSC_PLLClockIsLocked Function

C

```
bool PLIB_OSC_PLLClockIsLocked(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the lock status for the clock and PLL selections.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

- true - clock and PLL selections are locked
- false - clock and PLL selections are not locked

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPLLClockLock](#) in your application to determine whether this feature is available.

If the PLL does not stabilize properly during start-up, this function may not reflect the actual status of PLL lock, nor does it detect

when the PLL loses lock during normal operation.

Example

```
bool clockPLL_st;  
  
clockPLL_st = PLIB_OSC_PLLClockIsLocked(OSC_ID_0);
```

5.6.14.6.10.11 PLIB_OSC_PLLClockLock Function**C**

```
void PLIB_OSC_PLLClockLock(  
    OSC_MODULE_ID index  
);
```

Description

This function locks the clock and PLL selections.

Preconditions

The Fail-Safe Clock Monitor (FSCM) should be enabled.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPLLClockLock](#) in your application to determine whether this feature is available.

The data given by this API is valid only if clock switching and monitoring is enabled. Otherwise Clock and PLL selections are never locked and may be modified.

Example

```
PLIB_OSC_PLLClockLock(OSC_ID_0);
```

5.6.14.6.10.12 PLIB_OSC_PLLClockUnlock Function**C**

```
void PLIB_OSC_PLLClockUnlock(  
    OSC_MODULE_ID index  
);
```

Description

This function unlocks the clock and PLL selection so that the clock and PLL may be modified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPLL](#) in your application to determine whether this feature is available.

This function takes care of the register unlock, which is required per the device data sheet. Therefore, the user need not explicitly perform the unlock sequence.

Example

```
PLIB_OSC_PLLClockUnlock(OSC_ID_0);
```

5.6.14.6.10.13 PLIB_OSC_PLLEnable Function**C**

```
void PLIB_OSC_PLLEnable(  
    OSC_MODULE_ID index,  
    OSC_PLL_SELECT selectPLL  
);
```

Description

This function disables the the specified PLL.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
selectPLL	PLL to be enabled. One of the possible values from OSC_PLL_SELECT .

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_PLLEnable(OSC_ID_0, OSC_PLL_SYSTEM);
```

5.6.14.6.10.14 PLIB_OSC_PLLEnable Function**C**

```
void PLIB_OSC_PLLEnable(  
    OSC_MODULE_ID index,  
    OSC_PLL_SELECT selectPLL  
);
```

Description

This function enables the the specified PLL.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

selectPLL	PLL to be enabled. One of the possible values from OSC_PLL_SELECT .

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
PLIB_OSC_PLLEnable(OSC_ID_0, OSC_PLL_SYSTEM);
```

5.6.14.6.10.15 PLIB_OSC_PLLIsEnabled Function**C**

```
bool PLIB_OSC_PLLIsEnabled(  
    OSC_MODULE_ID index,  
    OSC_PLL_SELECT selectPLL  
);
```

Description

This function returns 'true' if the specified PLL is enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
selectPLL	PLL to be enabled. One of the possible values from OSC_PLL_SELECT .

Returns

- true - The specified PLL is enabled
- false - The specified PLL is disabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability.

Example

```
if (PLIB_OSC_PLLIsEnabled(OSC_ID_0, OSC_PLL_SYSTEM))  
{  
    //Do some action  
}
```

5.6.14.6.10.16 PLIB_OSC_PLLIsLocked Function**C**

```
bool PLIB_OSC_PLLIsLocked(  
    OSC_MODULE_ID index,  
    OSC_PLL_SELECT pllselect  
);
```

Description

This function returns the status of the PLL lock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllselect	Selects the PLL.

Returns

- true - PLL module is in lock or PLL module start-up timer is satisfied
- false - PLL module is out of lock, PLL start-up timer is running, or PLL is disabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPLLLockStatus](#) in your application to determine whether this feature is available.

If the PLL does not stabilize properly during start-up, this function may not reflect the actual status of PLL lock, nor does it detect when the PLL loses lock during normal operation.

Example

```
bool lockPLL_status;  
  
lockPLL_status = PLIB_OSC_PLLIsLocked(OSC_ID_0, OSC_PLL_SYSTEM);
```

5.6.14.6.10.17 PLIB_OSC_SysPLLFrequencyRangeGet Function**C**

```
OSC_SYSPLL_FREQ_RANGE PLIB_OSC_SysPLLFrequencyRangeGet(  
    OSC_MODULE_ID index  
);
```

Description

This function returns the frequency range set for PLL

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

PLLFrequencyRange - One of the possible values from [OSC_SYSPLL_FREQ_RANGE](#)

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLFrequencyRange](#) in your application to determine whether this feature is available.

Example

```
OSC_SYSPLL_FREQ_RANGE PLLFrequencyRange;  
PLLFrequencyRange = PLIB_OSC_SysPLLFrequencyRangeGet(OSC_ID_0);
```

5.6.14.6.10.18 PLIB_OSC_SysPLLFrequencyRangeSet Function**C**

```
void PLIB_OSC_SysPLLFrequencyRangeSet(  
    OSC_MODULE_ID index,  
    OSC_SYSPLL_FREQ_RANGE PLLFrequencyRange
```

```
);
```

Description

This function sets the frequency range for PLL

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLFrequencyRange	One of the possible values from OSC_SYSPLL_FREQ_RANGE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLFrequencyRange](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_SysPLLFrequencyRangeSet(OSC_ID_0, OSC_SYSPLL_FREQ_RANGE_5M_TO_10M);
```

5.6.14.6.10.19 PLIB_OSC_SysPLLInputClockSourceGet Function**C**

```
OSC_SYSPLL_IN_CLK_SOURCE PLIB_OSC_SysPLLInputClockSourceGet(  
    OSC_MODULE_ID index  
);
```

Description

This function returns the input clock source for the PLL module

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

PLLInClockSource - One of the possible values from [OSC_SYSPLL_IN_CLK_SOURCE](#)

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLInputClockSource](#) in your application to determine whether this feature is available.

Example

```
OSC_SYSPLL_IN_CLK_SOURCE PLLInClockSource;  
PLLInClockSource = PLIB_OSC_SysPLLInputClockSourceGet(OSC_ID_0);
```

5.6.14.6.10.20 PLIB_OSC_SysPLLInputClockSourceSet Function**C**

```
void PLIB_OSC_SysPLLInputClockSourceSet(  

```



```
    OSC_MODULE_ID index,  
    OSC_SYSPLL_IN_CLK_SOURCE PLLInClockSource  
);
```

Description

This function sets the input clock source for the PLL module

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInClockSource	One of the possible values from OSC_SYSPLL_IN_CLK_SOURCE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLInputClockSource](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_SysPLLInputClockSourceSet(OSC_ID_0, OSC_SYSPLL_IN_CLK_SOURCE_FRC);
```

5.6.14.6.10.21 PLIB_OSC_SysPLLInputDivisorGet Function**C**

```
OSC_SYSPLL_IN_DIV PLIB_OSC_SysPLLInputDivisorGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the input divisor for the PLL.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values from [OSC_SYSPLL_IN_DIV](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLInputDivisor](#) in your application to determine whether this feature is available.

Example

```
OSC_SYSPLL_IN_DIV PLLInDiv;  
  
PLLInDiv = PLIB_OSC_SysPLLInputDivisorGet(OSC_ID_0);
```

5.6.14.6.10.22 PLIB_OSC_SysPLLInputDivisorSet Function

C

```
void PLIB_OSC_SysPLLInputDivisorSet(  
    OSC_MODULE_ID index,  
    OSC_SYSPLL_IN_DIV PLLInDiv  
);
```

Description

This function sets the input divider for the PLL to the specified value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInDiv	One of the possible values from OSC_SYSPLL_In_DIV

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLInputDivisor](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_SysPLLInputDivisorSet(OSC_ID_0, OSC_SYSPLL_IN_DIV_1);
```

5.6.14.6.10.23 PLIB_OSC_SysPLLMultiplierGet Function

C

```
OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_SysPLLMultiplierGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the PLL multiplier.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values of PB clock divisor of type [OSC_SYSPLL_MULTIPLIER_TYPE](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLMultiplier](#) in your application to determine whether this feature is available.

Actual Multiplier value will be returned, NOT the 'PLLMULT' field value. Refer the data sheet of the device for the detail.

Example

```
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiply;  
  
pll_multiply = PLIB_OSC_SysPLLMultiplierGet(OSC_ID_0);
```

5.6.14.6.10.24 PLIB_OSC_SysPLLMultiplierSelect Function**C**

```
void PLIB_OSC_SysPLLMultiplierSelect(  
    OSC_MODULE_ID index,  
    OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier  
);
```

Description

This function sets the PLL multiplier to the specified value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pll_multiplier	One of the possible values of PB clock divisor of type OSC_PLL_MUL_TYPE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLMultiplier](#) in your application to determine whether this feature is available.

Use PLL Multiplier value directly for the parameter 'pll_multiplier', NOT the value of the 'PLLMULT' field. Library behaviour is undefined in case of used PLL Multiplier value is not supported by the selected device. Refer the data sheet of the device for the detail.

Example

```
PLIB_OSC_SysPLLMultiplierSelect (OSC_ID_0, 0x08);
```

5.6.14.6.10.25 PLIB_OSC_SysPLLOutputDivisorGet Function**C**

```
OSC_SYSPLL_OUT_DIV PLIB_OSC_SysPLLOutputDivisorGet(  
    OSC_MODULE_ID index  
);
```

Description

This function gets the output divisor for the PLL.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

One of the possible values from [OSC_SYSPLL_OUT_DIV](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLOutputDivisor](#) in your application to determine whether this feature is available.

Example

```
OSC_SYSPLL_OUT_DIV PLLOutDiv;  
  
PLLOutDiv = PLIB_OSC_SysPLLOutputDivisorGet(OSC_ID_0);
```

5.6.14.6.10.26 PLIB_OSC_SysPLLOutputDivisorSet Function**C**

```
void PLIB_OSC_SysPLLOutputDivisorSet(  
    OSC_MODULE_ID index,  
    OSC_SYSPLL_OUT_DIV PLLOutDiv  
);
```

Description

This function sets the output divider for the PLL to the specified value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLOutDiv	One of the possible values from OSC_SYSPLL_OUT_DIV

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsSysPLLOutputDivisor](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_SysPLLOutputDivisorSelect(OSC_ID_0, OSC_SYSPLL_OUT_DIV_1);
```

5.6.14.6.11 Peripheral Clock Setup Functions**5.6.14.6.11.1 PLIB_OSC_PBClockDivisorGet Function****C**

```
OSC_PB_CLOCK_DIV_TYPE PLIB_OSC_PBClockDivisorGet(  
    OSC_MODULE_ID index,  
    OSC_PERIPHERAL_BUS peripheralBusNumber  
);
```

Description

This function gets the peripheral bus clock divisor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Returns

One of the possible values of PB clock divisor of type [OSC_PB_CLOCK_DIV_TYPE](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPBClockDivisor](#) in your application to determine whether this feature is available.

Actual Divisor value will be returned, NOT the 'PBDIV' field value. Refer the data sheet of the device for the detail.

Example

```
OSC_PB_CLOCK_DIV_TYPE peripheralBusClkDiv;  
  
peripheralBusClkDiv = PLIB_OSC_PBClockDivisorGet(OSC_ID_0,  
                                                  OSC_PERIPHERAL_BUS_1);
```

5.6.14.6.11.2 PLIB_OSC_PBClockDivisorIsReady Function**C**

```
bool PLIB_OSC_PBClockDivisorIsReady(  
    OSC_MODULE_ID index,  
    OSC_PERIPHERAL_BUS peripheralBusNumber  
);
```

Description

This function checks whether the peripheral bus clock divisor is ready to be written.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Returns

- true - Peripheral bus clock divisor can be written
- false - Peripheral bus clock divisor cannot be written

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPBClockReady](#) in your application to determine whether this feature is available.

Example

```
if (PLIB_OSC_PBClockDivisorIsReady(OSC_ID_0, OSC_PERIPHERAL_BUS_1))  
{  
    PLIB_OSC_PBClockDivisorSet (OSC_ID_0, OSC_PERIPHERAL_BUS_1,  
                                0x01);  
}
```

```
}
```

5.6.14.6.11.3 PLIB_OSC_PBClockDivisorSet Function

C

```
void PLIB_OSC_PBClockDivisorSet(  
    OSC_MODULE_ID index,  
    OSC_PERIPHERAL_BUS peripheralBusNumber,  
    OSC_PB_CLOCK_DIV_TYPE peripheralBusClkDiv  
);
```

Description

This function sets the peripheral bus clock divisor to the specified value.

Preconditions

Peripheral bus clock divisor should be ready to be written.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus
peripheralBusClkDiv	One of the possible values of PB clock divisor of type OSC_PB_CLOCK_DIV_TYPE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPBClockDivisor](#) in your application to determine whether this feature is available.

Use PB Divider value directly for the parameter 'peripheralBusClkDiv', NOT the value of the 'PBDIV' field. Library behaviour is undefined in case of used PB Divider value is not supported by the selected device. Refer the data sheet of the device for the detail.

Example

```
PLIB_OSC_PBClockDivisorSet (OSC_ID_0, OSC_PERIPHERAL_BUS_1, 0x01);
```

5.6.14.6.11.4 PLIB_OSC_PBOutputClockDisable Function

C

```
void PLIB_OSC_PBOutputClockDisable(  
    OSC_MODULE_ID index,  
    OSC_PERIPHERAL_BUS peripheralBusNumber  
);
```

Description

This function disables the peripheral bus output clock

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Returns

None

Remarks

The clock for peripheral bus 1 cannot be turned off.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPBClockOutputEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_PBClockDisable (OSC_ID_0, OSC_PERIPHERAL_BUS_2);
```

5.6.14.6.11.5 PLIB_OSC_PBClockEnable Function

C

```
void PLIB_OSC_PBClockEnable(  
    OSC_MODULE_ID index,  
    OSC_PERIPHERAL_BUS peripheralBusNumber  
);
```

Description

This function enables the peripheral bus output clock

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Returns

None

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPBClockOutputEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_OSC_PBClockEnable (OSC_ID_0, OSC_PERIPHERAL_BUS_2);
```

5.6.14.6.11.6 PLIB_OSC_PBClockIsEnabled Function

C

```
bool PLIB_OSC_PBClockIsEnabled(  
    OSC_MODULE_ID index,  
    OSC_PERIPHERAL_BUS peripheralBusNumber  
);
```

Description

This function checks whether the peripheral bus clock output is enabled or not

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Returns

- true - Peripheral bus clock output is enabled
- false - Peripheral bus clock output is disabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsPBClockOutputEnable](#) in your application to determine whether this feature is available.

Example

```
if (PLIB_OSC_PBOutputClockIsEnabled(OSC_ID_0, OSC_PERIPHERAL_BUS_2))
{
    PLIB_OSC_PBOutputClockDisable(OSC_ID_0, OSC_PERIPHERAL_BUS_2);
}
```

5.6.14.6.12 Clock Fail Monitoring Functions

5.6.14.6.12.1 PLIB_OSC_ClockHasFailed Function

C

```
bool PLIB_OSC_ClockHasFailed(
    OSC_MODULE_ID index
);
```

Description

This function returns 'true' if the clock fails. Monitors the Fail-Safe Clock Monitor (FSCM).

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Returns

- true - Fail-Safe Clock Monitor (FSCM) detected a clock failure
- false - No clock failure has been detected

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_OSC_ExistsClockFail](#) in your application to determine whether this feature is available.

Example

```
bool clockStatus;
clockStatus = PLIB_OSC_ClockHasFailed(OSC_ID_0);
```

5.6.14.6.13 Data Types and Constants

5.6.14.6.13.1 OSC_PB_CLOCK_DIV_TYPE Macro

C

```
#define OSC_PB_CLOCK_DIV_TYPE SFR_TYPE
```

Description

Oscillator Peripheral Bus Clock Divisor Value Type

This macro defines the type of the oscillator PB Clock divisor value.

Remarks

None

5.6.14.6.13.2 OSC_REF_DIVISOR_TYPE Macro

C

```
#define OSC_REF_DIVISOR_TYPE SFR_TYPE
```

Description

Reference oscillator divisor type

This macro defines the reference oscillator divisor type. Please refer to the specific device data sheet to determine the possible values of this type.

Remarks

None.

5.6.14.6.13.3 OSC_REFERENCE_MAX_DIV Macro

C

```
#define OSC_REFERENCE_MAX_DIV 65534
```

Description

Reference clock output divisor maximum

This macro defines the reference clock output divisor maximum value.

Remarks

None

5.6.14.6.13.4 OSC_SYSPLL_MULTIPLIER_TYPE Macro

C

```
#define OSC_SYSPLL_MULTIPLIER_TYPE SFR_TYPE
```

Description

Oscillator System PLL Multiplier Value Type

This macro defines the type of the oscillator system PLL multiplier value.

Remarks

None

5.6.14.6.13.5 OSC_AUX_CLOCK_SOURCE Enumeration

C

```
typedef enum {  
    OSC_AUX_CLOCK_PRIMARY  
} OSC_AUX_CLOCK_SOURCE;
```

Description

AUX PLL Input divider.

This enumeration lists the possible Auxiliary PLL input divider values.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.6 OSC_AUX_MODE Enumeration

C

```
typedef enum {  
    OSC_AUX_MODE_EC,  
    OSC_AUX_MODE_XT,  
    OSC_AUX_MODE_HS,  
    OSC_AUX_DISABLE  
} OSC_AUX_MODE;
```

Description

AUX Oscillator modes.

This enumeration lists the possible Auxiliary Oscillator mode values.

Members

Members	Description
OSC_AUX_MODE_EC	External clock Input
OSC_AUX_MODE_XT	Crystal/resonator
OSC_AUX_MODE_HS	High speed crystal
OSC_AUX_DISABLE	AUX Oscillator disabled

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.7 OSC_AUXPLL_IN_DIV Enumeration

C

```
typedef enum {  
    OSC_AUXPLL_IN_DIV_1,  
    OSC_AUXPLL_IN_DIV_2,  
    OSC_AUXPLL_IN_DIV_3,  
    OSC_AUXPLL_IN_DIV_4,  
    OSC_AUXPLL_IN_DIV_5,  
    OSC_AUXPLL_IN_DIV_6,  
    OSC_AUXPLL_IN_DIV_10,  
    OSC_AUXPLL_IN_DIV_12  
} OSC_AUXPLL_IN_DIV;
```

Description

AUX PLL Input divider.

This enumeration lists the possible Auxiliary PLL input divider values.

Members

Members	Description
OSC_AUXPLL_IN_DIV_1	Input divided by 1
OSC_AUXPLL_IN_DIV_2	Input divided by 2
OSC_AUXPLL_IN_DIV_3	Input divided by 3
OSC_AUXPLL_IN_DIV_4	Input divided by 4
OSC_AUXPLL_IN_DIV_5	Input divided by 5
OSC_AUXPLL_IN_DIV_6	Input divided by 6
OSC_AUXPLL_IN_DIV_10	Input divided by 10
OSC_AUXPLL_IN_DIV_12	Input divided by 12

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.8 OSC_AUXPLL_MULTIPLIER Enumeration

C

```
typedef enum {  
    OSC_AUXPLL_MULTIPLIER_4  
} OSC_AUXPLL_MULTIPLIER;
```

Description

Aux PLL Multiplier

This enumeration lists the possible Auxiliary PLL multiplier

Members

Members	Description
OSC_AUXPLL_MULTIPLIER_4	Auxiliary PLL multiplier value is 1:4

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.9 OSC_AUXPLL_OUT_DIV Enumeration

C

```
typedef enum {  
    OSC_AUXPLL_OUT_DIV_1,  
    OSC_AUXPLL_OUT_DIV_2,  
    OSC_AUXPLL_OUT_DIV_4,  
    OSC_AUXPLL_OUT_DIV_8,  
    OSC_AUXPLL_OUT_DIV_16,  
    OSC_AUXPLL_OUT_DIV_32,  
    OSC_AUXPLL_OUT_DIV_64,  
    OSC_AUXPLL_OUT_DIV_256  
} OSC_AUXPLL_OUT_DIV;
```

Description

AUX PLL Output divider.

This enumeration lists the possible Auxiliary PLL output divider values.

Members

Members	Description
OSC_AUXPLL_OUT_DIV_1	output divided by 1
OSC_AUXPLL_OUT_DIV_2	output divided by 2
OSC_AUXPLL_OUT_DIV_4	output divided by 4
OSC_AUXPLL_OUT_DIV_8	output divided by 8
OSC_AUXPLL_OUT_DIV_16	output divided by 16
OSC_AUXPLL_OUT_DIV_32	output divided by 32
OSC_AUXPLL_OUT_DIV_64	output divided by 64
OSC_AUXPLL_OUT_DIV_256	output divided by 256

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.10 OSC_DOZE_RATIO Enumeration**C**

```
typedef enum {
    OSC_DOZE_RATIO_1,
    OSC_DOZE_RATIO_2,
    OSC_DOZE_RATIO_4,
    OSC_DOZE_RATIO_8,
    OSC_DOZE_RATIO_16,
    OSC_DOZE_RATIO_32,
    OSC_DOZE_RATIO_64,
    OSC_DOZE_RATIO_128
} OSC_DOZE_RATIO;
```

Description

DOZE bits(CPU peripheral clock ratio selection).

This enumeration lists the possible doze values(CPU clock to peripheral clock ratio).

Members

Members	Description
OSC_DOZE_RATIO_1	CPU peripheral clock ratio 1:1
OSC_DOZE_RATIO_2	CPU peripheral clock ratio 1:2
OSC_DOZE_RATIO_4	CPU peripheral clock ratio 1:4
OSC_DOZE_RATIO_8	CPU peripheral clock ratio 1:8
OSC_DOZE_RATIO_16	CPU peripheral clock ratio 1:16
OSC_DOZE_RATIO_32	CPU peripheral clock ratio 1:32
OSC_DOZE_RATIO_64	CPU peripheral clock ratio 1:64
OSC_DOZE_RATIO_128	CPU peripheral clock ratio 1:128

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.11 OSC_FRC_DIV Enumeration**C**

```
typedef enum {
    OSC_FRC_DIV_1,
```

```

OSC_FRC_DIV_2,
OSC_FRC_DIV_4,
OSC_FRC_DIV_8,
OSC_FRC_DIV_16,
OSC_FRC_DIV_32,
OSC_FRC_DIV_64,
OSC_FRC_DIV_256
} OSC_FRC_DIV;

```

Description

FRC divider.

This enumeration lists the possible FRC Oscillator divider values.

Members

Members	Description
OSC_FRC_DIV_1	Fast RC Oscillator output Divide by 1
OSC_FRC_DIV_2	Fast RC Oscillator output Divide by 2
OSC_FRC_DIV_4	Fast RC Oscillator output Divide by 4
OSC_FRC_DIV_8	Fast RC Oscillator output Divide by 8
OSC_FRC_DIV_16	Fast RC Oscillator output Divide by 16
OSC_FRC_DIV_32	Fast RC Oscillator output Divide by 32
OSC_FRC_DIV_64	Fast RC Oscillator output Divide by 64
OSC_FRC_DIV_256	Fast RC Oscillator output Divide by 256

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.12 OSC_FRC_TUNE Enumeration**C**

```

typedef enum {
    OSC_FRC_TUNE_PLUS_3_PERCENT,
    OSC_FRC_TUNE_PLUS_2_57_PERCENT,
    OSC_FRC_TUNE_PLUS_2_54_PERCENT,
    OSC_FRC_TUNE_PLUS_1_71_PERCENT,
    OSC_FRC_TUNE_PLUS_1_29_PERCENT,
    OSC_FRC_TUNE_PLUS_0_86_PERCENT,
    OSC_FRC_TUNE_PLUS_0_43_PERCENT,
    OSC_FRC_TUNE_CENTRAL_FREQ,
    OSC_FRC_TUNE_MINUS_0_375_PERCENT,
    OSC_FRC_TUNE_MINUS_0_75_PERCENT,
    OSC_FRC_TUNE_MINUS_1_125_PERCENT,
    OSC_FRC_TUNE_MINUS_1_5_PERCENT,
    OSC_FRC_TUNE_MINUS_1_875_PERCENT,
    OSC_FRC_TUNE_MINUS_2_25_PERCENT,
    OSC_FRC_TUNE_MINUS_2_625_PERCENT,
    OSC_FRC_TUNE_MINUS_3_PERCENT
} OSC_FRC_TUNE;

```

Description

Oscillator tuning value enumeration

This enumeration lists the possible oscillator tuning values, and is used by the [PLIB_OSC_FRCTuningSequenceValueSet](#) and [PLIB_OSC_FRCTuningSequenceValueGet](#) functions.

Members

Members	Description
OSC_FRC_TUNE_PLUS_3_PERCENT	OSC tune to Central frequency + 3
OSC_FRC_TUNE_PLUS_2_57_PERCENT	OSC tune to Central frequency + 2.57
OSC_FRC_TUNE_PLUS_2_54_PERCENT	OSC tune to Central frequency + 2.54
OSC_FRC_TUNE_PLUS_1_71_PERCENT	OSC tune to Central frequency + 1.71
OSC_FRC_TUNE_PLUS_1_29_PERCENT	OSC tune to Central frequency + 1.29
OSC_FRC_TUNE_PLUS_0_86_PERCENT	OSC tune to Central frequency + 0.86
OSC_FRC_TUNE_PLUS_0_43_PERCENT	OSC tune to Central frequency + 0.43
OSC_FRC_TUNE_CENTRAL_FREQ	OSC tune to Central frequency
OSC_FRC_TUNE_MINUS_0_375_PERCENT	OSC tune to Central frequency - 0.375
OSC_FRC_TUNE_MINUS_0_75_PERCENT	OSC tune to Central frequency - 0.75
OSC_FRC_TUNE_MINUS_1_125_PERCENT	OSC tune to Central frequency - 1.125
OSC_FRC_TUNE_MINUS_1_5_PERCENT	OSC tune to Central frequency - 1.5
OSC_FRC_TUNE_MINUS_1_875_PERCENT	OSC tune to Central frequency - 1.875
OSC_FRC_TUNE_MINUS_2_25_PERCENT	OSC tune to Central frequency - 2.25
OSC_FRC_TUNE_MINUS_2_625_PERCENT	OSC tune to Central frequency - 2.625
OSC_FRC_TUNE_MINUS_3_PERCENT	OSC tune to Central frequency - 3

Remarks

None.

5.6.14.6.13.13 OSC_GFX_CLOCK Enumeration**C**

```
typedef enum {
    OSC_GFX_CLOCK_96M,
    OSC_GFX_CLOCK_48M
} OSC_GFX_CLOCK;
```

Description

Graphics module clock values.

This enumeration lists the possible Graphics (display) module clock values.

Members

Members	Description
OSC_GFX_CLOCK_96M	Graphics module clock is 96 MHz
OSC_GFX_CLOCK_48M	Graphics module clock is 48 MHz

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.14 OSC_LFSR_TYPE Enumeration**C**

```
typedef enum {
    OSC_LFSR
} OSC_LFSR_TYPE;
```

Description

AUX PLL Input divider.

This enumeration lists the possible Auxiliary PLL input divider values.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.15 OSC_MODULE_ID Enumeration**C**

```
typedef enum {  
    OSC_ID_0  
} OSC_MODULE_ID;
```

Description

OSC module ID

This data type defines the possible instances of the OSC module.

Members

Members	Description
OSC_ID_0	first instance of the OSC

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.16 OSC_OPERATION_ON_WAIT Enumeration**C**

```
typedef enum {  
    OSC_ON_WAIT_IDLE,  
    OSC_ON_WAIT_SLEEP  
} OSC_OPERATION_ON_WAIT;
```

Description

Oscillator Operation on wait.

This enumeration lists the possible device status when a WAIT instruction is executed.

Members

Members	Description
OSC_ON_WAIT_IDLE	Idle on Wait Instruction
OSC_ON_WAIT_SLEEP	Idle on Sleep Instruction

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.17 OSC_PERIPHERAL_BUS Enumeration**C**

```
typedef enum {  
    OSC_PERIPHERAL_BUS_1,  
    OSC_PERIPHERAL_BUS_2,
```

```
OSC_PERIPHERAL_BUS_3,  
OSC_PERIPHERAL_BUS_4,  
OSC_PERIPHERAL_BUS_5,  
OSC_PERIPHERAL_BUS_6,  
OSC_PERIPHERAL_BUS_7,  
OSC_PERIPHERAL_BUS_8  
} OSC_PERIPHERAL_BUS;
```

Description

Peripheral Buses

This enumeration lists the possible peripheral buses available in the device

Members

Members	Description
OSC_PERIPHERAL_BUS_1	Peripheral bus 1
OSC_PERIPHERAL_BUS_2	Peripheral bus 2
OSC_PERIPHERAL_BUS_3	Peripheral bus 3
OSC_PERIPHERAL_BUS_4	Peripheral bus 4
OSC_PERIPHERAL_BUS_5	Peripheral bus 5
OSC_PERIPHERAL_BUS_6	Peripheral bus 6
OSC_PERIPHERAL_BUS_7	Peripheral bus 7
OSC_PERIPHERAL_BUS_8	Peripheral bus 8

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.18 OSC_PLL_SELECT Enumeration**C**

```
typedef enum {  
    OSC_PLL_SYSTEM,  
    OSC_PLL_USB  
} OSC_PLL_SELECT;
```

Description

Oscillator PLL selection.

This enumeration lists the available PLLs in the Oscillator module.

Members

Members	Description
OSC_PLL_SYSTEM	Select system PLL
OSC_PLL_USB	Select USB PLL

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.19 OSC_PLLAUX_CLOCK_SOURCE Enumeration**C**

```
typedef enum {  
    OSC_PLLAUX_CLOCK  
} OSC_PLLAUX_CLOCK_SOURCE;
```


Description

Auxiliary PLL Clock Source

This enumeration lists the possible Auxiliary PLL clock source.

Members

Members	Description
OSC_PLLAUX_CLOCK	PLL Auxiliary clock

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.20 OSC_REF_BASECLOCK Enumeration

C

```
typedef enum {  
    OSC_REF_BASECLOCK_SYSCLK,  
    OSC_REF_BASECLOCK_PBCLK,  
    OSC_REF_BASECLOCK_PRIMARY,  
    OSC_REF_BASECLOCK_FRC,  
    OSC_REF_BASECLOCK_LPRC,  
    OSC_REF_BASECLOCK_SOSC,  
    OSC_REF_BASECLOCK_USBCLK,  
    OSC_REF_BASECLOCK_SYSPLLOUT,  
    OSC_REF_BASECLOCK_EXT  
} OSC_REF_BASECLOCK;
```

Description

Base clock for reference oscillator.

This enumeration lists the possible base clock values for the reference oscillator.

Members

Members	Description
OSC_REF_BASECLOCK_SYSCLK	The base clock for reference clock is System clock
OSC_REF_BASECLOCK_PBCLK	The base clock for reference clock is Peripheral clock
OSC_REF_BASECLOCK_PRIMARY	The base clock for reference clock is Primary oscillator
OSC_REF_BASECLOCK_FRC	The base clock for reference clock is FRC oscillator
OSC_REF_BASECLOCK_LPRC	The base clock for reference clock is internal low power RC
OSC_REF_BASECLOCK_SOSC	The base clock for reference clock is Secondary oscillator
OSC_REF_BASECLOCK_USBCLK	The base clock for reference clock is USB clock
OSC_REF_BASECLOCK_SYSPLLOUT	The base clock for reference clock is System PLL output
OSC_REF_BASECLOCK_EXT	The base clock for reference clock is clock from external pin

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.21 OSC_REF_DIV Enumeration

C

```
typedef enum {  
    OSC_REF_DIV_1,  
    OSC_REF_DIV_2,  
    OSC_REF_DIV_4,  
}
```

```

OSC_REF_DIV_8,
OSC_REF_DIV_16,
OSC_REF_DIV_32,
OSC_REF_DIV_64,
OSC_REF_DIV_128,
OSC_REF_DIV_256,
OSC_REF_DIV_512,
OSC_REF_DIV_1024,
OSC_REF_DIV_2048,
OSC_REF_DIV_4096,
OSC_REF_DIV_8192,
OSC_REF_DIV_16384,
OSC_REF_DIV_32768
} OSC_REF_DIV;

```

Description

Reference Oscillator divisor.

This enumeration lists the possible reference oscillator divisor values.

Members

Members	Description
OSC_REF_DIV_1	Reference clock is same as base clock
OSC_REF_DIV_2	Reference clock is base clock divided by 2
OSC_REF_DIV_4	Reference clock is base clock divided by 4
OSC_REF_DIV_8	Reference clock is base clock divided by 8
OSC_REF_DIV_16	Reference clock is base clock divided by 16
OSC_REF_DIV_32	Reference clock is base clock divided by 32
OSC_REF_DIV_64	Reference clock is base clock divided by 64
OSC_REF_DIV_128	Reference clock is base clock divided by 128
OSC_REF_DIV_256	Reference clock is base clock divided by 256
OSC_REF_DIV_512	Reference clock is base clock divided by 512
OSC_REF_DIV_1024	Reference clock is base clock divided by 1024
OSC_REF_DIV_2048	Reference clock is base clock divided by 2048
OSC_REF_DIV_4096	Reference clock is base clock divided by 4096
OSC_REF_DIV_8192	Reference clock is base clock divided by 8192
OSC_REF_DIV_16384	Reference clock is base clock divided by 16384
OSC_REF_DIV_32768	Reference clock is base clock divided by 32768

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.22 OSC_REFERENCE Enumeration**C**

```

typedef enum {
    OSC_REFERENCE_1,
    OSC_REFERENCE_2,
    OSC_REFERENCE_3,
    OSC_REFERENCE_4
} OSC_REFERENCE;

```

Description

Reference Oscillators

This enumeration lists the possible reference oscillator instances available in the device

Members

Members	Description
OSC_REFERENCE_1	Reference Oscillator 1
OSC_REFERENCE_2	Reference Oscillator 2
OSC_REFERENCE_3	Reference Oscillator 3
OSC_REFERENCE_4	Reference Oscillator 4

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.23 OSC_SYS_CLOCK_DIV Enumeration

C

```
typedef enum {
    OSC_SYS_CLOCK_DIV_1,
    OSC_SYS_CLOCK_DIV_2,
    OSC_SYS_CLOCK_DIV_4,
    OSC_SYS_CLOCK_DIV_8
} OSC_SYS_CLOCK_DIV;
```

Description

System Clock selection.

This enumeration lists the possible system clock divisor.

Members

Members	Description
OSC_SYS_CLOCK_DIV_1	System clock divide by 1
OSC_SYS_CLOCK_DIV_2	System clock divide by 2
OSC_SYS_CLOCK_DIV_4	System clock divide by 4
OSC_SYS_CLOCK_DIV_8	System clock divide by 8

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.24 OSC_SYS_TYPE Enumeration

C

```
typedef enum {
    OSC_FRC,
    OSC_FRC_WITH_PLL,
    OSC_PRIMARY,
    OSC_PRIMARY_WITH_PLL,
    OSC_SECONDARY,
    OSC_LPRC,
    OSC_FRC_DIV_BY_16,
    OSC_FRC_BY_FRCDIV
} OSC_SYS_TYPE;
```

Description

Oscillator type.

This enumeration lists the possible oscillator type values.

Members

Members	Description
OSC_FRC	Fast RC Oscillator. This includes the 8MHz oscillator
OSC_FRC_WITH_PLL	Fast RC Oscillator with PLL
OSC_PRIMARY	Primary Oscillator
OSC_PRIMARY_WITH_PLL	Fast RC Oscillator with PLL and PostScaler. Includes the 8MHz oscillator
OSC_SECONDARY	Secondary Oscillator
OSC_LPRC	Low Power RC Oscillator
OSC_FRC_DIV_BY_16	Low-Power Fast RC Oscillator with PostScaler
OSC_FRC_BY_FRCDIV	Fast RC Oscillator divided by FRCDIV bits

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.25 OSC_SYSPLL_FREQ_RANGE Enumeration**C**

```
typedef enum {
    OSC_SYSPLL_FREQ_RANGE_BYPASS,
    OSC_SYSPLL_FREQ_RANGE_5M_TO_10M,
    OSC_SYSPLL_FREQ_RANGE_8M_TO_16M,
    OSC_SYSPLL_FREQ_RANGE_13M_TO_26M,
    OSC_SYSPLL_FREQ_RANGE_21M_TO_42M,
    OSC_SYSPLL_FREQ_RANGE_34M_TO_68M
} OSC_SYSPLL_FREQ_RANGE;
```

Description

System PLL Frequency Range

This enumeration lists the possible frequency range for PLL module available in the device

Members

Members	Description
OSC_SYSPLL_FREQ_RANGE_BYPASS	PLL freq range bypass
OSC_SYSPLL_FREQ_RANGE_5M_TO_10M	PLL frequency range is 5MHz to 10MHz
OSC_SYSPLL_FREQ_RANGE_8M_TO_16M	PLL frequency range is 8MHz to 16MHz
OSC_SYSPLL_FREQ_RANGE_13M_TO_26M	PLL frequency range is 13MHz to 26MHz
OSC_SYSPLL_FREQ_RANGE_21M_TO_42M	PLL frequency range is 21MHz to 42MHz
OSC_SYSPLL_FREQ_RANGE_34M_TO_68M	PLL frequency range is 34MHz to 68MHz

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.26 OSC_SYSPLL_IN_CLK_SOURCE Enumeration**C**

```
typedef enum {
    OSC_SYSPLL_IN_CLK_SOURCE_FRC,
    OSC_SYSPLL_IN_CLK_SOURCE_PRIMARY
} OSC_SYSPLL_IN_CLK_SOURCE;
```

Description

System PLL Input Clock Source

This enumeration lists the possible input clock source for PLL module available in the device

Members

Members	Description
OSC_SYSPLL_IN_CLK_SOURCE_FRC	FRC is input clock source for PLL module
OSC_SYSPLL_IN_CLK_SOURCE_PRIMARY	Primary clock is the input clock source for PLL module

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.27 OSC_SYSPLL_IN_DIV Enumeration**C**

```
typedef enum {  
    OSC_SYSPLL_IN_DIV_1,  
    OSC_SYSPLL_IN_DIV_2,  
    OSC_SYSPLL_IN_DIV_3,  
    OSC_SYSPLL_IN_DIV_4,  
    OSC_SYSPLL_IN_DIV_5,  
    OSC_SYSPLL_IN_DIV_6,  
    OSC_SYSPLL_IN_DIV_7,  
    OSC_SYSPLL_IN_DIV_8  
} OSC_SYSPLL_IN_DIV;
```

Description

PLL Input divider.

This enumeration lists the possible PLL input divider values.

Members

Members	Description
OSC_SYSPLL_IN_DIV_1	Divide by 1
OSC_SYSPLL_IN_DIV_2	Divide by 2
OSC_SYSPLL_IN_DIV_3	Divide by 4
OSC_SYSPLL_IN_DIV_4	Divide by 8
OSC_SYSPLL_IN_DIV_5	Divide by 16
OSC_SYSPLL_IN_DIV_6	Divide by 32
OSC_SYSPLL_IN_DIV_7	Divide by 64
OSC_SYSPLL_IN_DIV_8	Divide by 256

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.28 OSC_SYSPLL_OUT_DIV Enumeration**C**

```
typedef enum {  
    OSC_SYSPLL_OUT_DIV_1,  
    OSC_SYSPLL_OUT_DIV_2,  
    OSC_SYSPLL_OUT_DIV_4,  
    OSC_SYSPLL_OUT_DIV_8,  
}
```

```

    OSC_SYSPLL_OUT_DIV_16,
    OSC_SYSPLL_OUT_DIV_32,
    OSC_SYSPLL_OUT_DIV_64,
    OSC_SYSPLL_OUT_DIV_256
} OSC_SYSPLL_OUT_DIV;

```

Description

PLL Output divider.

This enumeration lists the possible PLL output divider values.

Members

Members	Description
OSC_SYSPLL_OUT_DIV_1	Divide by 1
OSC_SYSPLL_OUT_DIV_2	Divide by 2
OSC_SYSPLL_OUT_DIV_4	Divide by 4
OSC_SYSPLL_OUT_DIV_8	Divide by 8
OSC_SYSPLL_OUT_DIV_16	Divide by 16
OSC_SYSPLL_OUT_DIV_32	Divide by 32
OSC_SYSPLL_OUT_DIV_64	Divide by 64
OSC_SYSPLL_OUT_DIV_256	Divide by 256

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.13.29 OSC_TUNING_MODE Enumeration

C

```

typedef enum {
    OSC_TUNING_USING_NUMBER,
    OSC_TUNING_SEQ_DITHER,
    OSC_TUNING_PSEUDO_RANDOM
} OSC_TUNING_MODE;

```

Description

Oscillator Tuning modes enumeration

This enumeration lists the possible oscillator tuning modes, and is used by the [PLIB_OSC_FRCTuningModeSet](#) and [PLIB_OSC_FRCTuningModeGet](#) functions.

Members

Members	Description
OSC_TUNING_USING_NUMBER	FRC tuning is based on the number specified by the software. Set the number using PLIB_OSC_FRCTuningModeSet function
OSC_TUNING_SEQ_DITHER	FRC tuning is based on PWM
OSC_TUNING_PSEUDO_RANDOM	FRC tuning is Pseudo-Random number generation.

Remarks

None.

5.6.14.6.13.30 OSC_TUNING_SEQUENCERS Enumeration

C

```

typedef enum {

```

```

OSC_TUNING_SEQUENCER_1,
OSC_TUNING_SEQUENCER_2,
OSC_TUNING_SEQUENCER_3,
OSC_TUNING_SEQUENCER_4,
OSC_TUNING_SEQUENCER_5,
OSC_TUNING_SEQUENCER_6,
OSC_TUNING_SEQUENCER_7
} OSC_TUNING_SEQUENCERS;

```

Description

Oscillator sequencer entries

This enumeration lists the possible oscillator sequencers that are used by the [PLIB_OSC_FRCTuningSequenceValueSet](#) and [PLIB_OSC_FRCTuningSequenceValueGet](#) functions to tune the FRC Oscillator.

Members

Members	Description
OSC_TUNING_SEQUENCER_1	OSC tuning sequencer 1
OSC_TUNING_SEQUENCER_2	OSC tuning sequencer 2
OSC_TUNING_SEQUENCER_3	OSC tuning sequencer 3
OSC_TUNING_SEQUENCER_4	OSC tuning sequencer 4
OSC_TUNING_SEQUENCER_5	OSC tuning sequencer 5
OSC_TUNING_SEQUENCER_6	OSC tuning sequencer 6
OSC_TUNING_SEQUENCER_7	OSC tuning sequencer 7

Remarks

None.

5.6.14.6.13.31 OSC_USBCLOCK_SOURCE Enumeration

C

```

typedef enum {
    SYS_OSC_USBCLK_PRIMARY,
    SYS_OSC_USBCLK_FRC
} OSC_USBCLOCK_SOURCE;

```

Description

USB clock sources.

This enumeration lists the the possible USB clock sources.

Members

Members	Description
SYS_OSC_USBCLK_PRIMARY	USB clock source is primary oscillator
SYS_OSC_USBCLK_FRC	USB clock source is FRC oscillator

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

5.6.14.6.14 Feature Existence Functions

5.6.14.6.14.1 PLIB_OSC_ExistsClockFail Function

C

```
bool PLIB_OSC_ExistsClockFail(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ClockFail feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ClockHasFailed](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ClockFail feature is supported on the device
- false - If the ClockFail feature is not supported on the device

Remarks

None.

5.6.14.6.14.2 PLIB_OSC_ExistsFRCDivisor Function

C

```
bool PLIB_OSC_ExistsFRCDivisor(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the FRCDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_FRCDivisorSelect](#)
- [PLIB_OSC_FRCDivisorGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the FRCDivisor feature is supported on the device
- false - If the FRCDivisor feature is not supported on the device

Remarks

None.

5.6.14.6.14.3 PLIB_OSC_ExistsFRCTuning Function

C

```
bool PLIB_OSC_ExistsFRCTuning(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the FRCTuning feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_FRCTuningSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the FRCTuning feature is supported on the device
- false - If the FRCTuning feature is not supported on the device

Remarks

None.

5.6.14.6.14.4 PLIB_OSC_ExistsOnWaitAction Function

C

```
bool PLIB_OSC_ExistsOnWaitAction(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the OnWaitAction feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_OnWaitActionSet](#)
- [PLIB_OSC_OnWaitActionGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the OnWaitAction feature is supported on the device
- false - If the OnWaitAction feature is not supported on the device

Remarks

None.

5.6.14.6.14.5 PLIB_OSC_ExistsOscCurrentGet Function

C

```
bool PLIB_OSC_ExistsOscCurrentGet(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the OscCurrentGet feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_CurrentSysClockGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the OscCurrentGet feature is supported on the device
- false - If the OscCurrentGet feature is not supported on the device

Remarks

None.

5.6.14.6.14.6 PLIB_OSC_ExistsOscSelect Function

C

```
bool PLIB_OSC_ExistsOscSelect(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the OscSelect feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_SysClockSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the OscSelect feature is supported on the device
- false - If the OscSelect feature is not supported on the device

Remarks

None.

5.6.14.6.14.7 PLIB_OSC_ExistsOscSwitchInit Function

C

```
bool PLIB_OSC_ExistsOscSwitchInit(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the OscSwitchInit feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ClockSwitchingAbort](#)
- [PLIB_OSC_ClockSwitchingIsComplete](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the OscSwitchInit feature is supported on the device
- false - If the OscSwitchInit feature is not supported on the device

Remarks

None.

5.6.14.6.14.8 PLIB_OSC_ExistsPBClockDivisor Function

C

```
bool PLIB_OSC_ExistsPBClockDivisor(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PBClockDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_PBClockDivisorGet](#)
- [PLIB_OSC_PBClockDivisorSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PBClockDivisor feature is supported on the device
- false - If the PBClockDivisor feature is not supported on the device

Remarks

None.

5.6.14.6.14.9 PLIB_OSC_ExistsPBClockOutputEnable Function**C**

```
bool PLIB_OSC_ExistsPBClockOutputEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PBClockOutputEnable feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_PBClockOutputEnable](#)
- [PLIB_OSC_PBClockOutputDisable](#)
- [PLIB_OSC_PBClockOutputIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PBClockOutputEnable feature is supported on the device
- false - If the PBClockOutputEnable feature is not supported on the device

Remarks

None.

5.6.14.6.14.10 PLIB_OSC_ExistsPBClockReady Function**C**

```
bool PLIB_OSC_ExistsPBClockReady(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PBClockReady feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_PBClockDivisorIsReady](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PBClockReady feature is supported on the device
- false - If the PBClockReady feature is not supported on the device

Remarks

None.

5.6.14.6.14.11 PLIB_OSC_ExistsPLLClockLock Function**C**

```
bool PLIB_OSC_ExistsPLLClockLock(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PLLClockLock feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_PLLClockLock](#)
- [PLIB_OSC_PLLClockUnlock](#)
- [PLIB_OSC_PLLClockIsLocked](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PLLClockLock feature is supported on the device
- false - If the PLLClockLock feature is not supported on the device

Remarks

None.

5.6.14.6.14.12 PLIB_OSC_ExistsPLLLockStatus Function**C**

```
bool PLIB_OSC_ExistsPLLLockStatus(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PLLLockStatus feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_PLLIsLocked](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PLLLockStatus feature is supported on the device
- false - If the PLLLockStatus feature is not supported on the device

Remarks

None.

5.6.14.6.14.13 PLIB_OSC_ExistsReferenceOscBaseClock Function**C**

```
bool PLIB_OSC_ExistsReferenceOscBaseClock(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ReferenceOscBaseClock feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ReferenceOscBaseClockSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReferenceOscBaseClock feature is supported on the device
- false - If the ReferenceOscBaseClock feature is not supported on the device

Remarks

None.

5.6.14.6.14.14 PLIB_OSC_ExistsReferenceOscChange Function**C**

```
bool PLIB_OSC_ExistsReferenceOscChange(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ReferenceOscChange feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ReferenceOscSwitchIsComplete](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReferenceOscChange feature is supported on the device
- false - If the ReferenceOscChange feature is not supported on the device

Remarks

None.

5.6.14.6.14.15 PLIB_OSC_ExistsReferenceOscChangeActive Function**C**

```
bool PLIB_OSC_ExistsReferenceOscChangeActive(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ReferenceOscChangeActive feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ReferenceOscSourceChangelsActive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReferenceOscChangeActive feature is supported on the device
- false - If the ReferenceOscChangeActive feature is not supported on the device

Remarks

None.

5.6.14.6.14.16 PLIB_OSC_ExistsReferenceOscDivisor Function**C**

```
bool PLIB_OSC_ExistsReferenceOscDivisor(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ReferenceOscDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ReferenceOscDivisorValueSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReferenceOscDivisor feature is supported on the device
- false - If the ReferenceOscDivisor feature is not supported on the device

Remarks

None.

5.6.14.6.14.17 PLIB_OSC_ExistsReferenceOscEnable Function**C**

```
bool PLIB_OSC_ExistsReferenceOscEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ReferenceOscEnable feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ReferenceOscEnable](#)
- [PLIB_OSC_ReferenceOscDisable](#)
- [PLIB_OSC_ReferenceOscIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReferenceOscEnable feature is supported on the device
- false - If the ReferenceOscEnable feature is not supported on the device

Remarks

None.

5.6.14.6.14.18 PLIB_OSC_ExistsReferenceOscStopInIdleEnable Function**C**

```
bool PLIB_OSC_ExistsReferenceOscStopInIdleEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ReferenceOscStopInIdleEnable feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ReferenceOscStopInIdleEnable](#)

- [PLIB_OSC_ReferenceOscStopInIdleDisable](#)
- [PLIB_OSC_ReferenceOscStopInIdleIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReferenceOscStopInIdleEnable feature is supported on the device
- false - If the ReferenceOscStopInIdleEnable feature is not supported on the device

Remarks

None.

5.6.14.6.14.19 PLIB_OSC_ExistsReferenceOscStopInSleep Function**C**

```
bool PLIB_OSC_ExistsReferenceOscStopInSleep(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ReferenceOscStopInSleep feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ReferenceOscStopInSleepEnable](#)
- [PLIB_OSC_ReferenceOscStopInSleepDisable](#)
- [PLIB_OSC_ReferenceOscStopInSleepIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReferenceOscStopInSleep feature is supported on the device
- false - If the ReferenceOscStopInSleep feature is not supported on the device

Remarks

None.

5.6.14.6.14.20 PLIB_OSC_ExistsReferenceOscTrim Function**C**

```
bool PLIB_OSC_ExistsReferenceOscTrim(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ReferenceOscTrim feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ReferenceOscTrimSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReferenceOscTrim feature is supported on the device
- false - If the ReferenceOscTrim feature is not supported on the device

Remarks

None.

5.6.14.6.14.21 PLIB_OSC_ExistsReferenceOutputEnable Function**C**

```
bool PLIB_OSC_ExistsReferenceOutputEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the ReferenceOutputEnable feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_ReferenceOutputEnable](#)
- [PLIB_OSC_ReferenceOutputDisable](#)
- [PLIB_OSC_ReferenceOutputIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the ReferenceOutputEnable feature is supported on the device
- false - If the ReferenceOutputEnable feature is not supported on the device

Remarks

None.

5.6.14.6.14.22 PLIB_OSC_ExistsSecondaryEnable Function

C

```
bool PLIB_OSC_ExistsSecondaryEnable(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the SecondaryEnable feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_SecondaryEnable](#)
- [PLIB_OSC_SecondaryDisable](#)
- [PLIB_OSC_SecondaryIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SecondaryEnable feature is supported on the device
- false - If the SecondaryEnable feature is not supported on the device

Remarks

None.

5.6.14.6.14.23 PLIB_OSC_ExistsSecondaryReady Function

C

```
bool PLIB_OSC_ExistsSecondaryReady(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the SecondaryReady feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_SecondaryIsReady](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the SecondaryReady feature is supported on the device
- false - If the SecondaryReady feature is not supported on the device

Remarks

None.

5.6.14.6.14.24 PLIB_OSC_ExistsSysPLLFrequencyRange Function**C**

```
bool PLIB_OSC_ExistsSysPLLFrequencyRange(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PLLFrequencyRange feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_SysPLLFrequencyRangeSet](#)
- [PLIB_OSC_SysPLLFrequencyRangeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PLLFrequencyRange feature is supported on the device
- false - If the PLLFrequencyRange feature is not supported on the device

Remarks

None.

5.6.14.6.14.25 PLIB_OSC_ExistsSysPLLInputClockSource Function**C**

```
bool PLIB_OSC_ExistsSysPLLInputClockSource(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PLLInputClockSource feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_SysPLLInputClockSourceSet](#)
- [PLIB_OSC_SysPLLInputClockSourceGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PLLInputClockSource feature is supported on the device
- false - If the PLLInputClockSource feature is not supported on the device

Remarks

None.

5.6.14.6.14.26 PLIB_OSC_ExistsSysPLLInputDivisor Function**C**

```
bool PLIB_OSC_ExistsSysPLLInputDivisor(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PLLInputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_SysPLLInputDivisorSet](#)
- [PLIB_OSC_SysPLLInputDivisorGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PLLInputDivisor feature is supported on the device
- false - If the PLLInputDivisor feature is not supported on the device

Remarks

None.

5.6.14.6.14.27 PLIB_OSC_ExistsSysPLLMultiplier Function**C**

```
bool PLIB_OSC_ExistsSysPLLMultiplier(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PLLMultiplier feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_SysPLLMultiplierSelect](#)
- [PLIB_OSC_SysPLLMultiplierGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PLLMultiplier feature is supported on the device
- false - If the PLLMultiplier feature is not supported on the device

Remarks

None.

5.6.14.6.14.28 PLIB_OSC_ExistsSysPLLOutputDivisor Function**C**

```
bool PLIB_OSC_ExistsSysPLLOutputDivisor(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the PLLOutputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_SysPLLOutputDivisorSet](#)
- [PLIB_OSC_SysPLLOutputDivisorGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the PLLOutputDivisor feature is supported on the device
- false - If the PLLOutputDivisor feature is not supported on the device

Remarks

None.

5.6.14.6.14.29 PLIB_OSC_ExistsUsbClockSource Function**C**

```
bool PLIB_OSC_ExistsUsbClockSource(  
    OSC_MODULE_ID index  
);
```

Description

This function identifies whether the UsbClockSource feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- [PLIB_OSC_UsbClockSourceSelect](#)
- [PLIB_OSC_UsbClockSourceGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - If the UsbClockSource feature is supported on the device
- false - If the UsbClockSource feature is not supported on the device

Remarks

None.

5.6.14.7 Files

Files

Name	Description
plib_osc.h	Defines the Oscillator (OSC) Peripheral Library interface.

Description

5.6.14.7.1 plib_osc.h

Oscillator Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Oscillator (OSC) Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Oscillator module.

Functions

	Name	Description
≡	PLIB_OSC_AuxClockSourceGet	Gets the clock source for the auxiliary clock.
≡	PLIB_OSC_AuxClockSourceSet	Sets the clock source for the auxiliary clock divisor.
≡	PLIB_OSC_AuxModeGet	Gets the selected Auxiliary Oscillator mode.
≡	PLIB_OSC_AuxModeSelect	Selects the Auxiliary Oscillator mode.
≡	PLIB_OSC_ClockHasFailed	Returns 'true' if the clock fails.
≡	PLIB_OSC_ClockSwitchingAbort	Aborts an oscillator switch.
≡	PLIB_OSC_ClockSwitchingIsComplete	Gets the oscillator switch progress status.
≡	PLIB_OSC_CurrentSysClockGet	Gets the current oscillator selected.
≡	PLIB_OSC_DozeModeDisable	Disables Doze mode.
≡	PLIB_OSC_DozeModeEnable	Enables Doze mode.
≡	PLIB_OSC_DozeRatioGet	Gets the ratio of the processor clock to the peripheral clock.
≡	PLIB_OSC_DozeRatioSelect	Selects the Doze ratio of the processor clock to the peripheral clock.
≡	PLIB_OSC_DozeRecoverOnInterruptDisable	Disables the recovery from Doze mode upon an interrupt.
≡	PLIB_OSC_DozeRecoverOnInterruptEnable	Enables recovery from Doze mode upon an interrupt.

	<u>PLIB_OSC_DozeRecoverOnInterruptIsEnabled</u>	Returns 'true' if recover from Doze mode on interrupt is enabled.
	<u>PLIB_OSC_ExistsClockFail</u>	Identifies whether the ClockFail feature exists on the OSC module
	<u>PLIB_OSC_ExistsFRCDivisor</u>	Identifies whether the FRCDivisor feature exists on the OSC module
	<u>PLIB_OSC_ExistsFRCTuning</u>	Identifies whether the FRCTuning feature exists on the OSC module
	<u>PLIB_OSC_ExistsOnWaitAction</u>	Identifies whether the OnWaitAction feature exists on the OSC module
	<u>PLIB_OSC_ExistsOscCurrentGet</u>	Identifies whether the OscCurrentGet feature exists on the OSC module
	<u>PLIB_OSC_ExistsOscSelect</u>	Identifies whether the OscSelect feature exists on the OSC module
	<u>PLIB_OSC_ExistsOscSwitchInit</u>	Identifies whether the OscSwitchInit feature exists on the OSC module
	<u>PLIB_OSC_ExistsPBClockDivisor</u>	Identifies whether the PBClockDivisor feature exists on the OSC module
	<u>PLIB_OSC_ExistsPBClockOutputEnable</u>	Identifies whether the PBClockOutputEnable feature exists on the OSC module
	<u>PLIB_OSC_ExistsPBClockReady</u>	Identifies whether the PBClockReady feature exists on the OSC module
	<u>PLIB_OSC_ExistsPLLClockLock</u>	Identifies whether the PLLClockLock feature exists on the OSC module
	<u>PLIB_OSC_ExistsPLLLockStatus</u>	Identifies whether the PLLLockStatus feature exists on the OSC module
	<u>PLIB_OSC_ExistsReferenceOscBaseClock</u>	Identifies whether the ReferenceOscBaseClock feature exists on the OSC module
	<u>PLIB_OSC_ExistsReferenceOscChange</u>	Identifies whether the ReferenceOscChange feature exists on the OSC module
	<u>PLIB_OSC_ExistsReferenceOscChangeActive</u>	Identifies whether the ReferenceOscChangeActive feature exists on the OSC module
	<u>PLIB_OSC_ExistsReferenceOscDivisor</u>	Identifies whether the ReferenceOscDivisor feature exists on the OSC module
	<u>PLIB_OSC_ExistsReferenceOscEnable</u>	Identifies whether the ReferenceOscEnable feature exists on the OSC module
	<u>PLIB_OSC_ExistsReferenceOscStopInIdleEnable</u>	Identifies whether the ReferenceOscStopInIdleEnable feature exists on the OSC module
	<u>PLIB_OSC_ExistsReferenceOscStopInSleep</u>	Identifies whether the ReferenceOscStopInSleep feature exists on the OSC module
	<u>PLIB_OSC_ExistsReferenceOscTrim</u>	Identifies whether the ReferenceOscTrim feature exists on the OSC module
	<u>PLIB_OSC_ExistsReferenceOutputEnable</u>	Identifies whether the ReferenceOutputEnable feature exists on the OSC module
	<u>PLIB_OSC_ExistsSecondaryEnable</u>	Identifies whether the SecondaryEnable feature exists on the OSC module
	<u>PLIB_OSC_ExistsSecondaryReady</u>	Identifies whether the SecondaryReady feature exists on the OSC module
	<u>PLIB_OSC_ExistsSysPLLFrequencyRange</u>	Identifies whether the PLLFrequencyRange feature exists on the OSC module
	<u>PLIB_OSC_ExistsSysPLLInputClockSource</u>	Identifies whether the PLLInputClockSource feature exists on the OSC module

	<u>PLIB_OSC_ExistsSysPLLInputDivisor</u>	Identifies whether the PLLInputDivisor feature exists on the OSC module
	<u>PLIB_OSC_ExistsSysPLLMultiplier</u>	Identifies whether the PLLMultiplier feature exists on the OSC module
	<u>PLIB_OSC_ExistsSysPLLOutputDivisor</u>	Identifies whether the PLLOutputDivisor feature exists on the OSC module
	<u>PLIB_OSC_ExistsUsbClockSource</u>	Identifies whether the UsbClockSource feature exists on the OSC module
	<u>PLIB_OSC_FRCDitherEnable</u>	Enables the FRC Oscillator clock dithering.
	<u>PLIB_OSC_FRCDivisorGet</u>	Gets the FRC clock divisor.
	<u>PLIB_OSC_FRCDivisorSelect</u>	Sets the FRC clock divisor to the specified value.
	<u>PLIB_OSC_FRCTuningModeGet</u>	Gets the FRC oscillator tuning mode.
	<u>PLIB_OSC_FRCTuningModeSet</u>	Selects the FRC oscillator tuning mode.
	<u>PLIB_OSC_FRCTuningSelect</u>	Sets the FRC tuning value.
	<u>PLIB_OSC_FRCTuningSequenceValueGet</u>	Gets the value set in the FRC tune sequencer.
	<u>PLIB_OSC_FRCTuningSequenceValueSet</u>	Sets the value in the FRC tune sequencer.
	<u>PLIB_OSC_GraphicsClockDivisorGet</u>	Gets the Graphics module clock divisor.
	<u>PLIB_OSC_GraphicsClockDivisorSet</u>	Sets the Graphics module clock divisor.
	<u>PLIB_OSC_GraphicsClockSourceGet</u>	Gets the Graphics Controller module clock frequency value.
	<u>PLIB_OSC_GraphicsClockSourceSelect</u>	Sets the Graphics Controller module clock.
	<u>PLIB_OSC_LinearFeedbackShiftRegGet</u>	Gets the value from linear feedback shift register.
	<u>PLIB_OSC_LinearFeedbackShiftRegSet</u>	Sets the linear feedback shift register.
	<u>PLIB_OSC_OnWaitActionGet</u>	Gets the configured operation to be performed when a WAIT instruction is executed.
	<u>PLIB_OSC_OnWaitActionSet</u>	Selects the operation to be performed when a WAIT instruction is executed.
	<u>PLIB_OSC_PBClockDivisorGet</u>	Gets the peripheral bus clock divisor.
	<u>PLIB_OSC_PBClockDivisorIsReady</u>	Checks whether the peripheral bus clock divisor is ready to be written.
	<u>PLIB_OSC_PBClockDivisorSet</u>	Sets the peripheral bus clock divisor to the specified value.
	<u>PLIB_OSC_PBOutputClockDisable</u>	Disables the peripheral bus output clock
	<u>PLIB_OSC_PBOutputClockEnable</u>	Enables the peripheral bus output clock
	<u>PLIB_OSC_PBOutputClockIsEnabled</u>	Checks whether the peripheral bus clock output is enabled or not
	<u>PLIB_OSC_PLLAuxClockSourceGet</u>	Returns the clock source selected for the Auxiliary PLL.
	<u>PLIB_OSC_PLLAuxClockSourceSelect</u>	Sets the Auxiliary PLL clock source.
	<u>PLIB_OSC_PLLAuxInputDivisorGet</u>	Gets the input divisor for the Auxiliary PLL.
	<u>PLIB_OSC_PLLAuxInputDivisorSelect</u>	Sets the input divisor for the Auxiliary PLL to the specified value.
	<u>PLIB_OSC_PLLAuxIsLocked</u>	Returns whether the Auxiliary PLL is locked.
	<u>PLIB_OSC_PLLAuxMultiplierGet</u>	Gets the Auxiliary PLL multiplier.
	<u>PLIB_OSC_PLLAuxMultiplierSelect</u>	Sets the Auxiliary PLL multiplier to the specified value.
	<u>PLIB_OSC_PLLAuxOutputDivisorGet</u>	Gets the output divisor for the Auxiliary PLL.
	<u>PLIB_OSC_PLLAuxOutputDivisorSet</u>	Sets the output divisor for the Auxiliary PLL to the specified value.
	<u>PLIB_OSC_PLLClockIsLocked</u>	Gets the lock status for the clock and PLL selections.
	<u>PLIB_OSC_PLLClockLock</u>	Locks the clock and PLL selections.
	<u>PLIB_OSC_PLLClockUnlock</u>	Unlocks the clock and PLL selections.

	PLIB_OSC_PLLODisable	Disables the specified PLL.
	PLIB_OSC_PLLEnable	Enables the specified PLL.
	PLIB_OSC_PLLOIsEnabled	Returns 'true' if the specified PLL is enabled.
	PLIB_OSC_PLLOIsLocked	Returns 'true' if the selected PLL is locked.
	PLIB_OSC_PrimaryOscInSleepModeDisable	The Primary Oscillator is disabled during Sleep mode.
	PLIB_OSC_PrimaryOscInSleepModeEnable	The Primary Oscillator continues to operate during Sleep mode.
	PLIB_OSC_PrimaryOscInSleepModeIsEnabled	Returns 'true' if the Primary Oscillator is disabled during Sleep mode.
	PLIB_OSC_ReferenceOscBaseClockSelect	Sets the base clock for the reference oscillator.
	PLIB_OSC_ReferenceOscDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOscDivisorValueSet	Selects the reference oscillator divisor value.
	PLIB_OSC_ReferenceOscEnable	Enables the reference oscillator.
	PLIB_OSC_ReferenceOscIsEnabled	Gets the enable status of the reference oscillator output.
	PLIB_OSC_ReferenceOscSourceChangeIsActive	Returns 'true' if a reference oscillator source change request is active.
	PLIB_OSC_ReferenceOscStopInIdleDisable	Enables the reference oscillator in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleEnable	Configures the reference oscillator to stop operating in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleIsEnabled	Returns 'true' if the reference oscillator is disabled in Idle mode.
	PLIB_OSC_ReferenceOscStopInSleepDisable	Enables the reference oscillator in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepEnable	Configures the reference oscillator to stop operating in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepIsEnabled	Returns 'true' if the reference oscillator is disabled in Sleep mode.
	PLIB_OSC_ReferenceOscSwitchIsComplete	Returns 'true' if the reference oscillator base clock switching is complete.
	PLIB_OSC_ReferenceOscTrimSet	Sets the reference oscillator divisor trim value.
	PLIB_OSC_ReferenceOutputDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOutputEnable	Enables the reference oscillator output.
	PLIB_OSC_ReferenceOutputIsEnabled	Returns 'true' if the reference oscillator output is enabled.
	PLIB_OSC_SecondaryDisable	Disables the Secondary Oscillator.
	PLIB_OSC_SecondaryEnable	Enables the Secondary Oscillator.
	PLIB_OSC_SecondaryIsEnabled	Returns 'true' if the Secondary Oscillator is enabled.
	PLIB_OSC_SecondaryIsReady	Returns 'true' if the Secondary Oscillator is ready.
	PLIB_OSC_StartupTimerHasExpired	Returns 'true' if the oscillator start-up time-out timer has expired.
	PLIB_OSC_SysClockSelect	Selects the new oscillator.
	PLIB_OSC_SysPLLFrequencyRangeGet	Returns the frequency range for PLL
	PLIB_OSC_SysPLLFrequencyRangeSet	Sets the frequency range for PLL
	PLIB_OSC_SysPLLInputClockSourceGet	Returns the input clock source for the PLL module
	PLIB_OSC_SysPLLInputClockSourceSet	Sets the input clock source for the PLL module
	PLIB_OSC_SysPLLInputDivisorGet	Gets the input divisor for the PLL.
	PLIB_OSC_SysPLLInputDivisorSet	Sets the input divider for the PLL to the specified value.
	PLIB_OSC_SysPLLMultiplierGet	Gets the PLL multiplier.
	PLIB_OSC_SysPLLMultiplierSelect	Sets the PLL multiplier to the specified value.
	PLIB_OSC_SysPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_SysPLLOutputDivisorSet	Sets the output divider for the PLL to the specified value.
	PLIB_OSC_UsbClockSourceGet	Gets the USB module clock source.

	PLIB_OSC_UsbClockSourceSelect	Sets the USB module clock source.

File Name

plib_osc.h

Company

Microchip Technology Inc.

5.6.15 Prefetch Cache Peripheral Library

5.6.15.1 Introduction

Prefetch Cache Module Peripheral Library

for

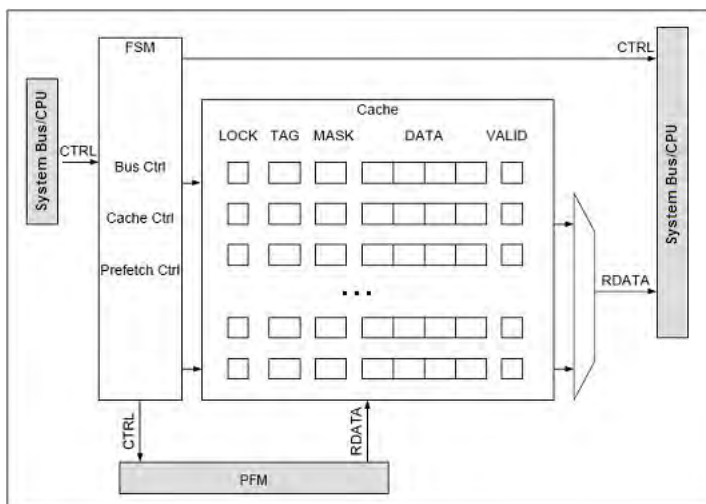
Microchip Microcontrollers

This library provides a low-level abstraction of the Prefetch Cache module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description**Prefetch Cache Module Overview**

The Prefetch Cache module increases system performance for most applications by:

- Predictive prefetching of instructions ahead of the current program counter, hiding the access time of the Flash memory
- Instruction and data caching



The program Flash memory (PFM) contains a Prefetch module and a number of cache lines, each containing four words. Some of the cache lines may be allocated to data or peripherals. Each cache line has an associated tag. The tag contains the address of the data in the cache line, and a number of status bits. The number of cache lines and number and type of status bits may vary between devices. Refer to your specific device data sheet for details.

5.6.15.2 Release Notes

MPLAB Harmony Version: v0.70b **Prefetch Cache Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.15.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.15.4 Using the Library

This topic describes the basic architecture of the Prefetch Cache Peripheral Library and provides information and examples on how to use it.

Interface Header File: peripheral.h

The interface to the Prefetch Cache Peripheral Library is defined in the "[plib_pcache.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the Prefetch Cache Peripheral Library must include "peripheral.h".

Library File:

The Prefetch Cache Peripheral Library is part of the processor-specific peripheral library archive (.a) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

5.6.15.4.1 Library Architecture

The cache consists of two arrays: tag and data. The data array contains a number of lines of program instructions or data words.

The word size is the same as the processor data width.

The tag array contains a number of corresponding fields, each containing:

- Mask – address mask value
- Tag – tag address to match against
- Valid – indicates whether the data in the cache line is valid or not
- Lock – indicates the line is locked to hold its contents
- Type – instruction, data or peripheral

Note: Not all Tag fields are available on all devices. The number of cache lines may vary between devices. Refer to the specific device data sheet for details.

Example Cache Line

Tag Array					Data Array			
Mask	Tag	Valid	Lock	Type	Word 3	Word 2	Word 1	Word 0

This library allows the programmer to perform various operations on the cache tag and data arrays:

- Enable/Disable
- Set the Wait state
- Read, write, lock, invalidate

Note: Not all operations are available on all devices. Refer to the specific device data sheet for details.

5.6.15.4.2 Hardware Abstraction Model

This library provides a low-level abstraction of the Prefetch Cache module on Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model

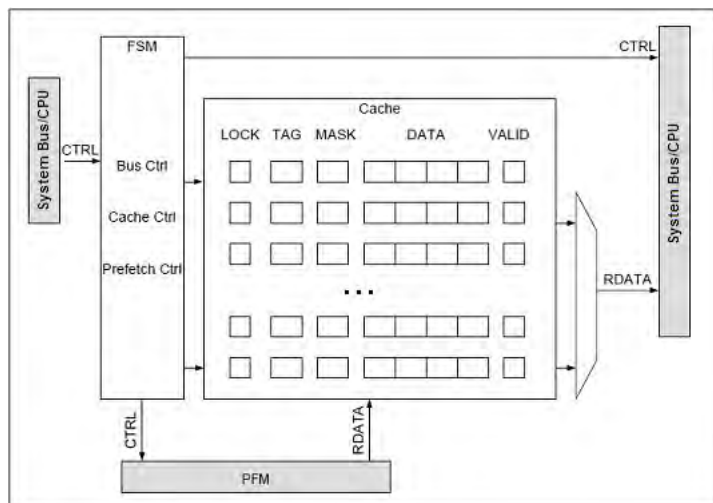
The Prefetch Cache module is a performance enhancing module included in some microcontrollers. When running at high clock rates, wait states must be inserted into program Flash memory (PFM) read transactions to meet the access time of the PFM. Wait states can be hidden to the core by prefetching and storing instructions in a temporary holding area that the CPU can access quickly.

There are two main functions that the Prefetch Cache module performs: Caching instructions when they are accessed, and prefetching instructions from the PFM before they are needed.

The cache holds a subset of the cacheable memory in temporary holding spaces known as cache lines. Each cache line has a tag describing what it is currently holding, and the address where it is mapped. Normally, the cache lines just hold a copy of what is currently in memory to make data available to the CPU without wait states.

CPU-requested data may or may not be in the cache. A cache-miss occurs if the CPU requests cacheable data that is not in the cache. In this case, a read is performed to the PFM at the correct address, the data is supplied to the cache and to the CPU. A cache-hit occurs if the cache contains the data that the CPU requests. In the case of a cache-hit, data is supplied to the CPU without wait states.

The second main function of the Prefetch Cache module is to prefetch CPU instructions. The module calculates the address of the next cache line and performs a read of the PFM to get the next cache line. This line is placed into a prefetch cache buffer in anticipation of executing straight-line code.



Cache Control

This library provides a set of functions to control the behavior of the overall Cache Module. Those functions include:

- Setting or reading the number of clock wait cycles
- Setting or reading the number of cache lines allocated to data
- Controlling which cache lines are invalidated on a PFM write cycle

Cache Line Operations

This library provides a set of functions to read, write and control the behavior of individual Cache Lines. Operations on individual Cache Lines are performed using a two-step process:

1. Select the individual cache line with [PLIB_PCACHE_CacheLineSelect](#).
2. Perform the required cache line operation using any of the routines in the "**Cache Line Operations**" section. The operations available are:
 - Set the cache line to data or instruction type
 - Mark the cache line as valid or invalid
 - Lock or unlock the cache line
 - Read or write the Flash type (boot or program) associated with the cache line
 - Read or write the address associated with the cache line
 - Read or write the mask field to force a match between the cache line and multiple physical addresses
 - Read or write any individual word in the cache line data array

Once the operation is performed, the cache line remains selected until [PLIB_PCACHE_CacheLineDeselect](#) is performed. Only one cache line can be selected at a time.

Cache Status

The cache maintains a count of hits and misses for profiling purposes. The cache hit counter increments each time the processor issues an instruction fetch or load that hits the prefetch cache from a cacheable region. Similarly, the cache miss counter increments each time the processor issues an instruction fetch or load that misses the prefetch cache from a cacheable region. Accesses to non-cacheable regions do not affect the counters. These values can be read or written using the Cache Status routines.

The cache uses a Least Recently Used (LRU) algorithm to replace cache lines on a miss. The specific algorithm may vary between devices. In some devices, the state of the LRU algorithm encoding bits can be read with

[PLIB_PCACHE_LeastRecentlyUsedStateRead](#).

Prefetch Control

Predictive prefetch is controlled by the Prefetch Control routines. Devices with no L1 CPU cache can prefetch from cacheable or non-cacheable regions or both. Devices with L1 CPU cache can prefetch from any address, CPU instructions or CPU instructions and data. Predictive prefetch can be disabled for either type of device.

Prefetch Status

The Prefetch Cache module maintains a count of prefetch aborts for profiling purposes. The prefetch abort counter is incremented each time a prefetch is aborted due to a non-sequential instruction fetch, load or store. This value can be read or written using the Prefetch Status routines.

Flash ECC

On some devices, the Flash ECC module has the ability to detect single and double bit errors on PFM reads. When a single bit error is detected, the error is corrected and the Single Error Correction (SEC) counter is decremented. When the SEC counter reaches zero, an interrupt is generated to the CPU. The user has the ability to enable or disable the SEC interrupt. It is disabled by default. The user may set the SEC counter value at any time. The maximum counter value may be device-specific and is defined for each processor by the [PLIB_PCACHE_MAX_SEC_COUNT](#) constant.

When a double bit error is detected, the device will set the DED status to true and generate a bus exception to the initiator.

5.6.15.4.3 How the Library Works

This library can be used to optimize the performance of the processor while executing out of cacheable program Flash memory. This is done by implementing the following:

- Instruction caching
- Data caching
- Cache line control
- Predictive prefetching
- ECC detection

5.6.15.4.3.1 Cache Control Operations

Cache Wait States

When operating at high clock rates, the Prefetch Cache module must insert wait states into Program Flash Memory (PFM) read operations to meet PFM access time requirements. One Wait state is equivalent to one clock cycle delay. At reset, the Prefetch Cache Module is set to the maximum number of wait states for the device. The PFM read time can be optimized for the device by selecting the minimum number of wait states for the clock rate used. The number of wait states for a given clock frequency may vary between devices.

Note: Refer to the specific device data sheet for details on how to calculate the optimum number of wait states for your device.

Example: Setting the Number of PFM wait states based on the System Clock

This example assumes one wait state for every 20 MHz of system clock.

```
if (sysclk < 20000000)
{
    PLIB_PCACHE_WaitStateSet(0);
}
else if (sysclk < 40000000)
{
    PLIB_PCACHE_WaitStateSet(1);
}
```



```
}
...
```

Data Caching

Some Prefetch Cache modules allow the user to allocate cache lines to data. This is to provide improved CPU access to constant data stored in PFM. By default, data caching is disabled. The Prefetch Cache Peripheral Library allows the user to set the number of cache lines allocated to data. The number of lines available may vary among devices. The available settings are provided by the device-dependent [PLIB_PCACHE_DATA_ENABLE](#) enumeration.

Example: Allocating two Cache Lines to Data

```
PLIB_PCACHE_DataCacheEnableSet(PLIB_PCACHE_DATA_2LINE);
```

Cache Line Invalidating on PFM Write

It's not possible to execute out of cache while programming the PFM. The PFM controller stalls the cache during the programming sequence. Therefore, user code that initiates a programming sequence should not be located in a cacheable address region.

During program operation, the Prefetch Cache is flushed by invalidating some or all of the cache lines. To invalidate all of the cache lines on a PFM write operation, the function [PLIB_PCACHE_InvalidateOnPFMProgramAll](#) should be called prior to the PFM write. This will invalidate and unlock every cache line during the write operation. The cache tags and masks are also cleared for all lines.

If [PLIB_PCACHE_InvalidateOnPFMProgramUnlocked](#) is called before the PFM write, only lines that are not locked will be forced invalid.

By default, only unlocked cache lines are invalidated during a PFM write.

5.6.15.4.3.2 Cache Line Operations

Performing a cache line operation is a two-step process. Before calling a Cache Line operation function, the cache line must be selected and write-enabled. Once selected and write-enabled, any number of cache line operations may be performed on the selected line until the line is cleared or a different line is selected and write-enabled.

Cache lines are selected and write-enabled using [PLIB_PCACHE_CacheLineSelect](#). Cache lines are write-disabled using [PLIB_PCACHE_CacheLineDeselect](#).

The following example shows operations on two different cache lines. The code comments describe each operation.

Example: Cache Line Operations

```
uint32_t cacheLine[4] = {0x01234567, 0x12345678, 0x23456789, 0x34567890};

//Select Cache Line 4
PLIB_PCACHE_CacheLineSelect(4);

//Invalidate Cache Line 4
PLIB_PCACHE_CacheLineInvalid();

//Select Cache Line 6
PLIB_PCACHE_CacheLineSelect(6);

//Set Cache Line 6 to Data Type
PLIB_PCACHE_CacheLineData();

//Write Data to Cache Line 6
for (i=0; i<PLIB_PCACHE_NUM_WORDS_PER_LINE; i++)
{
    PLIB_PCACHE_WordWrite(i, cacheLine[i]);
}
```



```
//Lock Cache Line 6
PLIB_PCACHE_CacheLineLock();

//Disable Writes to Cache Line 6
PLIB_PCACHE_CacheLineDeselect();
```

5.6.15.4.3.3 Cache Status Operations

Cache Statistics Counters

It is often useful to gather statistics related to cache utilization for performance or profiling reasons. Some devices with a Prefetch Cache module maintain cache hit and miss counters for this purpose. This library provides routines to read and write the cache hit and miss counters.

The cache hit counter increments each time the processor issues an instruction fetch or load that hits the prefetch cache from a cacheable region. Non-cacheable accesses do not modify this value. This value can be read with [PLIB_PCACHE_CacheHitRead](#).

Similarly, the cache miss counter increments each time the processor issues an instruction fetch or load that misses the prefetch cache from a cacheable region. This value can be read with [PLIB_PCACHE_CacheMissRead](#).

Cache Replacement State

The Prefetch Cache module uses a pseudo-Least Recently Used (LRU) algorithm to replace cache lines. The specific algorithm may vary among devices.

Note: Refer to the specific device data sheet for details on a specific device.

Some devices maintain the state of the LRU algorithm for use by software. The state of the LRU algorithm can be read with [PLIB_PCACHE_LeastRecentlyUsedStateRead](#).

5.6.15.4.3.4 Prefetch Control Operations

The Prefetch Cache module is configured differently for different CPU configurations. For devices with a CPU L1 cache, the Prefetch Cache module holds one line of CPU instructions, one line of CPU data, and two lines of peripheral data. For devices without a CPU L1 cache, the Prefetch Cache module holds up to 16 lines of instructions and up to four lines of data. Control of either configuration is done using the [PLIB_PCACHE_PrefetchEnableSet](#) function.

Note: Refer to the specific device data sheet for the Prefetch Cache module configuration for a specific device.

For the CPU L1 configuration, the Prefetch Cache module can be configured to enable any address, CPU instructions and data, or CPU instructions only. For devices without a CPU L1 cache, predictive prefetch can be enabled for cacheable regions, uncachable regions, or both. In either configuration, predictive prefetch may be disabled entirely.

The [PLIB_PCACHE_PrefetchEnableSet](#) and [PLIB_PCACHE_PrefetchEnableGet](#) functions take or return the enumerated value [PLIB_PCACHE_PREFETCH_ENABLE](#). This enumeration will be defined differently for different CPU configurations.

Example: Enable Prefetch for Non-Cacheable Regions Only (no L1 configuration)

```
PLIB_PCACHE_PrefetchEnableSet(PLIB_PCACHE_PREFETCH_ENABLE_NONCACHED_REGIONS);
```

Example: Enable Prefetch for CPU Instructions and CPU Data (L1 configuration)

```
PLIB_PCACHE_PrefetchEnableSet(PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST_DATA);
```

5.6.15.4.3.5 Prefetch Status Operations

Prefetch Abort Statistics Counter

Much like the cache hit/miss counters, the Prefetch Abort module maintains a prefetch abort counter. This counter is incremented each time an automatic prefetch is aborted due to a non-sequential instruction fetch, load or store. This value can

be read using the [PLIB_PCACHE_PrefetchAbortRead](#) function.

5.6.15.4.3.6 Flash ECC Operations

The Prefetch Cache module may handle and report information about two error types: ECC Double-bit Error Detected (DED) and ECC Single-bit Error Corrected (SEC).

A read from the Flash memory that results in a program Flash memory ECC DED causes the Prefetch Cache module to return a bus exception to the initiator. An ECC DED will also set the DED status bit. The state of the DED status bit may be read using the [PLIB_PCACHE_FlashDEDStatusGet](#) function.

A single-bit error read from the Flash memory is not a critical error, and as such is reported as an interrupt. The user has the option to enable or disable the SEC interrupt, and to set the number of SEC events that must occur before an SEC interrupt is generated. An SEC interrupt is generated when all of the following conditions are met:

- The Flash SEC counter is zero
- The Flash SEC interrupt is enabled
- An SEC event occurs

The Flash SEC counter is decremented once each time an SEC event occurs, and holds at zero. When a Flash SEC interrupt is triggered, the Flash SEC status bit is set. Once an SEC interrupt occurs, the Flash SEC status must be cleared by software and the Flash SEC counter must be reloaded by software.

Example: Set Flash SEC to Interrupt after three SEC events

```
PLIB_PCACHE_FlashSECStatusClear();
PLIB_PCACHE_FlashSECCountSet(3);
PLIB_PCACHE_FlashSECIntEnable();
```

It is possible to generate an SEC interrupt for testing purposes by enabling the Flash SEC interrupt and setting the Flash SEC status while the Flash SEC counter is zero.

Example: Generate an SEC Interrupt

```
PLIB_PCACHE_FlashSECStatusClear();
PLIB_PCACHE_FlashSECCountSet(0);
PLIB_PCACHE_FlashSECIntEnable();
PLIB_PCACHE_FlashSECStatusSet();
```

5.6.15.5 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.15.6 Library Interface

Data Types and Constants

	Name	Description
	PLIB_PCACHE_MAX_SEC_COUNT	Defines the maximum value for single bit error counter.
	PLIB_PCACHE_NUM_LINES	Defines the number of Prefetch Cache Lines available
	PLIB_PCACHE_NUM_WORDS_PER_LINE	Defines the number of Words per Prefetch Cache Line
	PCACHE_MODULE_ID	Lists the possible Module IDs for the pcache.
	PLIB_PCACHE_DATA_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache
	PLIB_PCACHE_PREFETCH_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache

Cache Control Functions

	Name	Description
≡	PLIB_PCACHE_DataCacheEnableGet	Gets the data cache enable bits.
≡	PLIB_PCACHE_DataCacheEnableSet	Sets the data cache enable bits.
≡	PLIB_PCACHE_InvalidateOnPFMProgramAll	Sets prefetch cache to invalidate all data and instruction lines on a PFM program cycle.
≡	PLIB_PCACHE_InvalidateOnPFMProgramUnlocked	Sets prefetch cache to invalidate all unlocked data and instruction lines on a PFM program cycle.
≡	PLIB_PCACHE_WaitStateGet	Gets the prefetch cache access time.
≡	PLIB_PCACHE_WaitStateSet	Sets the prefetch cache access time.

Cache Line Functions

	Name	Description
≡	PLIB_PCACHE_CacheLineAddrGet	Gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_CacheLineAddrSet	Sets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_CacheLineData	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to data type.
≡	PLIB_PCACHE_CacheLineDeselect	Disables write access to the cache line specified by cache_line.
≡	PLIB_PCACHE_CacheLineFlashTypeBoot	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in boot Flash.
≡	PLIB_PCACHE_CacheLineFlashTypeInst	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in instruction Flash.
≡	PLIB_PCACHE_CacheLineFlashTypeIsInst	Returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing in instruction Flash.
≡	PLIB_PCACHE_CacheLineInst	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.
≡	PLIB_PCACHE_CacheLineInvalid	Invalidates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_CacheLineIsInst	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.
≡	PLIB_PCACHE_CacheLineIsLocked	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.
≡	PLIB_PCACHE_CacheLineIsValid	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.
≡	PLIB_PCACHE_CacheLineLock	Locks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_CacheLineMaskGet	Returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_CacheLineMaskSet	Sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_CacheLineSelect	Enables write access to the cache line specified by cache_line.
≡	PLIB_PCACHE_CacheLineUnlock	Unlocks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_CacheLineValid	Validates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_WordRead	Reads the word specified by word from cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .

	PLIB_PCACHE_WordWrite	Writes the word (specified by word parameter) with data (specified by data parameter) to cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .

Cache Status Functions

	Name	Description
	PLIB_PCACHE_CacheHitRead	Reads the number of cache hits.
	PLIB_PCACHE_CacheHitWrite	Sets the number of cache hits.
	PLIB_PCACHE_CacheMissRead	Reads the number of cache misses.
	PLIB_PCACHE_CacheMissWrite	Sets the number of cache misses.
	PLIB_PCACHE_LeastRecentlyUsedStateRead	Reads the state of the cache least recently used encoding bits.

Feature Existence Functions

	Name	Description
	PLIB_PCACHE_ExistsCacheHit	Identifies that the CacheHit feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLine	Identifies that the CacheLineSelect feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineAddr	Identifies that the CacheLineAddr feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineFlashType	Identifies that the CacheLineFlashType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineLock	Identifies that the CacheLineLock feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineMask	Identifies that the CacheLineMask feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineType	Identifies that the CacheLineType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineValid	Identifies that the CacheLineValid feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheMiss	Identifies that the CacheMiss feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsDataCacheEnable	Identifies that the DataCacheEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashDEDStatus	Identifies that the FlashDEDStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECCount	Identifies that the FlashSECCount feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECInt	Identifies that the FlashSECInt feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECStatus	Identifies that the FlashSECStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsInvalidateOnPFMPProgram	Identifies that the InvalidateOnPFMPProgram feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsLeastRecentlyUsedState	Identifies that the LeastRecentlyUsedState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchAbort	Identifies that the PrefetchAbort feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchEnable	Identifies that the PrefetchEnable feature exists on the Prefetch Cache module.

	PLIB_PCACHE_ExistsWaitState	Identifies that the WaitState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWord	Identifies that the Word feature exists on the Prefetch Cache module.

Flash ECC Functions

	Name	Description
	PLIB_PCACHE_FlashDEDStatusClear	Clears a bus exception caused by a double-bit error.
	PLIB_PCACHE_FlashDEDStatusGet	Determines if a bus exception was caused by a double-bit error.
	PLIB_PCACHE_FlashSECCountGet	Returns the current value of the Flash SEC counter.
	PLIB_PCACHE_FlashSECCountSet	Sets the number of single-bit error corrections that must occur before the SEC status is set to true.
	PLIB_PCACHE_FlashSECIntDisable	Disables an interrupt on SEC.
	PLIB_PCACHE_FlashSECIntEnable	Enables an interrupt on SEC.
	PLIB_PCACHE_FlashSECStatusClear	Sets the single-bit error corrected status to false.
	PLIB_PCACHE_FlashSECStatusGet	Determines if a SEC event triggered a Prefetch Cache interrupt.
	PLIB_PCACHE_FlashSECStatusSet	Sets the single-bit error corrected status to true.

Prefetch Control Functions

	Name	Description
	PLIB_PCACHE_PrefetchEnableGet	Gets the predictive prefetch state for the Prefetch Cache module.
	PLIB_PCACHE_PrefetchEnableSet	Sets the predictive prefetch state for the Prefetch Cache module.

Prefetch Status Functions

	Name	Description
	PLIB_PCACHE_PrefetchAbortRead	Reads the number of prefetch aborts.
	PLIB_PCACHE_PrefetchAbortWrite	Sets the number of prefetch aborts.

Description

This section provides detailed descriptions of the functions available in the Prefetch Cache Peripheral Library.

5.6.15.6.1 Cache Control Functions

5.6.15.6.1.1 PLIB_PCACHE_DataCacheEnableGet Function

C

```
PLIB_PCACHE_DATA_ENABLE PLIB_PCACHE_DataCacheEnableGet(
    PCACHE_MODULE_ID index
);
```

Description

This function gets the data cache enable bits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

[PLIB_PCACHE_DATA_ENABLE](#)

Remarks

At reset, data caching is disabled, and this function will return `PLIB_PCACHE_DATA_DISABLE`.

This function implements an operation of the `DataCacheEnable` feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsDataCacheEnable` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_DATA_ENABLE dcache_en;  
dcache_en = PLIB_PCACHE_DataCacheEnableGet(PCACHE_ID_0);
```

5.6.15.6.1.2 PLIB_PCACHE_DataCacheEnableSet Function

C

```
void PLIB_PCACHE_DataCacheEnableSet(  
    PCACHE_MODULE_ID index,  
    PLIB_PCACHE_DATA_ENABLE dcache_en  
);
```

Description

This function sets the data cache enable bits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dcache_en	PLIB_PCACHE_DATA_ENABLE

Returns

None.

Remarks

Data caching is disabled at reset.

This function implements an operation of the `DataCacheEnable` feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsDataCacheEnable` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_DataCacheEnableSet(PCACHE_ID_0, PLIB_PCACHE_DATA_4LINE);
```

5.6.15.6.1.3 PLIB_PCACHE_InvalidateOnPFMProgramAll Function

C

```
void PLIB_PCACHE_InvalidateOnPFMProgramAll(  
    PCACHE_MODULE_ID index  
);
```

Description

This function sets the prefetch cache to invalidate all data and instruction lines on a PFM program cycle.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the InvalidateOnPFMProgram feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsInvalidateOnPFMProgram` in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_InvalidateOnPFMProgramAll(PCACHE_ID_0);
```

5.6.15.6.1.4 PLIB_PCACHE_InvalidateOnPFMProgramUnlocked Function**C**

```
void PLIB_PCACHE_InvalidateOnPFMProgramUnlocked(  
    PCACHE_MODULE_ID index  
);
```

Description

This function sets the prefetch cache to invalidate all unlocked data and instruction lines on a PFM program cycle.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This is the default state at reset.

This function implements an operation of the InvalidateOnPFMProgram feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsInvalidateOnPFMProgram` in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_InvalidateOnPFMProgramUnlocked(PCACHE_ID_0);
```

5.6.15.6.1.5 PLIB_PCACHE_WaitStateGet Function**C**

```
uint32_t PLIB_PCACHE_WaitStateGet(  
    PCACHE_MODULE_ID index  
);
```

Description

This function gets the prefetch cache access time in terms of SYSCLK wait. states.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Number of clock cycles.

Remarks

At reset, this value is set to seven wait states.

This function implements an operation of the WaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWaitState in your application to to automatically determine whether this feature is available.

Example

```
uint32_t wait;  
wait = PLIB_PCACHE_WaitStateGet(PCACHE_ID_0);
```

5.6.15.6.1.6 PLIB_PCACHE_WaitStateSet Function**C**

```
void PLIB_PCACHE_WaitStateSet(  
    PCACHE_MODULE_ID index,  
    uint32_t clocks  
);
```

Description

This function sets the prefetch cache access time in terms of SYSCLK wait. states.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clocks	Identifies the wait state in terms of number of SYSCLK cycles

Returns

None.

Remarks

At reset, this value is set to seven wait states.

This function implements an operation of the WaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWaitState in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_WaitStateSet(PCACHE_ID_0, 0);
```


5.6.15.6.2 Cache Line Functions

5.6.15.6.2.1 PLIB_PCACHE_CacheLineAddrGet Function

C

```
uint32_t PLIB_PCACHE_CacheLineAddrGet(  
    PCACHE_MODULE_ID index  
);
```

Description

This function gets the TAG address in the cache line previously write- enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Address in the cache line TAG register.

Remarks

This function implements an operation of the CacheLineAddr feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineAddr in your application to to automatically determine whether this feature is available.

Example

```
uint32_t addr;  
addr = PLIB_PCACHE_CacheLineAddrGet(PCACHE_ID_0);
```

5.6.15.6.2.2 PLIB_PCACHE_CacheLineAddrSet Function

C

```
void PLIB_PCACHE_CacheLineAddrSet(  
    PCACHE_MODULE_ID index,  
    uint32_t addr  
);
```

Description

This function Sets the TAG address in the cache line previously write- enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
addr	Address to program into the cache line TAG register

Returns

None.

Remarks

This function implements an operation of the CacheLineAddr feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineAddr` in your application to automatically determine whether this feature is available.

Example

```
uint32_t addr = 0x1FC01234;
PLIB_PCACHE_CacheLineAddrSet(PCACHE_ID_0, addr);
```

5.6.15.6.2.3 PLIB_PCACHE_CacheLineData Function**C**

```
void PLIB_PCACHE_CacheLineData(
    PCACHE_MODULE_ID index
);
```

Description

This function sets the cache line previously enabled by [PLIB_PCACHE_CacheLineSelect](#) to data type.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the CacheLineType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineType` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineData(PCACHE_ID_0);
```

5.6.15.6.2.4 PLIB_PCACHE_CacheLineDeselect Function**C**

```
void PLIB_PCACHE_CacheLineDeselect(
    PCACHE_MODULE_ID index,
    uint32_t cache_line
);
```

Description

This function disables write access to the cache line specified by `cache_line`.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

cache_line	Specifies the cache line to disable write access

Returns

None.

Remarks

This function implements an operation of the CacheLineSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineSelect` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineDeselect(PCACHE_ID_0, 0);
```

5.6.15.6.2.5 PLIB_PCACHE_CacheLineFlashTypeBoot Function

C

```
void PLIB_PCACHE_CacheLineFlashTypeBoot(  
    PCACHE_MODULE_ID index  
);
```

Description

This function sets the cache line tag for the line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#) as residing in boot Flash.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the CacheLineFlashType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineFlashType` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineFlashTypeBoot(PCACHE_ID_0);
```

5.6.15.6.2.6 PLIB_PCACHE_CacheLineFlashTypeInst Function

C

```
void PLIB_PCACHE_CacheLineFlashTypeInst(  
    PCACHE_MODULE_ID index  
);
```

Description

This function sets the cache line tag for the line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#) as residing in instruction Flash.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the CacheLineFlashType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineFlashType` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineFlashTypeInst(PCACHE_ID_0);
```

5.6.15.6.2.7 PLIB_PCACHE_CacheLineFlashTypeIsInst Function**C**

```
bool PLIB_PCACHE_CacheLineFlashTypeIsInst(  
    PCACHE_MODULE_ID index  
);
```

Description

This function returns true if the cache line tag for the line previously write-enabled by `PLIB_PCACHE_CacheLineSelect` shows the line is residing in instruction Flash.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - the cache line resides in instruction Flash
- false - the cache line resides in boot Flash

Remarks

This function implements an operation of the CacheLineFlashType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineFlashType` in your application to automatically determine whether this feature is available.

Example

```
bool LineFlashType;  
LineFlashType = PLIB_PCACHE_CacheLineFlashTypeIsInst(PCACHE_ID_0);
```

5.6.15.6.2.8 PLIB_PCACHE_CacheLineInst Function**C**

```
void PLIB_PCACHE_CacheLineInst(  
    PCACHE_MODULE_ID index  
);
```

Description

This function sets the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#) to instruction type.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the CacheLineType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineType` in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineInst(PCACHE_ID_0);
```

5.6.15.6.2.9 PLIB_PCACHE_CacheLineInvalid Function

C

```
void PLIB_PCACHE_CacheLineInvalid(  
    PCACHE_MODULE_ID index  
);
```

Description

This function invalidates the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the CacheLineValid feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineValid` in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineInvalid(PCACHE_ID_0);
```

5.6.15.6.2.10 PLIB_PCACHE_CacheLineIsInst Function

C

```
bool PLIB_PCACHE_CacheLineIsInst(  
    PCACHE_MODULE_ID index
```

```
);
```

Description

This function returns true if the cache line previously enabled by [PLIB_PCACHE_CacheLineSelect](#) is set to instruction type.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

- true - cache line is instruction
- false - cache line is data

Remarks

This function implements an operation of the CacheLineType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineType` in your application to automatically determine whether this feature is available.

Example

```
bool CacheLineType;  
CacheLineType = PLIB_PCACHE_CacheLineIsInst(PCACHE_ID_0);
```

5.6.15.6.2.11 PLIB_PCACHE_CacheLineIsLocked Function

C

```
bool PLIB_PCACHE_CacheLineIsLocked(  
    PCACHE_MODULE_ID index  
);
```

Description

This function returns true if the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#) is locked.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - cache line is locked
- false - cache line is unlocked

Remarks

This function implements an operation of the CacheLineLock feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineLock` in your application to automatically determine whether this feature is available.

Example

```
bool CacheLineLocked;  
CacheLineLocked = PLIB_PCACHE_CacheLineIsLocked(PCACHE_ID_0);
```

5.6.15.6.2.12 PLIB_PCACHE_CacheLineIsValid Function

C

```
bool PLIB_PCACHE_CacheLineIsValid(  
    PCACHE_MODULE_ID index  
);
```

Description

This function returns true if the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#) is valid.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - cache line is valid
- false - cache line is invalid

Remarks

This function implements an operation of the CacheLineValid feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineValid` in your application to automatically determine whether this feature is available.

Example

```
bool CacheLineValid;  
CacheLineValid = PLIB_PCACHE_CacheLineIsValid(PCACHE_ID_0);
```

5.6.15.6.2.13 PLIB_PCACHE_CacheLineLock Function

C

```
void PLIB_PCACHE_CacheLineLock(  
    PCACHE_MODULE_ID index  
);
```

Description

This function locks the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the CacheLineLock feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineLock` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineLock(PCACHE_ID_0);
```

5.6.15.6.2.14 PLIB_PCACHE_CacheLineMaskGet Function**C**

```
uint32_t PLIB_PCACHE_CacheLineMaskGet(  
    PCACHE_MODULE_ID index  
);
```

Description

This function returns the current state of the cache tag mask for the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Address mask - Bits set to '1' will force a match (valid bits: <15:5>).

Remarks

This function implements an operation of the CacheLineMask feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineMask in your application to to automatically determine whether this feature is available.

Example

```
uint32_t mask;  
mask = PLIB_PCACHE_CacheLineMaskGet(PCACHE_ID_0);
```

5.6.15.6.2.15 PLIB_PCACHE_CacheLineMaskSet Function**C**

```
void PLIB_PCACHE_CacheLineMaskSet(  
    PCACHE_MODULE_ID index,  
    uint32_t mask  
);
```

Description

This function sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mask	Address mask (bits set to '1' will force a match - valid bits: <15:5>)

Returns

None.

Remarks

This function implements an operation of the CacheLineMask feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineMask` in your application to automatically determine whether this feature is available.

Example

```
uint32_t mask = 0x0C0;
PLIB_PCACHE_CacheLineMaskSet(PCACHE_ID_0, mask);
```

5.6.15.6.2.16 PLIB_PCACHE_CacheLineSelect Function**C**

```
void PLIB_PCACHE_CacheLineSelect(
    PCACHE_MODULE_ID index,
    uint32_t cache_line
);
```

Description

This function enables write access to the cache line specified by `cache_line`.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cache_line	Specifies the cache line to enable write access

Returns

None.

Remarks

At reset, all cache lines are read-only.

This function implements an operation of the CacheLineSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineSelect` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineSelect(PCACHE_ID_0, 0);
```

5.6.15.6.2.17 PLIB_PCACHE_CacheLineUnlock Function**C**

```
void PLIB_PCACHE_CacheLineUnlock(
    PCACHE_MODULE_ID index
);
```

Description

This function unlocks the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the CacheLineLock feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineLock` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineUnlock(PCACHE_ID_0);
```

5.6.15.6.2.18 PLIB_PCACHE_CacheLineValid Function**C**

```
void PLIB_PCACHE_CacheLineValid(  
    PCACHE_MODULE_ID index  
);
```

Description

This function validates the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the CacheLineValid feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheLineValid` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheLineValid(PCACHE_ID_0);
```

5.6.15.6.2.19 PLIB_PCACHE_WordRead Function**C**

```
uint32_t PLIB_PCACHE_WordRead(  
    PCACHE_MODULE_ID index,  
    uint32_t word  
);
```

Description

This function reads the word specified by word from the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read
word	Location of the word in the data array to be read

Returns

32-bit unsigned word from cache data array.

Remarks

This function implements an operation of the Word feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWord in your application to to automatically determine whether this feature is available.

Example

```
uint32_t word;  
word = PLIB_PCACHE_WordRead(PCACHE_ID_0, 0);
```

5.6.15.6.2.20 PLIB_PCACHE_WordWrite Function**C**

```
void PLIB_PCACHE_WordWrite(  
    PCACHE_MODULE_ID index,  
    uint32_t word,  
    uint32_t data  
);
```

Description

This function writes the word (specified by word parameter) with data (specified by data parameter) to the cache line previously write-enabled by [PLIB_PCACHE_CacheLineSelect](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be written.
word	Location of the word in the data array to be written
data	32-bit unsigned word to write to cache data array

Returns

None.

Remarks

This function implements an operation of the Word feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWord in your application to to automatically determine whether this feature is available.

Example

```
uint32_t word = 0;
uint32_t data = 0xDEADBEEF;
PLIB_PCACHE_WordWrite(PCACHE_ID_0, word, data);
```

5.6.15.6.3 Cache Status Functions

5.6.15.6.3.1 PLIB_PCACHE_CacheHitRead Function

C

```
uint32_t PLIB_PCACHE_CacheHitRead(
    PCACHE_MODULE_ID index
);
```

Description

This function reads the number of cache hits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Number of cache hits.

Remarks

This function implements an operation of the CacheHit feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheHit in your application to to automatically determine whether this feature is available.

Example

```
uint32_t cache_hits;
cache_hits = PLIB_PCACHE_CacheHitRead(PCACHE_ID_0);
```

5.6.15.6.3.2 PLIB_PCACHE_CacheHitWrite Function

C

```
void PLIB_PCACHE_CacheHitWrite(
    PCACHE_MODULE_ID index,
    uint32_t data
);
```

Description

This function sets the number of cache hits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be written
data	Number of cache hits to write

Returns

None.

Remarks

This function implements an operation of the CacheHit feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheHit` in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheHitWrite(PCACHE_ID_0, 0xF00);
```

5.6.15.6.3.3 PLIB_PCACHE_CacheMissRead Function**C**

```
uint32_t PLIB_PCACHE_CacheMissRead(  
    PCACHE_MODULE_ID index  
);
```

Description

This function reads the number of cache misses.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read.

Returns

Number of cache misses.

Remarks

This function implements an operation of the CacheMiss feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheMiss` in your application to to automatically determine whether this feature is available.

Example

```
uint32_t cache_misses;  
cache_misses = PLIB_PCACHE_CacheMissRead(PCACHE_ID_0);
```

5.6.15.6.3.4 PLIB_PCACHE_CacheMissWrite Function**C**

```
void PLIB_PCACHE_CacheMissWrite(  
    PCACHE_MODULE_ID index,  
    uint32_t data  
);
```

Description

This function sets the number of cache misses.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be written
data	Number of cache misses to write

Returns

None.

Remarks

This function implements an operation of the CacheMiss feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsCacheMiss` in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_CacheMissWrite(PCACHE_ID_0, 0xF00);
```

5.6.15.6.3.5 PLIB_PCACHE_LeastRecentlyUsedStateRead Function**C**

```
uint32_t PLIB_PCACHE_LeastRecentlyUsedStateRead(  
    PCACHE_MODULE_ID index  
);
```

Description

This function reads the state of the cache least recently used encoding bits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read.

Returns

State of Least Recently Used encoding bits represented as a 25-bit unsigned integer.

Remarks

This function implements an operation of the LeastRecentlyUsedState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsLeastRecentlyUsedState` in your application to to automatically determine whether this feature is available.

Example

```
uint32_t lru_state;  
lru_state = PLIB_PCACHE_LeastRecentlyUsedStateRead(PCACHE_ID_0);
```

5.6.15.6.4 Prefetch Control Functions**5.6.15.6.4.1 PLIB_PCACHE_PrefetchEnableGet Function****C**

```
PLIB_PCACHE_PREFETCH_ENABLE PLIB_PCACHE_PrefetchEnableGet(  
    PCACHE_MODULE_ID index  
);
```

Description

This function gets the predictive prefetch state for the Prefetch Cache module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

[PLIB_PCACHE_PREFETCH_ENABLE](#)

Remarks

At reset, the Pretch Cache module is disabled and this function will return PLIB_PCACHE_DISABLE.

This function implements an operation of the PrefetchEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchEnable in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_PREFETCH_ENABLE region;  
region = PLIB_PCACHE_PrefetchEnableGet(PCACHE_ID_0);
```

5.6.15.6.4.2 PLIB_PCACHE_PrefetchEnableSet Function**C**

```
void PLIB_PCACHE_PrefetchEnableSet(  
    PCACHE_MODULE_ID index,  
    PLIB_PCACHE_PREFETCH_ENABLE region  
);
```

Description

This function sets the predictive prefetch state for the Prefetch Cache module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
region	PLIB_PCACHE_PREFETCH_ENABLE

Returns

None.

Remarks

At reset, the Pretch Cache module is disabled.

This function implements an operation of the PrefetchEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchEnable in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_PrefetchEnableSet(PCACHE_ID_0,
```

```
PLIB_PCACHE_PREFETCH_ENABLE_ALL);
```

5.6.15.6.5 Prefetch Status Functions

5.6.15.6.5.1 PLIB_PCACHE_PrefetchAbortRead Function

C

```
uint32_t PLIB_PCACHE_PrefetchAbortRead(  
    PCACHE_MODULE_ID index  
);
```

Description

This function reads the number of prefetch aborts.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

Number of prefetch aborts.

Remarks

This function implements an operation of the PrefetchAbort feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchAbort in your application to automatically determine whether this feature is available.

Example

```
uint32_t pfabt;  
pfabt = PLIB_PCACHE_PrefetchAbortRead(PCACHE_ID_0);
```

5.6.15.6.5.2 PLIB_PCACHE_PrefetchAbortWrite Function

C

```
void PLIB_PCACHE_PrefetchAbortWrite(  
    PCACHE_MODULE_ID index,  
    uint32_t data  
);
```

Description

This function Sets the number of prefetch aborts.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be written
data	Number of prefetch aborts

Returns

None.

Remarks

This function implements an operation of the PrefetchAbort feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsPrefetchAbort` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_PrefetchAbortWrite(PCACHE_ID_0, 0xF00);
```

5.6.15.6.6 Flash ECC Functions

5.6.15.6.6.1 PLIB_PCACHE_FlashDEDStatusClear Function

C

```
void PLIB_PCACHE_FlashDEDStatusClear(  
    PCACHE_MODULE_ID index  
);
```

Description

This function clears a bus exception caused by a double-bit error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be cleared

Returns

None

Remarks

The DED status is set by hardware.

This function implements an operation of the FlashDEDStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsFlashDEDStatus` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_FlashDEDStatusClear(PCACHE_ID_0);
```

5.6.15.6.6.2 PLIB_PCACHE_FlashDEDStatusGet Function

C

```
bool PLIB_PCACHE_FlashDEDStatusGet(  
    PCACHE_MODULE_ID index  
);
```

Description

This function determines if a bus exception was caused by a double-bit error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

- true - A double-bit error was detected
- false - A double-bit error was not detected

Remarks

The Double-bit Error Detected (DED) status is set by hardware.

This function implements an operation of the FlashDEDStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashDEDStatus in your application to automatically determine whether this feature is available.

Example

```
bool dedStatus;  
dedStatus = PLIB_PCACHE_FlashDEDStatusGet(PCACHE_ID_0);
```

5.6.15.6.6.3 PLIB_PCACHE_FlashSECCountGet Function**C**

```
uint8_t PLIB_PCACHE_FlashSECCountGet(  
    PCACHE_MODULE_ID index  
);
```

Description

This function returns the current value of the Flash SEC counter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

count - Number of SEC events to occur before the SEC status is set to true

Remarks

This function implements an operation of the FlashSECCount feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECCount in your application to automatically determine whether this feature is available.

Example

```
uint8_t count;  
count = PLIB_PCACHE_FlashSECCountGet(PCACHE_ID_0);
```

5.6.15.6.6.4 PLIB_PCACHE_FlashSECCountSet Function**C**

```
void PLIB_PCACHE_FlashSECCountSet(  
    PCACHE_MODULE_ID index,  
    uint8_t count  
);
```

Description

This function sets the number of single-bit error corrections that must occur before the SEC status is set to true.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
count	Number of SEC events to occur before SEC status is set to true

Returns

None

Remarks

This function implements an operation of the FlashSECCount feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsFlashSECCount` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_FlashSECCountSet(PCACHE_ID_0, 255);
```

5.6.15.6.6.5 PLIB_PCACHE_FlashSECIntDisable Function**C**

```
void PLIB_PCACHE_FlashSECIntDisable(  
    PCACHE_MODULE_ID index  
);
```

Description

This function disables an interrupt on SEC.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Returns

None

Remarks

At reset the SEC interrupt is disabled.

This function implements an operation of the FlashSECInt feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsFlashSECInt` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_FlashSECIntDisable(PCACHE_ID_0);
```

5.6.15.6.6.6 PLIB_PCACHE_FlashSECIntEnable Function

C

```
void PLIB_PCACHE_FlashSECIntEnable(  
    PCACHE_MODULE_ID index  
);
```

Description

This function enables an interrupt on SEC.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Returns

None

Remarks

At reset the SEC interrupt is disabled.

This function implements an operation of the FlashSECInt feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECInt in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_FlashSECIntEnable(PCACHE_ID_0);
```

5.6.15.6.6.7 PLIB_PCACHE_FlashSECStatusClear Function

C

```
void PLIB_PCACHE_FlashSECStatusClear(  
    PCACHE_MODULE_ID index  
);
```

Description

This function sets the single-bit error corrected status to false.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be cleared

Returns

None

Remarks

The SEC status must be cleared in response to a Prefetch Cache SEC interrupt.

This function implements an operation of the FlashSECStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECStatus in your application to

to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_FlashSECStatusClear(PCACHE_ID_0);
```

5.6.15.6.6.8 PLIB_PCACHE_FlashSECStatusGet Function

C

```
bool PLIB_PCACHE_FlashSECStatusGet(  
    PCACHE_MODULE_ID index  
);
```

Description

This function determines if a SEC event triggered a Prefetch Cache interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Returns

- true - A single-bit error was corrected
- false - A double-bit error was not corrected

Remarks

The SEC bit is set when a single-bit error is corrected and the SEC counter is zero. If the Flash SEC interrupt is enabled, an error event is reported to the CPU by a Prefetch Cache interrupt event.

This function implements an operation of the FlashSECStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECStatus in your application to automatically determine whether this feature is available.

Example

```
bool secStatus;  
secStatus = PLIB_PCACHE_FlashSECStatusGet(PCACHE_ID_0);
```

5.6.15.6.6.9 PLIB_PCACHE_FlashSECStatusSet Function

C

```
void PLIB_PCACHE_FlashSECStatusSet(  
    PCACHE_MODULE_ID index  
);
```

Description

This function sets the single-bit error corrected status to true.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None

Remarks

This function is provided for code testing and debug purposes. Setting the SEC status while the SEC count is zero and the SEC interrupt is enabled will trigger a Prefetch Cache interrupt to the CPU.

This function implements an operation of the FlashSECStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsFlashSECStatus` in your application to automatically determine whether this feature is available.

Example

```
PLIB_PCACHE_FlashSECStatusSet(PCACHE_ID_0);
```

5.6.15.6.7 Data Types and Constants

5.6.15.6.7.1 PLIB_PCACHE_MAX_SEC_COUNT Macro

C

```
#define PLIB_PCACHE_MAX_SEC_COUNT 255
```

Description

Maximum Value for Single Bit Error Counter

This definition specifies the maximum value for single bit error counter.

Remarks

The actual maximum value of the SEC counter is processor specific and is defined in the processor-specific header files (see `processor.h`).

5.6.15.6.7.2 PLIB_PCACHE_NUM_LINES Macro

C

```
#define PLIB_PCACHE_NUM_LINES
```

Description

Number of Cache Lines

This definition specifies the number of Prefetch Cache Lines available.

Remarks

The actual number of cache lines is processor specific and is defined in the processor-specific header files (see `processor.h`).

5.6.15.6.7.3 PLIB_PCACHE_NUM_WORDS_PER_LINE Macro

C

```
#define PLIB_PCACHE_NUM_WORDS_PER_LINE
```

Description

Number of Words per Cache Line

This definition specifies the number of Words per Prefetch Cache Line.

Remarks

The actual number of words per cache line is processor specific and is defined in the processor-specific header files (see processor.h).

5.6.15.6.7.4 PCACHE_MODULE_ID Enumeration**C**

```
typedef enum {
    PCACHE_ID_1,
    PCACHE_NUMBER_OF_MODULES
} PCACHE_MODULE_ID;
```

Description

Module ID

This enumeration lists the possible Module IDs for the pcache.

Remarks

Refer to the data sheet to get the correct number of modules defined for the desired device.

5.6.15.6.7.5 PLIB_PCACHE_DATA_ENABLE Enumeration**C**

```
typedef enum {
    PLIB_PCACHE_DATA_DISABLE,
    PLIB_PCACHE_DATA_1LINE,
    PLIB_PCACHE_DATA_2LINE,
    PLIB_PCACHE_DATA_4LINE
} PLIB_PCACHE_DATA_ENABLE;
```

Description

Data Cache Enable

This enumeration lists the possible predictive prefetch states for the Prefetch Cache.

Members

Members	Description
PLIB_PCACHE_DATA_DISABLE	Disable data caching
PLIB_PCACHE_DATA_1LINE	Enable data caching with a size of 1 Line
PLIB_PCACHE_DATA_2LINE	Enable data caching with a size of 2 Lines
PLIB_PCACHE_DATA_4LINE	Enable data caching with a size of 4 Lines

Remarks

Not all settings are available for all devices. See the device-specific data sheet for details. This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

5.6.15.6.7.6 PLIB_PCACHE_PREFETCH_ENABLE Enumeration**C**

```
typedef enum {
    PLIB_PCACHE_PREFETCH_DISABLE,
    PLIB_PCACHE_PREFETCH_ENABLE_CACHED_REGIONS,
    PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST,
    PLIB_PCACHE_PREFETCH_ENABLE_NONCACHED_REGIONS,
    PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST_DATA,
```

```
    PLIB_PCACHE_PREFETCH_ENABLE_ALL  
} PLIB_PCACHE_PREFETCH_ENABLE;
```

Description

Predictive Prefetch Cache Enable Bits

This enumeration lists the possible predictive prefetch states for the Prefetch Cache.

Members

Members	Description
PLIB_PCACHE_PREFETCH_DISABLE	Disable predictive prefetch enable
PLIB_PCACHE_PREFETCH_ENABLE_CACHED_REGIONS	Enable predictive prefetch cache for cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST	Enable predictive prefetch cache for cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_NONCACHED_REGIONS	Enable predictive prefetch cache for non-cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST_DATA	Enable predictive prefetch cache for non-cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_ALL	Enable predictive prefetch cache for cacheable and non-cacheable regions

Remarks

Not all settings are available for all devices. See the device-specific data sheet for details. This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

5.6.15.6.8 Feature Existence Functions

5.6.15.6.8.1 PLIB_PCACHE_ExistsCacheHit Function

C

```
bool PLIB_PCACHE_ExistsCacheHit(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the CacheHit feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_CacheHitRead](#)
- [PLIB_PCACHE_CacheHitWrite](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CacheHit feature is supported on the device
- false - The CacheHit feature is not supported on the device

Remarks

None.

5.6.15.6.8.2 PLIB_PCACHE_ExistsCacheLine Function

C

```
bool PLIB_PCACHE_ExistsCacheLine(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the CacheLineSelect feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_CacheLineSelect](#)
- [PLIB_PCACHE_CacheLineDeselect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CacheLineSelect feature is supported on the device
- false - The CacheLineSelect feature is not supported on the device

Remarks

None.

5.6.15.6.8.3 PLIB_PCACHE_ExistsCacheLineAddr Function

C

```
bool PLIB_PCACHE_ExistsCacheLineAddr(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the CacheLineAddr feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_CacheLineAddrSet](#)
- [PLIB_PCACHE_CacheLineAddrGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CacheLineAddr feature is supported on the device
- false - The CacheLineAddr feature is not supported on the device

Remarks

None.

5.6.15.6.8.4 PLIB_PCACHE_ExistsCacheLineFlashType Function

C

```
bool PLIB_PCACHE_ExistsCacheLineFlashType(
    PCACHE_MODULE_ID index
);
```

Description

This interface identifies that the CacheLineFlashType feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_CacheLineFlashTypeBoot](#)
- [PLIB_PCACHE_CacheLineFlashTypeInst](#)
- [PLIB_PCACHE_CacheLineFlashTypeIsInst](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CacheLineFlashType feature is supported on the device
- false - The CacheLineFlashType feature is not supported on the device

Remarks

None.

5.6.15.6.8.5 PLIB_PCACHE_ExistsCacheLineLock Function

C

```
bool PLIB_PCACHE_ExistsCacheLineLock(
    PCACHE_MODULE_ID index
);
```

Description

This interface identifies that the CacheLineLock feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_CacheLineLock](#)
- [PLIB_PCACHE_CacheLineUnlock](#)
- [PLIB_PCACHE_CacheLineIsLocked](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CacheLineLock feature is supported on the device
- false - The CacheLineLock feature is not supported on the device

Remarks

None.

5.6.15.6.8.6 PLIB_PCACHE_ExistsCacheLineMask Function**C**

```
bool PLIB_PCACHE_ExistsCacheLineMask(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the CacheLineMask feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_CacheLineMaskSet](#)
- [PLIB_PCACHE_CacheLineMaskGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CacheLineMask feature is supported on the device
- false - The CacheLineMask feature is not supported on the device

Remarks

None.

5.6.15.6.8.7 PLIB_PCACHE_ExistsCacheLineType Function**C**

```
bool PLIB_PCACHE_ExistsCacheLineType(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the CacheLineType feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_CacheLineData](#)
- [PLIB_PCACHE_CacheLineInst](#)

- [PLIB_PCACHE_CacheLineIsInst](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CacheLineType feature is supported on the device
- false - The CacheLineType feature is not supported on the device

Remarks

None.

5.6.15.6.8.8 PLIB_PCACHE_ExistsCacheLineValid Function

C

```
bool PLIB_PCACHE_ExistsCacheLineValid(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the CacheLineValid feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_CacheLineValid](#)
- [PLIB_PCACHE_CacheLineInvalid](#)
- [PLIB_PCACHE_CacheLineIsValid](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CacheLineValid feature is supported on the device
- false - The CacheLineValid feature is not supported on the device

Remarks

None.

5.6.15.6.8.9 PLIB_PCACHE_ExistsCacheMiss Function

C

```
bool PLIB_PCACHE_ExistsCacheMiss(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the CacheMiss feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_CacheMissRead](#)
- [PLIB_PCACHE_CacheMissWrite](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CacheMiss feature is supported on the device
- false - The CacheMiss feature is not supported on the device

Remarks

None.

5.6.15.6.8.10 PLIB_PCACHE_ExistsDataCacheEnable Function**C**

```
bool PLIB_PCACHE_ExistsDataCacheEnable(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the DataCacheEnable feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_DataCacheEnableSet](#)
- [PLIB_PCACHE_DataCacheEnableGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataCacheEnable feature is supported on the device
- false - The DataCacheEnable feature is not supported on the device

Remarks

None.

5.6.15.6.8.11 PLIB_PCACHE_ExistsFlashDEDStatus Function

C

```
bool PLIB_PCACHE_ExistsFlashDEDStatus(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the FlashDEDStatus feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_FlashDEDStatusGet](#)
- [PLIB_PCACHE_FlashDEDStatusClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FlashDEDStatus feature is supported on the device
- false - The FlashDEDStatus feature is not supported on the device

Remarks

None.

5.6.15.6.8.12 PLIB_PCACHE_ExistsFlashSECCount Function

C

```
bool PLIB_PCACHE_ExistsFlashSECCount(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the FlashSECCount feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_FlashSECCountSet](#)
- [PLIB_PCACHE_FlashSECCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FlashSECCount feature is supported on the device
- false - The FlashSECCount feature is not supported on the device

Remarks

None.

5.6.15.6.8.13 PLIB_PCACHE_ExistsFlashSECInt Function**C**

```
bool PLIB_PCACHE_ExistsFlashSECInt(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the FlashSECInt feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_FlashSECIntEnable](#)
- [PLIB_PCACHE_FlashSECIntDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FlashSECInt feature is supported on the device
- false - The FlashSECInt feature is not supported on the device

Remarks

None.

5.6.15.6.8.14 PLIB_PCACHE_ExistsFlashSECStatus Function**C**

```
bool PLIB_PCACHE_ExistsFlashSECStatus(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the FlashSECStatus feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_FlashSECStatusGet](#)
- [PLIB_PCACHE_FlashSECStatusSet](#)
- [PLIB_PCACHE_FlashSECStatusClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FlashSECStatus feature is supported on the device
- false - The FlashSECStatus feature is not supported on the device

Remarks

None.

5.6.15.6.8.15 PLIB_PCACHE_ExistsInvalidateOnPFMProgram Function**C**

```
bool PLIB_PCACHE_ExistsInvalidateOnPFMProgram(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the InvalidateOnPFMProgram feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_InvalidateOnPFMProgramAll](#)
- [PLIB_PCACHE_InvalidateOnPFMProgramUnlocked](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InvalidateOnPFMProgram feature is supported on the device
- false - The InvalidateOnPFMProgram feature is not supported on the device

Remarks

None.

5.6.15.6.8.16 PLIB_PCACHE_ExistsLeastRecentlyUsedState Function**C**

```
bool PLIB_PCACHE_ExistsLeastRecentlyUsedState(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the LeastRecentlyUsedState feature is available on the Prefetch Cache module. When this interface returns true, this function is supported on the device:

- [PLIB_PCACHE_LeastRecentlyUsedStateRead](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LeastRecentlyUsedState feature is supported on the device
- false - The LeastRecentlyUsedState feature is not supported on the device

Remarks

None.

5.6.15.6.8.17 PLIB_PCACHE_ExistsPrefetchAbort Function**C**

```
bool PLIB_PCACHE_ExistsPrefetchAbort(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the PrefetchAbort feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_PrefetchAbortRead](#)
- [PLIB_PCACHE_PrefetchAbortWrite](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PrefetchAbort feature is supported on the device
- false - The PrefetchAbort feature is not supported on the device

Remarks

None.

5.6.15.6.8.18 PLIB_PCACHE_ExistsPrefetchEnable Function**C**

```
bool PLIB_PCACHE_ExistsPrefetchEnable(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the PrefetchEnable feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_PrefetchEnableSet](#)
- [PLIB_PCACHE_PrefetchEnableGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PrefetchEnable feature is supported on the device
- false - The PrefetchEnable feature is not supported on the device

Remarks

None.

5.6.15.6.8.19 PLIB_PCACHE_ExistsWaitState Function**C**

```
bool PLIB_PCACHE_ExistsWaitState(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the WaitState feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_WaitStateSet](#)
- [PLIB_PCACHE_WaitStateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WaitState feature is supported on the device
- false - The WaitState feature is not supported on the device

Remarks

None.

5.6.15.6.8.20 PLIB_PCACHE_ExistsWord Function**C**

```
bool PLIB_PCACHE_ExistsWord(  
    PCACHE_MODULE_ID index  
);
```

Description

This interface identifies that the Word feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- [PLIB_PCACHE_WordRead](#)
- [PLIB_PCACHE_WordWrite](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Word feature is supported on the device
- false - The Word feature is not supported on the device

Remarks

None.

5.6.15.7 Files

Files

Name	Description
plib_pcache.h	Defines the Prefetch Cache (PCACHE) Peripheral Library Interface

Description

5.6.15.7.1 plib_pcache.h

Prefetch Cache Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Prefetch Cache (PCACHE) Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Prefetch Cache module.

Functions

	Name	Description
≡	PLIB_PCACHE_CacheHitRead	Reads the number of cache hits.
≡	PLIB_PCACHE_CacheHitWrite	Sets the number of cache hits.
≡	PLIB_PCACHE_CacheLineAddrGet	Gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_CacheLineAddrSet	Sets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
≡	PLIB_PCACHE_CacheLineData	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to data type.
≡	PLIB_PCACHE_CacheLineDeselect	Disables write access to the cache line specified by cache_line.
≡	PLIB_PCACHE_CacheLineFlashTypeBoot	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in boot Flash.

	PLIB_PCACHE_CacheLineFlashTypeInst	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in instruction Flash.
	PLIB_PCACHE_CacheLineFlashTypeIsInst	Returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing in instruction Flash.
	PLIB_PCACHE_CacheLineInst	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.
	PLIB_PCACHE_CacheLineInvalid	Invalidates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
	PLIB_PCACHE_CacheLineIsInst	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.
	PLIB_PCACHE_CacheLineIsLocked	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.
	PLIB_PCACHE_CacheLineIsValid	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.
	PLIB_PCACHE_CacheLineLock	Locks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
	PLIB_PCACHE_CacheLineMaskGet	Returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
	PLIB_PCACHE_CacheLineMaskSet	Sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
	PLIB_PCACHE_CacheLineSelect	Enables write access to the cache line specified by <code>cache_line</code> .
	PLIB_PCACHE_CacheLineUnlock	Unlocks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
	PLIB_PCACHE_CacheLineValid	Validates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .
	PLIB_PCACHE_CacheMissRead	Reads the number of cache misses.
	PLIB_PCACHE_CacheMissWrite	Sets the number of cache misses.
	PLIB_PCACHE_DataCacheEnableGet	Gets the data cache enable bits.
	PLIB_PCACHE_DataCacheEnableSet	Sets the data cache enable bits.
	PLIB_PCACHE_ExistsCacheHit	Identifies that the CacheHit feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLine	Identifies that the CacheLineSelect feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineAddr	Identifies that the CacheLineAddr feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineFlashType	Identifies that the CacheLineFlashType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineLock	Identifies that the CacheLineLock feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineMask	Identifies that the CacheLineMask feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineType	Identifies that the CacheLineType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineValid	Identifies that the CacheLineValid feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheMiss	Identifies that the CacheMiss feature exists on the Prefetch Cache module.

	PLIB_PCACHE_ExistsDataCacheEnable	Identifies that the DataCacheEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashDEDStatus	Identifies that the FlashDEDStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECCount	Identifies that the FlashSECCount feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECInt	Identifies that the FlashSECInt feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECStatus	Identifies that the FlashSECStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsInvalidateOnPFMProgram	Identifies that the InvalidateOnPFMProgram feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsLeastRecentlyUsedState	Identifies that the LeastRecentlyUsedState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchAbort	Identifies that the PrefetchAbort feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchEnable	Identifies that the PrefetchEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWaitState	Identifies that the WaitState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWord	Identifies that the Word feature exists on the Prefetch Cache module.
	PLIB_PCACHE_FlashDEDStatusClear	Clears a bus exception caused by a double-bit error.
	PLIB_PCACHE_FlashDEDStatusGet	Determines if a bus exception was caused by a double-bit error.
	PLIB_PCACHE_FlashSECCountGet	Returns the current value of the Flash SEC counter.
	PLIB_PCACHE_FlashSECCountSet	Sets the number of single-bit error corrections that must occur before the SEC status is set to true.
	PLIB_PCACHE_FlashSECIntDisable	Disables an interrupt on SEC.
	PLIB_PCACHE_FlashSECIntEnable	Enables an interrupt on SEC.
	PLIB_PCACHE_FlashSECStatusClear	Sets the single-bit error corrected status to false.
	PLIB_PCACHE_FlashSECStatusGet	Determines if a SEC event triggered a Prefetch Cache interrupt.
	PLIB_PCACHE_FlashSECStatusSet	Sets the single-bit error corrected status to true.
	PLIB_PCACHE_InvalidateOnPFMProgramAll	Sets prefetch cache to invalidate all data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_InvalidateOnPFMProgramUnlocked	Sets prefetch cache to invalidate all unlocked data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_LeastRecentlyUsedStateRead	Reads the state of the cache least recently used encoding bits.
	PLIB_PCACHE_PrefetchAbortRead	Reads the number of prefetch aborts.
	PLIB_PCACHE_PrefetchAbortWrite	Sets the number of prefetch aborts.
	PLIB_PCACHE_PrefetchEnableGet	Gets the predictive prefetch state for the Prefetch Cache module.
	PLIB_PCACHE_PrefetchEnableSet	Sets the predictive prefetch state for the Prefetch Cache module.
	PLIB_PCACHE_WaitStateGet	Gets the prefetch cache access time.
	PLIB_PCACHE_WaitStateSet	Sets the prefetch cache access time.
	PLIB_PCACHE_WordRead	Reads the word specified by word from cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .

	PLIB_PCACHE_WordWrite	Writes the word (specified by word parameter) with data (specified by data parameter) to cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect .

File Name

plib_pcache.h

Company

Microchip Technology Inc.

5.6.16 PMP Peripheral Library

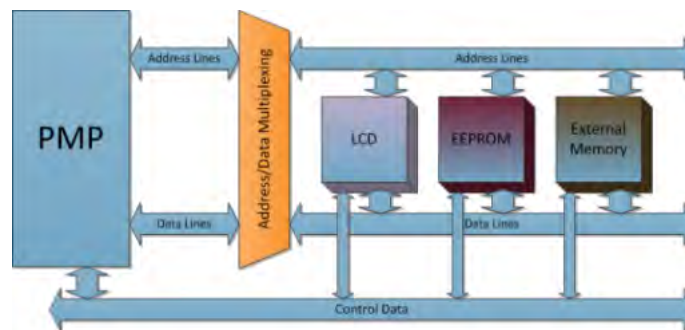
5.6.16.1 Introduction

Parallel Master Port (PMP) Peripheral Library for Microchip Microcontrollers

This library provides a low-level abstraction of the Parallel Master Port (PMP) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The following diagram shows a block diagram of the PMP module and how it interacts with other peripherals.



PMP Overview

5.6.16.2 Release Notes

MPLAB Harmony Version: v0.70b **PMP Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.16.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.16.4 Using the Library

Using the Library

This topic describes the basic architecture of the PMP Peripheral Library and provides information and examples on its use.

Interface Header File: [plib_pmp.h](#)

The interface to the PMP Peripheral library is defined in the "[plib_pmp.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the PMP Peripheral Library must include "peripheral.h".

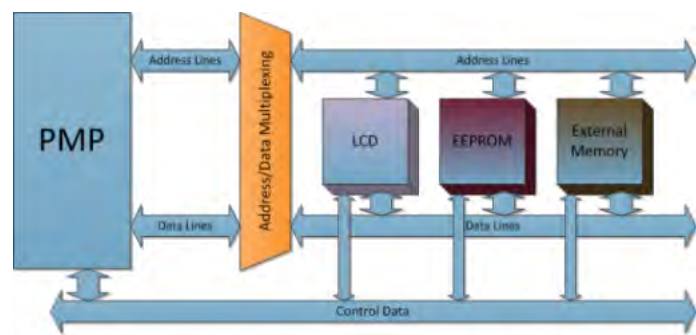
5.6.16.4.1 Hardware Abstraction Model

This library provides a low-level abstraction of the Parallel Master Port (PMP) module on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model Description

The PMP module provides interface routines to interact with external peripherals such as LCD, EEPROM, Flash memory, etc., as shown in the following diagram. The diagram shows the PMP module acting as a master. The PMP module can be easily configured to act as a slave. The address and data lines can be multiplexed to suit the application. The address and data buffers are up to 2-byte (16-bit) buffers for data transmitted or received by the parallel interface to the PMP bus over the data and address lines synchronized with control logic including the read and write strobe.



PMP Hardware Abstraction Model Diagram

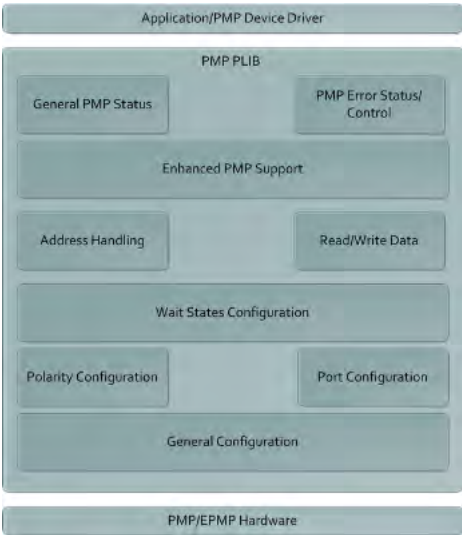
The desired timing wait states to suit different peripheral timings can also be programmed using the PMP module.

5.6.16.4.2 Library Usage Model

This section describes the library usage model for the PMP Peripheral Library.

Description

The following diagram shows the high level organization of the PMP PLIB usage model. The items in the diagram correspond to the groupings in the [Library Interface](#) section.



PMP Library Usage Model

Library Interface Section	Description

5.6.16.4.3 How the Library Works

How the Library Works

Before enabling the PMP module, the caller must initialize basic parameters such as wait states timings, and interrupt mode features (see the [Initialization](#) section).

After the module has been enabled, it can begin monitoring the bus as a slave device waiting to be addressed by an external master (see the [Operating as a Slave](#) section). A slave device only transfers data on the bus when it has been addressed by a master. If the module is used to start a transfer, it is operating as a master. A master addresses a slave and controls the transfer

of data by providing the clock (see the [Operating as a Master](#) section).

Some operations in the PMP peripheral library initiate actions on the PMP bus that require time to complete. Also, some events occur asynchronously on the bus. In each of these cases, the module causes either a "master", "slave", or "error" interrupt. The [State Machine](#) section describes the different states that can cause an interrupt and show what causes the transition from one state to another. The [Handling Errors](#) section describes the various errors that can occur and how to deal with them.

5.6.16.4.3.1 Initialization

This section describes the general initialization of the PMP module.

Description

The important configurations to look for are as follows.

Operating Mode

Depending on the device, the PMP module can function in Master modes 1 or 2, buffered/enhanced slave mode or the legacy slave mode. To set the PMP module in the appropriate mode, use [PLIB_PMP_OperationModeSelect](#).

Multiplexing Mode

Depending on the application requirement or hardware arrangement, the data lines can be used to multiplex the addressing information. Various multiplexing modes supported are listed by [PMP_MUX_MODE](#). Use [PLIB_PMP_MultiplexModeSelect](#) to select the appropriate multiplexing mode.

Chip Select Mode

As needed, the Chip Select lines can be made to function as Chip Select or as address lines. Use [PLIB_PMP_ChipSelectFunctionSelect](#) to select the required functionality of Chip Select lines.

Interrupt Mode

PMP generates interrupts based on the option selected for the interrupt mode. Interrupts can be enabled, disabled, generated at the end of a read or write cycle, or when other events occur. Use the [PLIB_PMP_InterruptModeSelect](#) to select these options.

Address Increment Mode

After every read/write cycle, the PMP module can be configured to automatically increment or decrement the address it is accessing. Use [PLIB_PMP_AddressIncrementModeSelect](#) to select this option.

To initialize the PMP module, perform the following sequence:

1. Select the desired initialization options using the General Initialization Functions to select the desired operation of the following features:
 - Operating mode
 - Multiplexing mode
 - Chip Select mode
 - Interrupt mode
 - Address Incrementing mode
 - Stop-in-Idle operation
2. Program the wait states of the PMP module. Refer to Wait States Initialization Functions for an example.
3. If operating in Master mode, set the desired address (8-bit in this case, the size may vary depending on the application) using [PLIB_PMP_AddressSet](#).
4. If running in an interrupt-driven environment, clear any pending interrupts and enable the appropriate (master, slave, and

error) PMP interrupts. (**Note:** Refer to the "Interrupt Controller" chapter in the specific device data sheet for details).

5. Enable the module for operation using `PLIB_PMP_Enable`.

Example

```
// Configure General PMP Options
// Select the PMP operation mode
PLIB_PMP_OperationModeSelect( PMP_ID_0, PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT );
// Set the multiplexing to none. Address and data lines are not multiplexed.
PLIB_PMP_MultiplexModeSelect( PMP_ID_0, PMP_MUX_NONE );
// Select the function of chip-select lines
PLIB_PMP_ChipSelectFunctionSelect( PMP_ID_0, PMCS1_PMCS2_AS_ADDRESS_LINES );
// Select the interrupt mode
PLIB_PMP_InterruptModeSelect( PMP_ID_0, PMP_INTERRUPT_NONE );
// disable the auto increment/decrement of address after each read/write
PLIB_PMP_AddressIncrementModeSelect( PMP_ID_0, PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE );
// Enable stop in Idle mode.
PLIB_PMP_StopInIdleEnable( PMP_ID_0 );
// Set Desired Wait State Values
// Set the data wait states
PLIB_PMP_WaitStatesDataSetUpSelect( PMP_ID_0, PMP_DATA_WAIT_FOUR );
// Set the strobe wait states
PLIB_PMP_WaitStatesStrobeSelect( PMP_ID_0, PMP_STROBE_WAIT_1 );
// Set the data hold wait states.
PLIB_PMP_WaitStatesDataHoldSelect( PMP_ID_0, PMP_DATA_HOLD_1 );
// Set the appropriate address (MASTER only)
PLIB_PMP_AddressSet ( PMP_ID_0, 0x12 );

// Optional: Clear and enable interrupts before enabling the PMP module.
// Enable the module
PLIB_PMP_Enable( PMP_ID_0 );
```

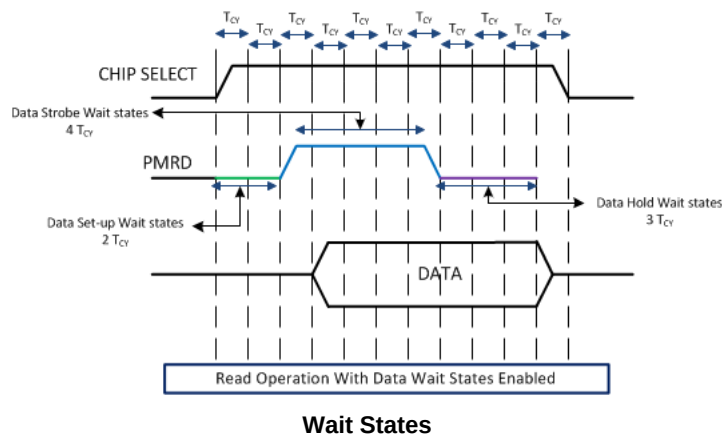
Note: Refer to the "Interrupt Controller" chapter in the specific device data sheet for details on how to clear and enable the PMP module interrupts.

5.6.16.4.3.2 Wait States Initialization

This section describes set up and initialization of the wait states for the PMP module.

Description

In Master mode, the user can control the duration of the read, write and address cycles by configuring the module Wait states. One Wait state period is equivalent to one peripheral bus clock cycles. The following figure shows an example of a Master Read operation using Wait states.



Wait states can be added to the beginning, middle and end of any read or write cycle using the corresponding bits in the PMP

Configuration. The wait states differ depending on which PMP mode you are in, but setting the wait states is the same per mode and device, as shown in the following example.

Initializing Wait States

The following sequence can be used to set up the wait states for the PMP Master mode.

Preconditions:

None.

Process:

1. Set the data hold time using [PLIB_PMP_WaitStatesDataHoldSelect](#).
2. Set the strobe wait time using [PLIB_PMP_WaitStatesStrobeSelect](#).
3. Set the data wait time using [PLIB_PMP_WaitStatesDataSetUpSelect](#).

Example

```
// Set the data wait states
PLIB_PMP_WaitStatesDataSetUpSelect( PMP_ID_0, PMP_DATA_WAIT_ONE );
// Set the strobe wait states
PLIB_PMP_WaitStatesStrobeSelect( PMP_ID_0, PMP_STROBE_WAIT_1 );
// Set the data hold wait states.
PLIB_PMP_WaitStatesDataHoldSelect( PMP_ID_0, PMP_DATA_HOLD_1 );
```

5.6.16.4.3.3 Operating as a Master

This section describes how to set up the PMP module to operate as a master.

Description

The PMP module, while acting as a master, waits for its input or output buffers to be read or written. Once the buffers have been manipulated, the appropriate action takes place.

A PMP bus transfer always begins with reading or writing to the appropriate PMP input/output buffers. A read from a PMP input buffer performs a PMP read. A write to a PMP output buffer performs a PMP write. The address pins associated with the transfer will have the value that is inside the address register of the PMP module. These steps are summarized as follows.

Summary of Steps

- Set up the proper PMP address
- Send or read data bytes

Each of these steps making up a transfer will take some time to complete. By monitoring the busy bit of the PMP module, the user application can determine whether the transfer is complete. The completion of each step can cause a PMP interrupt.

Sending a Data Byte

The following sequence can be used to [send](#) a data byte and repeated to [send](#) an arbitrary number of data bytes.

Preconditions:

- The address of the destination must be configured in the PMP module
- The proper control signals are configured for PMP operation
- The PMP module is configured for the desired Master mode
- The PMP module is enabled

Process:

1. Ensure that the PMP module is not busy by using [PLIB_PMP_PortIsBusy](#).

2. Write the output data buffer using `PLIB_PMP_MasterSend`.

Example:

```
uint8_t data = 'a';
// Set the destination address to be written
PLIB_PMP_AddressSet(PMP_ID_0, 0x1234 );
// Check to see if the PMP is busy, then send the byte
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    // Send the data
    PLIB_PMP_MasterSend( PMP_ID_0, data );
}
```

Receiving a Data byte

The following sequence can be used to [send](#) a data byte and repeated to [send](#) an arbitrary number of data bytes.

Preconditions:

- The address of the destination must be configured in the PMP module.
- The proper control signal are configured for PMP operation.

Process:

1. Ensure that the PMP module is not busy by using `PLIB_PMP_PortIsBusy`.
2. Write the output data buffer using `PLIB_PMP_MasterReceive`.

Example:

```
uint8_t data;
// Set the source address
PLIB_PMP_AddressSet(PMP_ID_0, 0x1234 );
// Check to see the PMP is not busy, then read the data
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    // receive the data
    data = PLIB_PMP_MasterReceive( PMP_ID_0 );
}
```

5.6.16.4.3.4 Operating as a Slave

This section describes how to set up the PMP module to operate as a slave.

Description

The PMP module while acting as a slave, waits for read or write strobes to occur. Once the buffers have been manipulated, the appropriate action takes place.

Summary of Steps

- Check to see if data is available to be read
- Prepare output buffer for slave read

Each of these steps making up a transfer will take some time to complete. By monitoring the busy bit of the PMP module, the user application can determine whether the transfer is complete. The completion of each step can cause a PMP interrupt.

Reading Available Data

The following sequence can be used to receive a data byte and repeated to receive an arbitrary number of data bytes.

Preconditions:

- The address of the destination must be set in the PMP module
- The proper control signals are initialized for PMP operation

Process:

1. Check to see if data is available using [PLIB_PMP_InputBufferXIsFull](#).
2. Read the available data using [PLIB_PMP_InputBufferXByteReceive](#).

Example:

```
uint8_t data;
// Check to see if data is available, then receive the byte
if(PLIB_PMP_InputBufferXIsFull( PMP_ID_0, 1))
{
    // Receive the data from buffer one.
    data = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 1 );
}
```

Preparing Output Buffer for Slave Write

The following sequence can be used to [send](#) a data byte and repeated to [send](#) an arbitrary number of data bytes.

Preconditions:

The proper control signals are initialized for PMP operation.

Process:

1. Ensure that the output buffer to be written is empty by using [PLIB_PMP_OutputBufferXIsEmpty](#).
2. Write the output data buffer using [PLIB_PMP_OutputBufferXByteSend](#).

Example:

```
uint8_t data;
// Check to see if output buffer is available, then send the byte
if(PLIB_PMP_OutputBufferXIsEmpty( PMP_ID_0, 1))
{
    // Fill the output buffer-1 with the data to be sent
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 1, value);
}
```

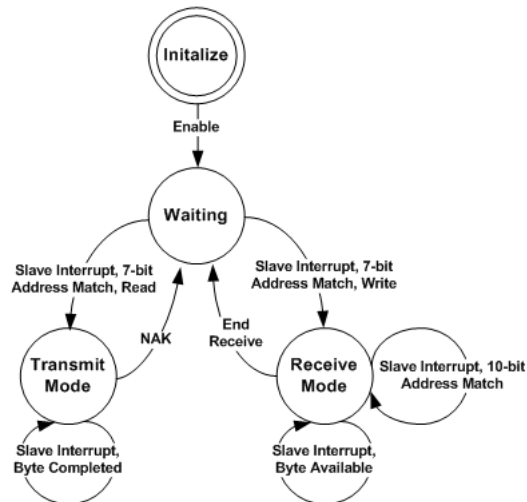
5.6.16.4.3.5 State Machine

The PMP state machine can be used in either a polled or an interrupt-driven manner. However, in both cases, software must check the state of the master, slave, and error interrupt flags to identify when a state transition occurs.

Description

The PMP module has one basic interrupt that is triggered when data has been written or read. In Buffered Slave mode, this interrupt can be set to wait until an 'X' amount of buffers are written or read. This interrupt must be cleared in software, but can also be used in DMA transactions to trigger the next DMA transfer.

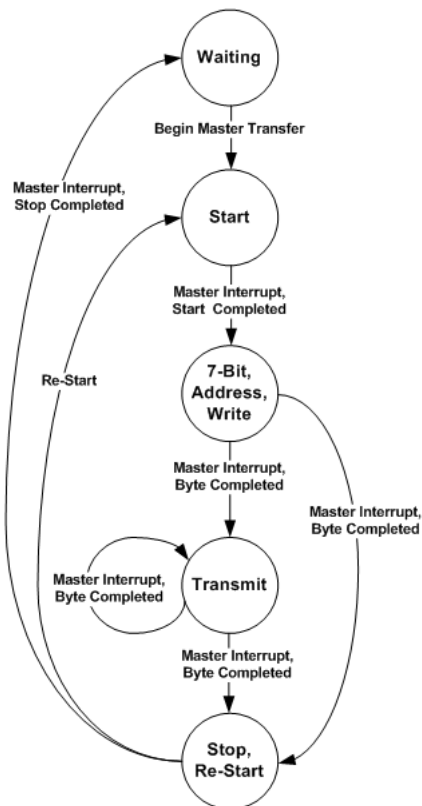
The PMP module will not start generating interrupts (setting the PMP interrupt flags, even if interrupts are disabled) until it is properly configured and enabled. After that, interrupts are generated as described in subsequent sections. Software has to respond appropriately once the interrupt has occurred (the flag has been set) to allow the state machine to advance to the next state. These actions are described in the following sections for the state machine diagram to which they refer.



Slave Mode State Machine

Slave-Mode State Transitions

In Slave mode, state transitions occur under hardware control automatically or in response to an interrupt. Software must respond appropriately to ensure that the PMP module continues to operate correctly. Basically, if a read/write strobe occurs, then the appropriate read/write happens, which in turn sets the interrupt.



Master Mode State Machine

Master-Mode State Transitions

In Master mode, state transitions occur under hardware control automatically or in response to an interrupt. Software must respond appropriately to ensure that the PMP module continues to operate correctly. Basically, if an input or output register is manipulated, the appropriate read/write happens, which in turn sets the interrupt. The PMP module can also be configured to have read/write strobes indicating whether a read/write has completed.

5.6.16.4.3.6 Handling Errors

There are two basic types of errors that can occur during various slave PMP operations:

- Input Buffer Overrun
- Output Buffer Underflow

This section provides information on handling these types of errors.

Description

Handling Input Buffer Overrun Errors

An input buffer overflow error occurs when the software is not reading or clearing the input buffers fast enough for the master PMP device attached to it. When this occurs, the master write is not allowed and the "input buffer overflow" status bit is set. This can be identified by calling [PLIB_PMP_InputOverflowHasOccurred](#). Additional attempts to write to the input buffer will not be allowed until the overflow error is cleared by calling [PLIB_PMP_InputOverflowClear](#). This type of error should be either avoided by having proper communication techniques with the PMP Master device or checked using [PLIB_PMP_InputBuffersAreFull](#), which informs the controller that no buffers have data in them (overflow) .

Interrupts: The input buffer overflow error does not trigger an interrupt.

Handling Output Buffer Underflow Errors

An output buffer underflow error occurs when the software is not writing to the output buffers fast enough for the master PMP device attached to it. When this occurs, the master read is not allowed and the "output buffer underflow" status bit is set. This can be identified by calling [PLIB_PMP_OutputUnderflowHasOccurred](#). Additional attempts to read from the output buffer will not be allowed until the overflow error is cleared by calling [PLIB_PMP_OutputUnderflowClear](#). This sort of error should be either avoided by having proper communication techniques with the PMP Master device, or checked using [PLIB_PMP_OutputBuffersAreEmpty](#), which informs the controller that buffers have data in them (underflow) .

Interrupts: The input buffer overflow error does not trigger an interrupt.

5.6.16.4.3.7 Other Features

This section provides information on additional features that may be available. These features are not available on all devices. Please refer to the **"Parallel Master Port (PMP)"** chapter in the specific device data sheet to determine which features are supported by your device.

Description

Operation During Sleep Mode

When the device enters Sleep mode, the system clock is disabled. The consequences of Sleep mode depend on which mode the PMP module is configured in at the time that Sleep mode is invoked.

Master

If the device enters Sleep mode while the PMP module is operating in Master mode, PMP operation will be suspended in its current state until clock execution resumes. As this may cause unexpected control pin timings, users should avoid invoking

Sleep mode when continuous use of the module is needed.

Slave

While the PMP module is inactive, but enabled for any Slave mode operation, any read or write operations occurring at that time will be able to complete without the use of the microcontroller clock. Once the operation is completed, the module will issue an interrupt.

This interrupt may be used to wake the device from Sleep or Idle modes, depending on the configuration and capabilities of the device. Refer to the specific device data sheet or the related peripheral section in the reference manual to determine the capabilities of your device.

Operation During Idle Mode

When the device enters Idle mode, the system clock sources remain functional and the CPU stops code execution. PMP operation during Idle mode can be controlled using the [PLIB_PMP_StopInIdleEnable](#) or the [PLIB_PMP_StopInIdleDisable](#). By default, the PMP module continues to operate in Idle mode and provide full functionality.

5.6.16.5 Configuring the Library

The library is configured for the supported PMP Peripheral Library is configured in the MPLAB IDE.

5.6.16.6 Library Interface

Data Types and Constants

	Name	Description
	PMP_ACK_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_ADDRESS_HOLD_LATCH_WAIT_STATES	PMP hold after address latch strobe wait states configuration.
	PMP_ADDRESS_LATCH	Address Latch Strobe configuration.
	PMP_ADDRESS_LATCH_WAIT_STATES	PMP address latch strobe wait states configuration.
	PMP_ADDRESS_PORT	Defines the different address lines that are available for configuration.
	PMP_ALTERNATE_MASTER_WAIT_STATES	PMP alternate master wait states.
	PMP_CHIP_SELECT	PMP Chip Select data type.
	PMP_CHIPSELECT_FUNCTION	Defines different functions available for the Chip Select lines multiplexed with address lines.
	PMP_DATA_HOLD_STATES	PMP Data Hold after strobe wait state.
	PMP_DATA_LENGTH	Possible data lengths handled by the PMP module.
	PMP_DATA_SIZE	PMP data size.
	PMP_DATA_WAIT_STATES	PMP data setup time configuration.
	PMP_INCREMENT_MODE	PMP address incrementing configuration.
	PMP_INPUT_BUFFER_TYPE	PMP Input Buffers type.
	PMP_INTERRUPT_MODE	PMP interrupt request mode data type.
	PMP_MASTER_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_MODULE_ID	Possible instances of the PMP module.
	PMP_MUX_MODE	Defines the different mode configurations in which the PMP can be enabled.

	PMP_OPERATION_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_PMBE_PORT	Defines the available Byte Enable ports.
	PMP_POLARITY_LEVEL	Possible polarity levels for the PMP pins.
	PMP_STROBE_WAIT_STATES	PMP strobe signal wait time configuration.

Address Handling Functions

	Name	Description
	PLIB_PMP_AddressGet	Gets the current address of the PMP module.
	PLIB_PMP_AddressLinesA0A1Get	Gets the value of the address lines PMA0 and PMA1.
	PLIB_PMP_AddressLinesA0A1Set	Sets the address lines PMA0 and PMA1 with the value specified.
	PLIB_PMP_AddressSet	Sets the current address of the PMP module to the specified address.

Data Read and Write Functions

	Name	Description
	PLIB_PMP_InputBuffersAreFull	Gets the state of the input buffers.
	PLIB_PMP_InputBufferXByteReceive	Data to be received in Byte mode.
	PLIB_PMP_InputBufferXIsFull	Gets the state of the identified input buffer.
	PLIB_PMP_IsDataReceived	Checks and returns if the data has been received in the specified buffer in Slave mode.
	PLIB_PMP_IsDataTransmitted	Checks and returns if the data has been transmitted from the specified buffer in Slave mode.
	PLIB_PMP_MasterReceive	Receives the data in the master mode.
	PLIB_PMP_MasterSend	Sends the specified data in Master mode.
	PLIB_PMP_OutputBuffersAreEmpty	Gets the state of the output buffers.
	PLIB_PMP_OutputBufferXByteSend	Data to be transmitted in Byte mode.
	PLIB_PMP_OutputBufferXIsEmpty	Gets the state of the input buffer.
	PLIB_PMP_SlaveReceive	Receives the data in Slave mode.
	PLIB_PMP_SlaveSend	Sends the specified data in Slave mode.

Enhanced Address Handling Functions

	Name	Description
	PLIB_PMP_ChipXBaseAddressSet	Sets the base address of the specified device.
	PLIB_PMP_ReservedAddressSpaceBitsSet	Sets the address bits of the reserved address space of the EPMP. The end address of the Chip Select-2 (if present).

Enhanced Error/Status Control Functions

	Name	Description
	PLIB_PMP_TransactionErrorClear	Clears the transaction error flag.
	PLIB_PMP_TransactionErrorHasOccurred	Returns the status of a transaction error.
	PLIB_PMP_TransactionHasTimedOut	Returns the time-out status.
	PLIB_PMP_TransactionTimeoutClear	Clears the time-out flag.

Enhanced General Initialization Functions

	Name	Description
	PLIB_PMP_BusKeeperDisable	Disables the bus keeper (high-impedance) state.
	PLIB_PMP_BusKeeperEnable	Enables the bus keeper (high-impedance) state.

	PLIB_PMP_ChipSelectXEnableStrobeDeSelect	Selects the read and write strobes. The enable strobe is not selected (not required).
	PLIB_PMP_ChipSelectXEnableStrobeSelect	Selects the read/write and enable strobes.
	PLIB_PMP_SmartAddressStrobeDisable	Disables "smart" address strobes.
	PLIB_PMP_SmartAddressStrobeEnable	Enables "smart" address strobes.
	PLIB_PMP_ChipXAckModeSelect	Selects the acknowledge mode for the PMP module.
	PLIB_PMP_ChipXDataSizeSelect	Enables data transfer size for the PMP for the specified device.
	PLIB_PMP_EnhancedMasterModeSelect	Selects Master mode for the PMP module.

Enhanced General Status Functions

	Name	Description
	PLIB_PMP_AlternateMasterHasAccess	Gets the status of which alternate masters have gained access to the PMP module.
	PLIB_PMP_AlternateMasterRequestStatus	Gets the status of the request from the alternate master.

Enhanced Polarity Configuration Functions

	Name	Description
	PLIB_PMP_ChipSelectXAckPolaritySelect	Sets the Chip Select acknowledge polarity.
	PLIB_PMP_ChipSelectXByteEnablePortPolaritySelect	Sets the nibble/byte enable polarity.
	PLIB_PMP_ChipSelectXReadWriteStrobePolaritySelect	Sets the read/write strobe polarity.
	PLIB_PMP_ChipSelectXWriteEnableStrobePolaritySelect	Sets the write/enable strobe polarity.

Enhanced Port Configuration Functions

	Name	Description
	PLIB_PMP_ChipSelectXPortDisable	Disables the specified Chip Select's port.
	PLIB_PMP_ChipSelectXPortEnable	Enables the specified Chip Select's port.

Enhanced Wait States Initialization Functions

	Name	Description
	PLIB_PMP_ChipXWaitStatesAlternateMasterSelect	Selects the wait states of the alternate master for Chip Select-X.
	PLIB_PMP_ChipXWaitStatesDataHoldSelect	Configures the data hold states (after data transfer) needed to be used with the PMP module for the specified device.
	PLIB_PMP_ChipXWaitStatesDataSetupSelect	Configures the data wait states (before the data transfer) needed to be used with the PMP module for the specified device.
	PLIB_PMP_ChipXWaitStatesStrobeSelect	Configures the strobe wait states (during the data transfer) needed to be used with the PMP module for the specified device.
	PLIB_PMP_WaitStatesAddressHoldStrobeSelect	Configures the address wait states needed to be used with the PMP module.
	PLIB_PMP_WaitStatesAddressLatchStrobeSelect	Configures the address wait states needed to be used with the PMP module.

Error Status/Control Functions

	Name	Description
	PLIB_PMP_InputOverflowClear	Clears a PMP Overflow error.
	PLIB_PMP_OutputUnderflowClear	Clears a PMP output underflow error.

Feature Existence Functions

	Name	Description
	<u>PLIB_PMP_ExistsAddressControl</u>	Identifies that the AddressControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsAddressLatchPolarity</u>	Identifies that the AddressLatchPolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsAddressLatchStrobePortControl</u>	Identifies that the AddressLatchStrobePortControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsAddressPortPinControl</u>	Identifies that the AddressPortPinControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsAltMasterRequest</u>	Identifies that the AltMasterRequest feature exists on the PMP module.
	<u>PLIB_PMP_ExistsBufferOverFlow</u>	Identifies that the BufferOverFlow feature exists on the PMP module.
	<u>PLIB_PMP_ExistsBufferRead</u>	Identifies that the BufferRead feature exists on the PMP module.
	<u>PLIB_PMP_ExistsBufferType</u>	Identifies that the BufferType feature exists on the PMP module.
	<u>PLIB_PMP_ExistsBufferUnderFlow</u>	Identifies that the BufferUnderFlow feature exists on the PMP module.
	<u>PLIB_PMP_ExistsBufferWrite</u>	Identifies that the BufferWrite feature exists on the PMP module.
	<u>PLIB_PMP_ExistsBusKeeper</u>	Identifies that the BusKeeper feature exists on the PMP module.
	<u>PLIB_PMP_ExistsBusyStatus</u>	Identifies that the BusyStatus feature exists on the PMP module.
	<u>PLIB_PMP_ExistsByteEnablePolarity</u>	Identifies that the ByteEnablePolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsByteEnablePortControl</u>	Identifies that the ByteEnablePortControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipSelectEnable</u>	Identifies that the ChipSelectEnable feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipSelectoperation</u>	Identifies that the ChipSelectoperation feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXACKMode</u>	Identifies that the ChipXACKMode feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXACKPolarity</u>	Identifies that the ChipXACKPolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXAltMasterWaitStates</u>	Identifies that the ChipXAltMasterWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXBaseAddress</u>	Identifies that the ChipXBaseAddress feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXByteEnablePolarity</u>	Identifies that the ChipXByteEnablePolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXDataHoldWaitStates</u>	Identifies that the ChipXDataHoldWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXDataSetupWaitStates</u>	Identifies that the ChipXDataSetupWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXDataSize</u>	Identifies that the ChipXDataSize feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXEnableStorbeSelect</u>	Identifies that the ChipXEnableStorbeSelect feature exists on the PMP module.

	<u>PLIB_PMP_ExistsChipXPolarity</u>	Identifies that the ChipXPolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXReadWritePolarity</u>	Identifies that the ChipXReadWritePolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXStrobeWaitStates</u>	Identifies that the ChipXStrobeWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXWriteEnablePolarity</u>	Identifies that the ChipXWriteEnablePolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsCSXActiveStatus</u>	Identifies that the CSXActiveStatus feature exists on the PMP module.
	<u>PLIB_PMP_ExistsCSXPortControl</u>	Identifies that the CSXPortControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsCurrentMaster</u>	Identifies that the CurrentMaster feature exists on the PMP module.
	<u>PLIB_PMP_ExistsDataHoldWaitStates</u>	Identifies that the DataHoldWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsDataSetUpWaitStates</u>	Identifies that the DataSetUpWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsDataStrobeWaitStates</u>	Identifies that the DataStrobeWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsDataTransferSize</u>	Identifies that the DataTransferSize feature exists on the PMP module.
	<u>PLIB_PMP_ExistsEnableControl</u>	Identifies that the EnableControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsEnhancedMasterMode</u>	Identifies that the EnhancedMasterMode feature exists on the PMP module.
	<u>PLIB_PMP_ExistsIncrementMode</u>	Identifies that the IncrementMode feature exists on the PMP module.
	<u>PLIB_PMP_ExistsInputBufferFull</u>	Identifies that the InputBufferFull feature exists on the PMP module.
	<u>PLIB_PMP_ExistsInputBufferXStatus</u>	Identifies that the InputBufferXStatus feature exists on the PMP module.
	<u>PLIB_PMP_ExistsInterruptMode</u>	Identifies that the InterruptMode feature exists on the PMP module.
	<u>PLIB_PMP_ExistsMasterRXTX</u>	Identifies that the MasterRXTX feature exists on the PMP module.
	<u>PLIB_PMP_ExistsMUXModeSelect</u>	Identifies that the MUXModeSelect feature exists on the PMP module.
	<u>PLIB_PMP_ExistsOperationMode</u>	Identifies that the OperationMode feature exists on the PMP module.
	<u>PLIB_PMP_ExistsOutPutBufferEmpty</u>	Identifies that the OutPutBufferEmpty feature exists on the PMP module.
	<u>PLIB_PMP_ExistsOutputBufferXStatus</u>	Identifies that the OutputBufferXStatus feature exists on the PMP module.
	<u>PLIB_PMP_ExistsReadWritePolarity</u>	Identifies that the ReadWritePolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsReadWriteStrobePortControl</u>	Identifies that the ReadWriteStrobePortControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsReservedAddrSpace</u>	Identifies that the ReservedAddrSpace feature exists on the PMP module.
	<u>PLIB_PMP_ExistsSlaveRX</u>	Identifies that the SlaveRX feature exists on the PMP module.

	PLIB_PMP_ExistsSlaveTX	Identifies that the SlaveTX feature exists on the PMP module.
	PLIB_PMP_ExistsSmartAddress	Identifies that the SmartAddress feature exists on the PMP module.
	PLIB_PMP_ExistsStopInIdleControl	Identifies that the StopInIdleControl feature exists on the PMP module.
	PLIB_PMP_ExistsTransactionError	Identifies that the TransactionError feature exists on the PMP module.
	PLIB_PMP_ExistsTransactionTimeOut	Identifies that the TransactionTimeOut feature exists on the PMP module.
	PLIB_PMP_ExistsWaitStatesAddrHoldStrobe	Identifies that the WaitStatesAddrHoldStrobe feature exists on the PMP module.
	PLIB_PMP_ExistsWaitStatesAddrLatchStrobe	Identifies that the WaitStatesAddrLatchStrobe feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePolarity	Identifies that the WriteEnablePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePortControl	Identifies that the WriteEnablePortControl feature exists on the PMP module.

General Initialization Functions

	Name	Description
	PLIB_PMP_AddressIncrementModeGet	Gets the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressIncrementModeSelect	Configures the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressLatchStrobeDisable	Disables the specific address latch strobe.
	PLIB_PMP_AddressLatchStrobeEnable	Enables the specific address latch strobe.
	PLIB_PMP_ChipSelectFunctionSelect	Defines the functionality of the pins intended to be used as Chip Select.
	PLIB_PMP_ChipSelectXDisable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXEnable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXIsActive	Gets the current status of the specified Chip Select.
	PLIB_PMP_DataSizeSelect	Enables data transfer size for the PMP module.
	PLIB_PMP_Disable	Disables the specific PMP module.
	PLIB_PMP_Enable	Enables the specific PMP module.
	PLIB_PMP_InputBufferTypeSelect	Selects the input buffer based on the input passed.
	PLIB_PMP_InterruptModeGet	Gets the current configured interrupt mode being used with the PMP module.
	PLIB_PMP_InterruptModeSelect	Configures the interrupt request mode being used with the PMP module.
	PLIB_PMP_MultiplexModeGet	Gets the current multiplexing mode configured between the address and data of the PMP module.
	PLIB_PMP_MultiplexModeSelect	Configures the multiplexing between the address and data of the PMP module.
	PLIB_PMP_OperationModeGet	Gets the current operation mode of the PMP module.
	PLIB_PMP_OperationModeSelect	Configures the operation mode of the PMP module.
	PLIB_PMP_StopInIdleDisable	Enables the PMP module to continue operation in Idle mode.
	PLIB_PMP_StopInIdleEnable	Discontinues PMP module operation when the device enters Idle mode.

General Status Functions

	Name	Description
≡	PLIB_PMP_IsEnabled	Checks whether or not the PMP module is enabled.
≡	PLIB_PMP_PortsBusy	Identifies if the (Master mode) PMP port is busy.
≡	PLIB_PMP_InputOverflowHasOccurred	Identifies if there was a receiver overflow error.
≡	PLIB_PMP_OutputUnderflowHasOccurred	Identifies if there was an output buffer sent out with no data.

Polarity Configuration Functions

	Name	Description
≡	PLIB_PMP_AddressLatchPolaritySelect	Sets the address latch strobe polarity.
≡	PLIB_PMP_ByteEnablePolaritySelect	Sets the byte enable polarity.
≡	PLIB_PMP_ChipSelectXPolaritySelect	Sets the specified Chip Select polarity.
≡	PLIB_PMP_ReadWriteStrobePolaritySelect	Sets the polarity of the read strobe.
≡	PLIB_PMP_WriteEnableStrobePolaritySelect	Sets the polarity of the write enable strobe.

Port Configuration Functions

	Name	Description
≡	PLIB_PMP_AddressPortDisable	Disables the port lines specified as PMP address lines.
≡	PLIB_PMP_AddressPortEnable	Enables the port lines specified as PMP address lines.
≡	PLIB_PMP_ByteEnablePortDisable	Disables the byte enable port.
≡	PLIB_PMP_ByteEnablePortEnable	Enables the byte enable port.
≡	PLIB_PMP_ReadWriteStrobePortDisable	Disables the PMP module read strobe port.
≡	PLIB_PMP_ReadWriteStrobePortEnable	Enables the PMP module read strobe port.
≡	PLIB_PMP_WriteEnableStrobePortDisable	Disables the PMP module write strobe port.
≡	PLIB_PMP_WriteEnableStrobePortEnable	Enables the PMP module write strobe port.

Wait States Initialization Functions

	Name	Description
≡	PLIB_PMP_WaitStatesDataHoldSelect	Configures the data hold states (after data transfer) needed to be used with the PMP module.
≡	PLIB_PMP_WaitStatesDataSetUpSelect	Configures the data wait states (before the data transfer) needed to be used with the PMP module.
≡	PLIB_PMP_WaitStatesStrobeSelect	Configures the strobe wait states (during the data transfer) needed to be used with the PMP module.

Description

The PMP peripheral library consists of a set of interface routines, data types and constants, and macros provided by the [plib_pmp.h](#) header file. This section describes those items.

5.6.16.6.1 General Initialization Functions**5.6.16.6.1.1 PLIB_PMP_AddressIncrementModeGet Function****C**

```
PMP_INCREMENT_MODE PLIB_PMP_AddressIncrementModeGet (
    PMP_MODULE_ID index
);
```

Description

This function gets the pins used by the PMP module. Refer to the enumeration [PMP_INCREMENT_MODE](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

[PMP_INCREMENT_MODE](#) - One of the possible values from [PMP_INCREMENT_MODE](#)

Remarks

This function implements an operation of the IncrementMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsIncrementMode](#) in your application to automatically determine whether this feature is available.

Example

```
PMP_INCREMENT_MODE curAddressIncMode;  
curAddressIncMode = PLIB_PMP_AddressIncrementModeGet( PMP_ID_0 );
```

5.6.16.6.1.2 PLIB_PMP_AddressIncrementModeSelect Function**C**

```
void PLIB_PMP_AddressIncrementModeSelect(  
    PMP_MODULE_ID index,  
    PMP_INCREMENT_MODE incrementMode  
);
```

Description

This function configures the pins used by the PMP module. Refer to the enumeration [PMP_INCREMENT_MODE](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
incrementMode	One of the possible values from PMP_INCREMENT_MODE

Returns

None.

Remarks

This function implements an operation of the IncrementMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsIncrementMode](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_AddressIncrementModeSelect( PMP_ID_0, PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE );
```

5.6.16.6.1.3 PLIB_PMP_AddressLatchStrobeDisable Function

C

```
void PLIB_PMP_AddressLatchStrobeDisable(  
    PMP_MODULE_ID index,  
    PMP_ADDRESS_LATCH latch  
);
```

Description

This function disables the PMP module with a specific address latch strobe depending on which strobes are not needed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
latch	Identifier for the latch to be disabled

Returns

None.

Remarks

This function implements an operation of the AddressLatchStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAddressLatchStrobePortControl](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_AddressLatchStrobeDisable( PMP_ID_0, PMP_ADDRESS_LATCH_UPPER );  
PLIB_PMP_AddressLatchStrobeDisable( PMP_ID_0, PMP_ADDRESS_LATCH_HIGH );  
PLIB_PMP_AddressLatchStrobeDisable( PMP_ID_0, PMP_ADDRESS_LATCH_LOW );
```

5.6.16.6.1.4 PLIB_PMP_AddressLatchStrobeEnable Function

C

```
void PLIB_PMP_AddressLatchStrobeEnable(  
    PMP_MODULE_ID index,  
    PMP_ADDRESS_LATCH latch  
);
```

Description

This function enables the PMP module with a specific address latch strobe depending on which strobes are needed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
latch	Identifier for the latch to be enabled

Returns

None.

Remarks

This function implements an operation of the AddressLatchStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAddressLatchStrobePortControl](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_AddressLatchStrobeEnable( PMP_ID_0, PMP_ADDRESS_LATCH_UPPER );
PLIB_PMP_AddressLatchStrobeEnable( PMP_ID_0, PMP_ADDRESS_LATCH_HIGH );
PLIB_PMP_AddressLatchStrobeEnable( PMP_ID_0, PMP_ADDRESS_LATCH_LOW );
```

5.6.16.6.1.5 PLIB_PMP_ChipSelectFunctionSelect Function**C**

```
void PLIB_PMP_ChipSelectFunctionSelect(
    PMP_MODULE_ID index,
    PMP_CHIPSELECT_FUNCTION chipselectfunc
);
```

Description

This function selects the PMCS1/PMCS2 as either Chip Select or as address lines depending on the way these bits are programmed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipselectfunc	One of the possible values from PMP_CHIPSELECT_FUNCTION

Returns

None.

Remarks

This function implements an operation of the ChipSelectoperation feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipSelectoperation](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipSelectFunctionSelect( PMP_ID_0, PMCS1_PMCS2_AS_ADDRESS_LINES );
```

5.6.16.6.1.6 PLIB_PMP_ChipSelectXDisable Function**C**

```
void PLIB_PMP_ChipSelectXDisable(
    PMP_MODULE_ID index,
    PMP_CHIP_SELECT chipSelect
);
```

Description

This function configures the Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as the address pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Returns

None.

Remarks

This function implements an operation of the ChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipSelectEnable](#) in your application to automatically determine whether this feature is available.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;  
PLIB_PMP_ChipSelectXDisable( PMP_ID_0, chipSelect );
```

5.6.16.6.1.7 PLIB_PMP_ChipSelectXEnable Function**C**

```
void PLIB_PMP_ChipSelectXEnable(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect  
);
```

Description

This function configures the Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as ChipSelect.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Returns

None.

Remarks

This function implements an operation of the ChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipSelectEnable](#) in your application to automatically determine whether this feature is available.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;  
PLIB_PMP_ChipSelectXEnable( PMP_ID_0, chipSelect );
```

5.6.16.6.1.8 PLIB_PMP_ChipSelectXIsActive Function

C

```
bool PLIB_PMP_ChipSelectXIsActive(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect  
);
```

Description

This function gets the current status of the specified Chip Select.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for the Chip Select to be checked

Returns

- true - Chip Select is active
- false - Chip Select is not active

Remarks

This function implements an operation of the CSXActiveStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsCSXActiveStatus](#) in your application to automatically determine whether this feature is available.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;  
if(PLIB_PMP_ChipSelectXIsActive( PMP_ID_0, chipSelect ))  
{  
    //Do something useful  
}
```

5.6.16.6.1.9 PLIB_PMP_DataSizeSelect Function

C

```
void PLIB_PMP_DataSizeSelect(  
    PMP_MODULE_ID index,  
    PMP_DATA_SIZE size  
);
```

Description

This function enables 4-bit, 8-bit, or 16-bit data transfer for the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
size	Identifier for the data size to be used

Returns

None.

Remarks

This function implements an operation of the DataTransferSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsDataTransferSize](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_DataSizeSelect( PMP_ID_0, PMP_DATA_SIZE_8_BITS );
```

5.6.16.6.1.10 PLIB_PMP_Disable Function**C**

```
void PLIB_PMP_Disable(  
    PMP_MODULE_ID index  
);
```

Description

This function disables the specific PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

The default state after any reset for a PMP module is the disable state. If the PMP is disabled, all the related pins are in control of the general purpose I/O logic.

Disabling the PMP module resets the buffers to empty states. Any data characters in the buffers are lost. All error and status bits are also reset.

Disabling the PMP while the PMP is active, will abort all pending transmissions and receptions. Re-enabling the PMP will restart the module in the same configuration.

When disabled, the PMP power consumption is minimal.

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsEnableControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_Disable( PMP_ID_0 );
```

5.6.16.6.1.11 PLIB_PMP_Enable Function**C**

```
void PLIB_PMP_Enable(  
    PMP_MODULE_ID index  
);
```

Description

This function enables the specific PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsEnableControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_Enable( PMP_ID_0 );
```

5.6.16.6.1.12 PLIB_PMP_InputBufferTypeSelect Function**C**

```
void PLIB_PMP_InputBufferTypeSelect(  
    PMP_MODULE_ID index,  
    PMP_INPUT_BUFFER_TYPE inputBuffer  
);
```

Description

This function selects the input buffer based on the input passed. Either TTL or Schmitt Trigger input buffers are selected.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
inputBuffer	One of the possible input buffer types

Returns

None.

Remarks

This function implements an operation of the BufferType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBufferType](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_InputBufferTypeSelect( PMP_ID_0, PMP_INPUT_BUFFER_TTL );
```

5.6.16.6.1.13 PLIB_PMP_InterruptModeGet Function

C

```
PMP_INTERRUPT_MODE PLIB_PMP_InterruptModeGet(  
    PMP_MODULE_ID index  
);
```

Description

This function gets the current configured interrupt mode being used with the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

One of the possible values from [PMP_INTERRUPT_MODE](#).

Remarks

This function implements an operation of the InterruptMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsInterruptMode](#) in your application to to automatically determine whether this feature is available.

Example

```
PMP_INTERRUPT_MODE currentIntMode;  
currentIntMode = PLIB_PMP_InterruptModeGet ( PMP_ID_0 );
```

5.6.16.6.1.14 PLIB_PMP_InterruptModeSelect Function

C

```
void PLIB_PMP_InterruptModeSelect(  
    PMP_MODULE_ID index,  
    PMP_INTERRUPT_MODE interruptMode  
);
```

Description

This function configures the pins used by the PMP module. Refer to the enumeration [PMP_INTERRUPT_MODE](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
interruptMode	One of the possible values from PMP_INTERRUPT_MODE

Returns

None.

Remarks

This function implements an operation of the InterruptMode feature. This feature may not be available on all devices. Please

refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsInterruptMode](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_InterruptModeSelect( PMP_ID_0, PMP_INTERRUPT_NONE );
```

5.6.16.6.1.15 PLIB_PMP_MultiplexModeGet Function

C

```
PMP_MUX_MODE PLIB_PMP_MultiplexModeGet(
    PMP_MODULE_ID index
);
```

Description

This function gets the current multiplexing mode configured between the address and data of the PMP module.

Preconditions

None.

Returns

index - Identifier for the device instance to be addressed [PMP_MUX_MODE](#) - One of the possible values from [PMP_MUX_MODE](#)

Remarks

This function implements an operation of the MUXModeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsMUXModeSelect](#) in your application to to automatically determine whether this feature is available.

Example

```
PMP_MUX_MODE currentMuxMode;
currentMuxMode = PLIB_PMP_MultiplexModeGet( PMP_ID_0 );
```

5.6.16.6.1.16 PLIB_PMP_MultiplexModeSelect Function

C

```
void PLIB_PMP_MultiplexModeSelect(
    PMP_MODULE_ID index,
    PMP_MUX_MODE multiplexMode
);
```

Description

This function configures the pins used by the PMP module. Refer to the enumeration [PMP_MUX_MODE](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
multiplexMode	One of the possible values from the PMP_MUX_MODE

Returns

None.

Remarks

This function implements an operation of the MUXModeSelect feature. This feature may not be available on all devices. Please

refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsMUXModeSelect](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_MultiplexModeSelect( PMP_ID_0, PMP_MUX_NONE );
```

5.6.16.6.1.17 PLIB_PMP_OperationModeGet Function**C**

```
PMP_OPERATION_MODE PLIB_PMP_OperationModeGet(  
    PMP_MODULE_ID index  
);
```

Description

This function gets the current operation mode of the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

[PMP_OPERATION_MODE](#) - One of the possible values from [PMP_OPERATION_MODE](#)

Remarks

This function implements an operation of the OperationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsOperationMode](#) in your application to to automatically determine whether this feature is available.

Example

```
PMP_OPERATION_MODE curOpMode;  
curOpMode = PLIB_PMP_OperationModeGet( PMP_ID_0 );
```

5.6.16.6.1.18 PLIB_PMP_OperationModeSelect Function**C**

```
void PLIB_PMP_OperationModeSelect(  
    PMP_MODULE_ID index,  
    PMP_OPERATION_MODE operationMode  
);
```

Description

This function configures operation mode of the PMP module. Refer to the enumeration [PMP_OPERATION_MODE](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
operationMode	One of the possible values from PMP_OPERATION_MODE

Returns

None.

Remarks

This function implements an operation of the OperationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsOperationMode](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_OperationModeSelect( PMP_ID_0,  
                              PMP_MASTER_READ_WRITE_STROBES_MULTIPLEXED );
```

5.6.16.6.1.19 PLIB_PMP_StopInIdleDisable Function**C**

```
void PLIB_PMP_StopInIdleDisable(  
    PMP_MODULE_ID index  
);
```

Description

This function disables the stop in idle flag. The PMP module continues operation in Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

By default, the PMP module will continue operation in Idle mode.

This function implements an operation of the StopInIdleControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsStopInIdleControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_StopInIdleDisable( PMP_ID_0 );
```

5.6.16.6.1.20 PLIB_PMP_StopInIdleEnable Function**C**

```
void PLIB_PMP_StopInIdleEnable(  
    PMP_MODULE_ID index  
);
```

Description

This function enables the PMP module to discontinue operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the StopInIdleControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsStopInIdleControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_StopInIdleEnable( PMP_ID_0 );
```

5.6.16.6.2 Enhanced General Initialization Functions

5.6.16.6.2.1 PLIB_PMP_BusKeeperDisable Function

C

```
void PLIB_PMP_BusKeeperDisable(  
    PMP_MODULE_ID index  
);
```

Description

The data bus can be kept in a high-impedance state when not actively being driven. This function disables this "bus keeper" state.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the BusKeeper feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBusKeeper](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_BusKeeperDisable ( PMP_ID_0 );
```

5.6.16.6.2.2 PLIB_PMP_BusKeeperEnable Function

C

```
void PLIB_PMP_BusKeeperEnable(  
    PMP_MODULE_ID index  
);
```

Description

The data bus can be kept in a high-impedance state when not actively being driven. This function enables this "bus keeper" state.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the BusKeeper feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBusKeeper](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_BusKeeperEnable ( PMP_ID_0 );
```

5.6.16.6.2.3 PLIB_PMP_ChipSelectXEnableStrobeDeSelect Function**C**

```
void PLIB_PMP_ChipSelectXEnableStrobeDeSelect(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect  
);
```

Description

This function selects the read and write strobes. The read and write strobes have independent lines.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	One of the possible values from PMP_CHIP_SELECT

Returns

None.

Remarks

This function implements an operation of the ChipXEnableStorbeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXEnableStorbeSelect](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipSelectXEnableStrobeDeSelect ( PMP_ID_0, PMP_CHIP_SELECT_ONE );
```

5.6.16.6.2.4 PLIB_PMP_ChipSelectXEnableStrobeSelect Function

C

```
void PLIB_PMP_ChipSelectXEnableStrobeSelect(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect  
);
```

Description

This function selects the read/write and enable strobes. The read and write strobes share a common line. A separate enable line is provided to differentiate between read and write operations.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	One of the possible values from PMP_CHIP_SELECT

Returns

None.

Remarks

This function implements an operation of the ChipXEnableStrobeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXEnableStrobeSelect](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipSelectXEnableStrobeSelect ( PMP_ID_0, PMP_CHIP_SELECT_ONE );
```

5.6.16.6.2.5 PLIB_PMP_SmartAddressStrobeDisable Function

C

```
void PLIB_PMP_SmartAddressStrobeDisable(  
    PMP_MODULE_ID index  
);
```

Description

This function disables "smart" address strobes (each address phase is only present if the current access would cause a different address in the latch than the previous address).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the SmartAddress feature. This feature may not be available on all devices. Please

refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsSmartAddress](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_SmartAddressStrobeDisable ( PMP_ID_0 );
```

5.6.16.6.2.6 PLIB_PMP_SmartAddressStrobeEnable Function**C**

```
void PLIB_PMP_SmartAddressStrobeEnable(  
    PMP_MODULE_ID index  
);
```

Description

This function enables "smart" address strobes (each address phase is only present if the current access would cause a different address in the latch than the previous address).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the SmartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsSmartAddress](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_SmartAddressStrobeEnable ( PMP_ID_0 );
```

5.6.16.6.2.7 PLIB_PMP_ChipXAckModeSelect Function**C**

```
void PLIB_PMP_ChipXAckModeSelect(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect,  
    PMP_ACK_MODE mode  
);
```

Description

This function sets the acknowledge for the PMP module. Either enables, disables, or enables it with time-out. See the [PMP_ACK_MODE](#) enumeration for additional information.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
mode	Identifier for the acknowledge mode to be selected

chipSelect	One of the possible values from PMP_CHIP_SELECT

Returns

None.

Remarks

This function implements an operation of the ChipXACKMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXACKMode](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipXAckModeSelect ( PMP_ID_0, PMP_CHIP_SELECT_ONE, PMP_ACK_DISABLED );
```

5.6.16.6.2.8 PLIB_PMP_ChipXDataSizeSelect Function

C

```
void PLIB_PMP_ChipXDataSizeSelect(
    PMP_MODULE_ID index,
    PMP_CHIP_SELECT chipSelect,
    PMP_DATA_SIZE size
);
```

Description

This function enables 4-bit, 8-bit, or 16-bit data transfer for the PMP for the specified device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
size	Identifier for the data size to be used
chipSelect	One of the possible values from PMP_CHIP_SELECT

Returns

None.

Remarks

This function implements an operation of the ChipXDataSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXDataSize](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipXDataSizeSelect( PMP_ID_0,
                               PMP_CHIP_SELECT_ONE,
                               PMP_DATA_SIZE_8_BITS );
```

5.6.16.6.2.9 PLIB_PMP_EnhancedMasterModeSelect Function

C

```
void PLIB_PMP_EnhancedMasterModeSelect(
    PMP_MODULE_ID index,
    PMP_MASTER_MODE master
);
```

Description

This function sets the PMP module for Master mode (either CPU or alternate master). The alternate master can have direct I/O access.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
master	Identifier for the master to be selected

Returns

None.

Remarks

This function implements an operation of the EnhancedMasterMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsEnhancedMasterMode](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_EnhancedMasterModeSelect ( PMP_ID_0, PMP_CPU_MASTER );
```

5.6.16.6.3 General Status Functions

5.6.16.6.3.1 PLIB_PMP_IsEnabled Function

C

```
bool PLIB_PMP_IsEnabled(  
    PMP_MODULE_ID index  
);
```

Description

This function checks whether or not the PMP module is enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If the PMP module is enabled
- false - if the PMP module is disabled

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsEnableControl](#) in your application to to automatically determine whether this feature is available.

Example

```
bool pmpStatus;  
pmpStatus = PLIB_PMP_IsEnabled( PMP_ID_0 );
```

5.6.16.6.3.2 PLIB_PMP_PortIsBusy Function**C**

```
bool PLIB_PMP_PortIsBusy(  
    PMP_MODULE_ID index  
);
```

Description

This function identifies if the PMP port is busy (in Master mode).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If the port is busy
- false - If the port is not busy

Remarks

Works only in Master mode. This function implements an operation of the BusyStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBusyStatus](#) in your application to automatically determine whether this feature is available.

Example

```
bool status;  
status = PLIB_PMP_PortIsBusy( PMP_ID_0 );
```

5.6.16.6.3.3 PLIB_PMP_InputOverflowHasOccurred Function**C**

```
bool PLIB_PMP_InputOverflowHasOccurred(  
    PMP_MODULE_ID index  
);
```

Description

This function identifies if there was a receiver overflow error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If the input buffer has overflowed
- false - If the input buffer has not overflowed

Remarks

This function implements an operation of the BufferOverflow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBufferOverflow](#) in your application to automatically determine whether this feature is available.

Example

```
if(PLIB_PMP_InputOverflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_InputOverflowClear( PMP_ID_0 );
}
```

5.6.16.6.3.4 PLIB_PMP_OutputUnderflowHasOccurred Function**C**

```
bool PLIB_PMP_OutputUnderflowHasOccurred(
    PMP_MODULE_ID index
);
```

Description

This function identifies if there was a output buffer was sent out with no data.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If the input buffer was empty when data was sent
- false - If the output buffer was not empty when data was sent

Remarks

This function implements an operation of the BufferUnderFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBufferUnderFlow](#) in your application to automatically determine whether this feature is available.

Example

```
if(PLIB_PMP_OutputUnderflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_OutputUnderflowClear( PMP_ID_0 );
}
```

5.6.16.6.4 Enhanced General Status Functions**5.6.16.6.4.1 PLIB_PMP_AlternateMasterHasAccess Function****C**

```
bool PLIB_PMP_AlternateMasterHasAccess(
    PMP_MODULE_ID index
);
```

Description

This function gets the status of the alternate masters that have gained access to the PMP module, whether or not the alternate

master has access to the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If the alternate master has gained access to the PMP module
- false - If the CPU has gained access to the PMP module

Remarks

This function implements an operation of the CurrentMaster feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsCurrentMaster](#) in your application to automatically determine whether this feature is available.

Example

```
status = PLIB_PMP_AlternateMasterHasAccess ( PMP_ID_0 );
```

5.6.16.6.4.2 PLIB_PMP_AlternateMasterRequestStatus Function

C

```
bool PLIB_PMP_AlternateMasterRequestStatus(  
    PMP_MODULE_ID index  
);
```

Description

This function gets the status of the request from the alternate master, whether or not the master is requesting use of the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If the alternate master is requesting use of the PMP module
- false - If the alternate master is not a making request for the PMP module

Remarks

This function implements an operation of the AltMasterRequest feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAltMasterRequest](#) in your application to automatically determine whether this feature is available.

Example

```
status = PLIB_PMP_AlternateMasterRequestStatus ( PMP_ID_0 );
```

5.6.16.6.5 Error Status/Control Functions

5.6.16.6.5.1 PLIB_PMP_InputOverflowClear Function

C

```
void PLIB_PMP_InputOverflowClear(  
    PMP_MODULE_ID index  
);
```

Description

This function clears an overflow error. Clearing the error resets the receive buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the BufferOverflow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBufferOverflow](#) in your application to automatically determine whether this feature is available.

Example

```
if(PLIB_PMP_InputOverflowHasOccurred( PMP_ID_0 ))  
{  
    PLIB_PMP_InputOverflowClear( PMP_ID_0 );  
}
```

5.6.16.6.5.2 PLIB_PMP_OutputUnderflowClear Function

C

```
void PLIB_PMP_OutputUnderflowClear(  
    PMP_MODULE_ID index  
);
```

Description

This function clears a PMP output underflow error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the BufferUnderFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBufferUnderFlow](#) in your application to automatically determine whether this feature is available.

Example

```
if(PLIB_PMP_OutputUnderflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_OutputUnderflowClear( PMP_ID_0 );
}
```

5.6.16.6.6 Enhanced Error/Status Control Functions

5.6.16.6.6.1 PLIB_PMP_TransactionErrorClear Function

C

```
void PLIB_PMP_TransactionErrorClear(
    PMP_MODULE_ID index
);
```

Description

This function clears the transaction error flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the TransactionError feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsTransactionError](#) in your application to automatically determine whether this feature is available.

Example

```
if(PLIB_PMP_TransactionErrorHasOccurred ( PMP_ID_0 ) )
    PLIB_PMP_TransactionErrorClear ( PMP_ID_0 );
```

5.6.16.6.6.2 PLIB_PMP_TransactionErrorHasOccurred Function

C

```
bool PLIB_PMP_TransactionErrorHasOccurred(
    PMP_MODULE_ID index
);
```

Description

This function returns the status of a transaction error if an illegal transaction was requested.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If illegal transaction was requested
- false - Transaction completed successfully

Remarks

This function implements an operation of the TransactionError feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsTransactionError](#) in your application to to automatically determine whether this feature is available.

Example

```
status = PLIB_PMP_TransactionErrorHasOccurred ( PMP_ID_0 );
```

5.6.16.6.6.3 PLIB_PMP_TransactionHasTimedOut Function**C**

```
bool PLIB_PMP_TransactionHasTimedOut(  
    PMP_MODULE_ID index  
);
```

Description

This function returns the time-out status of the PMP transaction.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If the transaction timed out
- false - If the transaction completed successfully

Remarks

This function implements an operation of the TransactionTimeout feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsTransactionTimeout](#) in your application to to automatically determine whether this feature is available.

Example

```
status = PLIB_PMP_TransactionHasTimedOut ( PMP_ID_0 );
```

5.6.16.6.6.4 PLIB_PMP_TransactionTimeoutClear Function**C**

```
void PLIB_PMP_TransactionTimeoutClear(  
    PMP_MODULE_ID index  
);
```

Description

This function clears the time-out flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the TransactionTimeout feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsTransactionTimeout](#) in your application to to automatically determine whether this feature is available.

Example

```
if(PLIB_PMP_TransactionHasTimedOut ( PMP_ID_0 ))
    PLIB_PMP_TransactionTimeoutClear ( PMP_ID_0 );
```

5.6.16.6.7 Data Read and Write Functions

5.6.16.6.7.1 PLIB_PMP_InputBuffersAreFull Function

C

```
bool PLIB_PMP_InputBuffersAreFull(
    PMP_MODULE_ID index
);
```

Description

This function gets the state of the input buffers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If all input buffers are full
- false - If all input buffers are not full

Remarks

This function implements an operation of the InputBufferFull feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsInputBufferFull](#) in your application to to automatically determine whether this feature is available.

Example

```
uint8_t value;
if(PLIB_PMP_InputBuffersAreFull( PMP_ID_0 ))
{
    value = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 1 );
}
```

5.6.16.6.7.2 PLIB_PMP_InputBufferXByteReceive Function

C

```
uint8_t PLIB_PMP_InputBufferXByteReceive(  
    PMP_MODULE_ID index,  
    uint8_t buffer  
);
```

Description

The data to be received in Byte mode from the specified PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible buffers (valid values are 0 to 3)

Returns

data - Data to be received

Remarks

This function implements an operation of the BufferRead feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBufferRead](#) in your application to automatically determine whether this feature is available.

Example

```
uint8_t mydata;  
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))  
{  
    // get data from buffer-1  
    mydata = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 1 );  
}
```

5.6.16.6.7.3 PLIB_PMP_InputBufferXIsFull Function

C

```
bool PLIB_PMP_InputBufferXIsFull(  
    PMP_MODULE_ID index,  
    uint8_t buffer  
);
```

Description

This function gets the state of the identified input buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	The input buffer number (valid values are 0 to 3)

Returns

- true - If all input buffers are full
- false - If all input buffers are not full

Remarks

This function implements an operation of the InputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsInputBufferXStatus](#) in your application to automatically determine whether this feature is available.

Example

```
uint8_t value;
// Check the status of buffer-2
if(PLIB_PMP_InputBufferXIsFull( PMP_ID_0, 2 ))
{
    // get data from buffer 2
    value = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 2 );
}
```

5.6.16.6.7.4 PLIB_PMP_IsDataReceived Function**C**

```
bool PLIB_PMP_IsDataReceived(
    PMP_MODULE_ID index,
    uint8_t buffer
);
```

Description

This function checks and returns if the data has been received in the specified buffer in Slave mode.

Preconditions

The PMP module should be configured for Slave mode operation.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible input buffers (valid values are 0 to 3)

Returns

- true - Data has been received in the specified buffer
- false - Data has not been received in the specified buffer

Remarks

This function implements an operation of the InputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsInputBufferXStatus](#) in your application to automatically determine whether this feature is available.

Example

```
int8_t data;
// Check if data is received on buffer-2
if(PLIB_PMP_IsDataReceived( PMP_ID_0, 2 ))
{
    // get data from buffer-2
    data = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 2 );
}
```


5.6.16.6.7.5 PLIB_PMP_IsDataTransmitted Function

C

```
bool PLIB_PMP_IsDataTransmitted(  
    PMP_MODULE_ID index,  
    uint8_t buffer  
);
```

Description

Checks and returns if data has been transmitted from the specified buffer.

Preconditions

The PMP module should be configured for Slave mode operation.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible output buffers (valid range is 0 to 3)

Returns

- true - If data has been transmitted from the specified buffer
- false - If data has not been transmitted from the specified buffer

Remarks

This function implements an operation of the OutputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsOutputBufferXStatus](#) in your application to automatically determine whether this feature is available.

Example

```
uint8_t data;  
if(PLIB_PMP_IsDataTransmitted( PMP_ID_0, 2 ))  
{  
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 2, data );  
}
```

5.6.16.6.7.6 PLIB_PMP_MasterReceive Function

C

```
uint16_t PLIB_PMP_MasterReceive(  
    PMP_MODULE_ID index  
);
```

Description

This function receives the data. The dataflow is from slave to master.

Preconditions

The PMP module is configured as master.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed.

Returns

uint16_t - Data received

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the MasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsMasterRXTX](#) in your application to to automatically determine whether this feature is available.

Example

```
uint16_t data;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    data = PLIB_PMP_MasterReceive( PMP_ID_0 );
}
```

5.6.16.6.7.7 PLIB_PMP_MasterSend Function

C

```
void PLIB_PMP_MasterSend(
    PMP_MODULE_ID index,
    uint16_t data
);
```

Description

This function sends the specified data. The data flow is from master to slave.

Preconditions

The PMP module is configured for Master Mode.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
data	Data to be transmitted

Returns

None.

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the MasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsMasterRXTX](#) in your application to to automatically determine whether this feature is available.

Example

```
uint16_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_MasterSend( PMP_ID_0, data );
}
```

5.6.16.6.7.8 PLIB_PMP_OutputBuffersAreEmpty Function

C

```
bool PLIB_PMP_OutputBuffersAreEmpty(
    PMP_MODULE_ID index
);
```

Description

This function gets the state of the output buffers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

- true - If all output buffers are empty
- false - If all output buffers are not empty

Remarks

This function implements an operation of the OutPutBufferEmpty feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsOutPutBufferEmpty](#) in your application to to automatically determine whether this feature is available.

Example

```
uint8_t value=0xEF;

if(PLIB_PMP_OutputBuffersAreEmpty( PMP_ID_0 ))
{
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 1, value);
}
```

5.6.16.6.7.9 PLIB_PMP_OutputBufferXByteSend Function**C**

```
void PLIB_PMP_OutputBufferXByteSend(
    PMP_MODULE_ID index,
    uint8_t buffer,
    uint8_t data
);
```

Description

The data is transmitted in Byte mode for the specified PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible output buffers (valid range is 0 to 3)
data	Data to be transmitted

Returns

None.

Remarks

This function implements an operation of the BufferWrite feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsBufferWrite](#) in your application to to automatically determine whether this feature is available.

Example

```
uint8_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 1, data );
}
```

5.6.16.6.7.10 PLIB_PMP_OutputBufferXIsEmpty Function**C**

```
bool PLIB_PMP_OutputBufferXIsEmpty(
    PMP_MODULE_ID index,
    uint8_t buffer
);
```

Description

This function gets the state of the input buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	Output buffer number (valid range is 0 to 3)

Returns

- true - If the identified output buffer is empty
- false - If the identified output buffer is not empty

Remarks

This function implements an operation of the OutputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsOutputBufferXStatus](#) in your application to to automatically determine whether this feature is available.

Example

```
uint8_t value = 0xEF;

if(PLIB_PMP_OutputBufferXIsEmpty( PMP_ID_0, 1 ) )
{
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0,1, value);
}
```

5.6.16.6.7.11 PLIB_PMP_SlaveReceive Function**C**

```
uint16_t PLIB_PMP_SlaveReceive(
    PMP_MODULE_ID index
);
```

Description

This function receives the data. The dataflow is from master to slave.

Preconditions

The PMP module is configured as slave.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed.

Returns

uint16_t - Data received

Remarks

This function to be used only when the PMP is acting as slave. This function implements an operation of the SlaveRX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsSlaveRX](#) in your application to to automatically determine whether this feature is available.

Example

```
uint16_t data;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    data = PLIB_PMP_SlaveReceive( PMP_ID_0 );
}
```

5.6.16.6.7.12 PLIB_PMP_SlaveSend Function**C**

```
void PLIB_PMP_SlaveSend(
    PMP_MODULE_ID index,
    uint16_t data
);
```

Description

This function sends the specified data. The data flow is from slave to master.

Preconditions

The PMP module is configured for Slave Mode.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
data	Data to be transmitted

Returns

None.

Remarks

This function to be used only when the PMP is acting as slave. This function implements an operation of the SlaveTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsSlaveTX](#) in your application to to automatically determine whether this feature is available.

Example

```
uint16_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_SlaveSend( PMP_ID_0, data );
}
```

5.6.16.6.8 Wait States Initialization Functions

5.6.16.6.8.1 PLIB_PMP_WaitStatesDataHoldSelect Function

C

```
void PLIB_PMP_WaitStatesDataHoldSelect(  
    PMP_MODULE_ID index,  
    PMP_DATA_HOLD_STATES dataHoldState  
);
```

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration [PMP_DATA_HOLD_STATES](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
dataHoldState	One of the possible values from PMP_DATA_HOLD_STATES

Returns

None.

Remarks

This function implements an operation of the DataHoldWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsDataHoldWaitStates](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_WaitStatesDataHoldSelect( PMP_ID_0, PMP_DATA_HOLD_2 );
```

5.6.16.6.8.2 PLIB_PMP_WaitStatesDataSetUpSelect Function

C

```
void PLIB_PMP_WaitStatesDataSetUpSelect(  
    PMP_MODULE_ID index,  
    PMP_DATA_WAIT_STATES dataWaitState  
);
```

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration [PMP_DATA_WAIT_STATES](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
dataWaitState	One of the possible values from PMP_DATA_WAIT_STATES

Returns

None.

Remarks

This function implements an operation of the DataSetUpWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsDataSetUpWaitStates](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_WaitStatesDataSetUpSelect( PMP_ID_0, PMP_DATA_WAIT_TWO );
```

5.6.16.6.8.3 PLIB_PMP_WaitStatesStrobeSelect Function**C**

```
void PLIB_PMP_WaitStatesStrobeSelect(
    PMP_MODULE_ID index,
    PMP_STROBE_WAIT_STATES strobeWaitState
);
```

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration [PMP_STROBE_WAIT_STATES](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
strobeWaitState	One of the possible values from PMP_STROBE_WAIT_STATES

Returns

None.

Remarks

This function implements an operation of the DataStrobeWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsDataStrobeWaitStates](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_WaitStatesStrobeSelect( PMP_ID_0, PMP_STROBE_WAIT_2 );
```

5.6.16.6.9 Enhanced Wait States Initialization Functions**5.6.16.6.9.1 PLIB_PMP_ChipXWaitStatesAlternateMasterSelect Function****C**

```
void PLIB_PMP_ChipXWaitStatesAlternateMasterSelect(
    PMP_MODULE_ID index,
    PMP_CHIP_SELECT chipSelect,
    PMP_ALTERNATE_MASTER_WAIT_STATES waitStates
);
```

Description

Selects the wait states of the alternate master for Chip Select-X.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
waitStates	Identifier for wait states
chipSelect	One of the possible values from PMP_CHIP_SELECT

Returns

None.

Remarks

This function implements an operation of the ChipXAltMasterWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXAltMasterWaitStates](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipXWaitStatesAlternateMasterSelect ( PMP_ID_0,  
                                                PMP_CHIP_SELECT_ONE,  
                                                PMP_ALTERNATE_MASTER_WAIT_THREE );
```

5.6.16.6.9.2 PLIB_PMP_ChipXWaitStatesDataHoldSelect Function**C**

```
void PLIB_PMP_ChipXWaitStatesDataHoldSelect(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipselect,  
    PMP_DATA_HOLD_STATES dataHoldState  
);
```

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration [PMP_DATA_HOLD_STATES](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipselect	One of the possible values from PMP_CHIP_SELECT
dataHoldState	One of the possible values from PMP_DATA_HOLD_STATES

Returns

None.

Remarks

This function implements an operation of the ChipXDataHoldWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXDataHoldWaitStates](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipXWaitStatesDataHoldSelect( PMP_ID_0,
                                         PMP_CHIP_SELECT_ONE,
                                         PMP_DATA_HOLD_2 );
```

5.6.16.6.9.3 PLIB_PMP_ChipXWaitStatesDataSetupSelect Function**C**

```
void PLIB_PMP_ChipXWaitStatesDataSetupSelect(
    PMP_MODULE_ID index,
    PMP_CHIP_SELECT chipselect,
    PMP_DATA_WAIT_STATES dataWaitState
);
```

Description

This function configures the number of peripheral bus clock cycles needed for wait states for the specified device. Refer to the enumeration [PMP_DATA_WAIT_STATES](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipselect	One of the possible values from PMP_CHIP_SELECT
dataWaitState	One of the possible values from PMP_DATA_WAIT_STATES

Returns

None.

Remarks

This function implements an operation of the ChipXDataSetupWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXDataSetupWaitStates](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipXWaitStatesDataSetupSelect( PMP_ID_0,
                                         PMP_CHIP_SELECT_ONE,
                                         PMP_DATA_WAIT_ONE );
```

5.6.16.6.9.4 PLIB_PMP_ChipXWaitStatesStrobeSelect Function**C**

```
void PLIB_PMP_ChipXWaitStatesStrobeSelect(
    PMP_MODULE_ID index,
    PMP_CHIP_SELECT chipselect,
    PMP_STROBE_WAIT_STATES strobeWaitState
);
```

Description

This function configures the number of peripheral bus clock cycles needed for wait states for the specified device. Refer to the enumeration [PMP_STROBE_WAIT_STATES](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipselect	One of the possible values from PMP_CHIP_SELECT
strobeWaitState	One of the possible values from PMP_STROBE_WAIT_STATES

Returns

None.

Remarks

This function implements an operation of the ChipXStrobeWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXStrobeWaitStates](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipXWaitStatesStrobeSelect( PMP_ID_0,  
                                      PMP_CHIP_SELECT_ONE,  
                                      PMP_STROBE_WAIT_3 );
```

5.6.16.6.9.5 PLIB_PMP_WaitStatesAddressHoldStrobeSelect Function**C**

```
void PLIB_PMP_WaitStatesAddressHoldStrobeSelect(  
    PMP_MODULE_ID index,  
    PMP_ADDRESS_HOLD_LATCH_WAIT_STATES waitState  
);
```

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration [PMP_ADDRESS_HOLD_LATCH_WAIT_STATES](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
waitState	One of the possible values from PMP_ADDRESS_HOLD_LATCH_WAIT_STATES

Returns

None.

Remarks

This function implements an operation of the WaitStatesAddrHoldStrobe feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsWaitStatesAddrHoldStrobe](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_WaitStatesAddressHoldStrobeSelect( PMP_ID_0,  
                                             PMP_ADDRESS_HOLD_ONE_AND_ONE_QUARTER );
```

5.6.16.6.9.6 PLIB_PMP_WaitStatesAddressLatchStrobeSelect Function

C

```
void PLIB_PMP_WaitStatesAddressLatchStrobeSelect(  
    PMP_MODULE_ID index,  
    PMP_ADDRESS_LATCH_WAIT_STATES waitState  
);
```

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration [PMP_ADDRESS_LATCH_WAIT_STATES](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
waitState	One of the possible values from PMP_ADDRESS_LATCH_WAIT_STATES

Returns

None.

Remarks

This function implements an operation of the WaitStatesAddrLatchStrobe feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsWaitStatesAddrLatchStrobe](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_WaitStatesAddressLatchStrobeSelect( PMP_ID_0,  
                                              PMP_ADDRESS_WAIT_ONE_AND_HALF );
```

5.6.16.6.10 Address Handling Functions

5.6.16.6.10.1 PLIB_PMP_AddressGet Function

C

```
uint32_t PLIB_PMP_AddressGet(  
    PMP_MODULE_ID index  
);
```

Description

This function gets the current address of the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

address - Device address to be set

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAddressControl](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t address;  
address = PLIB_PMP_AddressGet( PMP_ID_0 );
```

5.6.16.6.10.2 PLIB_PMP_AddressLinesA0A1Get Function**C**

```
uint8_t PLIB_PMP_AddressLinesA0A1Get(  
    PMP_MODULE_ID index  
);
```

Description

This function gets the value of the address lines PMA0 and PMA1. This function is used in the addressable parallel slave port mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

uint8_t - The two-bit address

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAddressControl](#) in your application to automatically determine whether this feature is available.

Example

```
uint8_t bufferAddress;  
bufferAddress = PLIB_PMP_AddressLinesA0A1Get( PMP_ID_0 );
```

5.6.16.6.10.3 PLIB_PMP_AddressLinesA0A1Set Function**C**

```
void PLIB_PMP_AddressLinesA0A1Set(  
    PMP_MODULE_ID index,  
    uint8_t address  
);
```

Description

This function sets the address lines PMA0 and PMA1 with the value specified. This function is used in the addressable parallel slave port mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
address	The two-bit address

Returns

None.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAddressControl](#) in your application to automatically determine whether this feature is available.

Example

```
uint8_t bufferAddress = 0x2;  
PLIB_PMP_AddressLinesA0A1Set( PMP_ID_0, bufferAddress );
```

5.6.16.6.10.4 PLIB_PMP_AddressSet Function**C**

```
void PLIB_PMP_AddressSet(  
    PMP_MODULE_ID index,  
    uint32_t address  
);
```

Description

This function sets the current address of the PMP module to the specified value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
address	Device address to be set

Returns

None.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAddressControl](#) in your application to automatically determine whether this feature is available.

Example

```
uint8_t no_of_addressLines = 8;  
PLIB_PMP_AddressSet( PMP_ID_0, 0xff );
```

5.6.16.6.11 Enhanced Address Handling Functions

5.6.16.6.11.1 PLIB_PMP_ChipXBaseAddressSet Function

C

```
void PLIB_PMP_ChipXBaseAddressSet(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect,  
    uint16_t baseAddress  
);
```

Description

This function sets the base address of the specified device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
baseAddress	Base address of the chip
chipSelect	One of the possible values from PMP_CHIP_SELECT

Returns

None.

Remarks

This function implements an operation of the ChipXBaseAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXBaseAddress](#) in your application to automatically determine whether this feature is available.

Example

```
uint16_t baseAddress = 0x00FF;  
PLIB_PMP_ChipXBaseAddressSet( PMP_ID_0, PMP_CHIP_SELECT_ONE, baseAddress );
```

5.6.16.6.11.2 PLIB_PMP_ReservedAddressSpaceBitsSet Function

C

```
void PLIB_PMP_ReservedAddressSpaceBitsSet(  
    PMP_MODULE_ID index,  
    uint8_t reservedAddress  
);
```

Description

This function sets the address bits of the reserved address space of the EPMP. The reserved address space determines the end address of the Chip Select-2 (if used). If the reserved address space is set to 0, the last EDS address for the Chip Select-2 will be 0xFFFFF. For the address range to be accessible for the Chip Select-2 set the reserved address space to a value higher than Chip Select-2 base address.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
reservedAddress	The reserved address bits

Returns

None.

Remarks

This function implements an operation of the ReservedAddrSpace feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsReservedAddrSpace](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ReservedAddressSpaceBitsSet ( PMP_ID_0, 0xDF );
```

5.6.16.6.12 Port Configuration Functions

5.6.16.6.12.1 PLIB_PMP_AddressPortDisable Function

C

```
void PLIB_PMP_AddressPortDisable(  
    PMP_MODULE_ID index,  
    PMP_ADDRESS_PORT portfunctions  
);
```

Description

This function disables the port lines specified as PMP address lines. They function as normal I/O lines.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
portfunctions	One of the possible values from PMP_ADDRESS_PORT

Returns

None.

Remarks

This function implements an operation of the AddressPortPinControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAddressPortPinControl](#) in your application to to automatically determine whether this feature is available.

Example

```
//Example-1  
PMP_ADDRESS_PORT portfunctions = PMP_PMA2_TO_PMA13_PORTS;  
PLIB_PMP_AddressPortDisable( PMP_ID_0, PMP_PMA2_TO_PMA13_PORTS );  
  
//Example-2  
PLIB_PMP_AddressPortDisable( PMP_ID_0,  
    ( PMP_PMA14_PORT | PMP_PMA15_PORT ) );
```

5.6.16.6.12.2 PLIB_PMP_AddressPortEnable Function

C

```
void PLIB_PMP_AddressPortEnable(  
    PMP_MODULE_ID index,
```

```
    PMP_ADDRESS_PORT portfunctions  
);
```

Description

This function enables the port lines specified as PMP address lines.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
portfunctions	One of the possible values from PMP_ADDRESS_PORT

Returns

None.

Remarks

This function implements an operation of the AddressPortPinControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAddressPortPinControl](#) in your application to automatically determine whether this feature is available.

Example

```
//Example-1  
PMP_ADDRESS_PORT portfunctions = PMP_PMA2_TO_PMA13_PORTS;  
PLIB_PMP_AddressPortEnable( PMP_ID_0, PMP_PMA2_TO_PMA13_PORTS );  
  
//Example-2  
PLIB_PMP_AddressPortEnable( PMP_ID_0,  
                             ( PMP_PMA14_PORT | PMP_PMA15_PORT ) );
```

5.6.16.6.12.3 PLIB_PMP_ByteEnablePortDisable Function

C

```
void PLIB_PMP_ByteEnablePortDisable(  
    PMP_MODULE_ID index,  
    PMP_PMBE_PORT pmbeInstance  
);
```

Description

This function disables the byte enable port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
pmbeInstance	One of the possible values from PMP_PMBE_PORT

Returns

None.

Remarks

This function implements an operation of the ExistsByteEnablePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_PMP_ExistsExistsByteEnablePortControl in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ByteEnablePortDisable( PMP_ID_0, PMBE_0 );
```

5.6.16.6.12.4 PLIB_PMP_ByteEnablePortEnable Function**C**

```
void PLIB_PMP_ByteEnablePortEnable(  
    PMP_MODULE_ID index,  
    PMP_PMBE_PORT pmbeInstance  
);
```

Description

This function enables the byte enable port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
pmbeInstance	One of the possible values from PMP_PMBE_PORT

Returns

None.

Remarks

This function implements an operation of the ExistsByteEnablePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsExistsByteEnablePortControl in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ByteEnablePortEnable( PMP_ID_0, PMBE_0 );
```

5.6.16.6.12.5 PLIB_PMP_ReadWriteStrobePortDisable Function**C**

```
void PLIB_PMP_ReadWriteStrobePortDisable(  
    PMP_MODULE_ID index  
);
```

Description

This function disables the read strobe port of the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the ReadWriteStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsReadWriteStrobePortControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ReadWriteStrobePortDisable( PMP_ID_0 );
```

5.6.16.6.12.6 PLIB_PMP_ReadWriteStrobePortEnable Function**C**

```
void PLIB_PMP_ReadWriteStrobePortEnable(  
    PMP_MODULE_ID index  
);
```

Description

This function enables the read strobe port of the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the ReadWriteStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsReadWriteStrobePortControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ReadWriteStrobePortEnable( PMP_ID_0 );
```

5.6.16.6.12.7 PLIB_PMP_WriteEnableStrobePortDisable Function**C**

```
void PLIB_PMP_WriteEnableStrobePortDisable(  
    PMP_MODULE_ID index  
);
```

Description

This function disables the write strobe port of the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the WriteEnablePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsWriteEnablePortControl](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_WriteEnableStrobePortDisable( PMP_ID_0 );
```

5.6.16.6.12.8 PLIB_PMP_WriteEnableStrobePortEnable Function**C**

```
void PLIB_PMP_WriteEnableStrobePortEnable(  
    PMP_MODULE_ID index  
);
```

Description

This function enables the write strobe port of the PMP module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Returns

None.

Remarks

This function implements an operation of the WriteEnablePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsWriteEnablePortControl](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_WriteEnableStrobePortEnable( PMP_ID_0 );
```

5.6.16.6.13 Enhanced Port Configuration Functions**5.6.16.6.13.1 PLIB_PMP_ChipSelectXPortDisable Function****C**

```
void PLIB_PMP_ChipSelectXPortDisable(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect  
);
```

Description

This function disables the specified Chip Select's port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	One of the possible values from PMP_CHIP_SELECT

Returns

None.

Remarks

This function implements an operation of the CSXPortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsCSXPortControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipSelectXPortDisable ( PMP_ID_0, PMP_CHIP_SELECT_ONE );
```

5.6.16.6.13.2 PLIB_PMP_ChipSelectXPortEnable Function**C**

```
void PLIB_PMP_ChipSelectXPortEnable(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect  
);
```

Description

This function enables the specified Chip Select's port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	One of the possible values from PMP_CHIP_SELECT

Returns

None.

Remarks

This function implements an operation of the CSXPortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsCSXPortControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipSelectXPortEnable ( PMP_ID_0, PMP_CHIP_SELECT_ONE );
```

5.6.16.6.14 Polarity Configuration Functions**5.6.16.6.14.1 PLIB_PMP_AddressLatchPolaritySelect Function****C**

```
void PLIB_PMP_AddressLatchPolaritySelect(  
    PMP_MODULE_ID index,
```

```
    PMP_POLARITY_LEVEL polarity
);
```

Description

This function sets the address latch polarity to the level specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Returns

None.

Remarks

This function implements an operation of the AddressLatchPolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsAddressLatchPolarity](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_AddressLatchPolaritySelect( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

5.6.16.6.14.2 PLIB_PMP_ByteEnablePolaritySelect Function

C

```
void PLIB_PMP_ByteEnablePolaritySelect(
    PMP_MODULE_ID index,
    PMP_POLARITY_LEVEL polarity
);
```

Description

This function sets the byte enable polarity to the level specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Returns

None.

Remarks

This function implements an operation of the ByteEnablePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsByteEnablePolarity](#) in your application to to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ByteEnablePolaritySelect ( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

5.6.16.6.14.3 PLIB_PMP_ChipSelectXPolaritySelect Function

C

```
void PLIB_PMP_ChipSelectXPolaritySelect(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect,  
    PMP_POLARITY_LEVEL polarity  
);
```

Description

This function sets the specified Chip Select polarity to the level specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for Chip Select
polarity	Possible polarity levels

Returns

None.

Remarks

This function implements an operation of the ChipXPolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXPolarity](#) in your application to automatically determine whether this feature is available.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;  
PLIB_PMP_ChipSelectXPolaritySelect( PMP_ID_0,  
                                     chipSelect,  
                                     PMP_POLARITY_ACTIVE_HIGH );
```

5.6.16.6.14.4 PLIB_PMP_ReadWriteStrobePolaritySelect Function

C

```
void PLIB_PMP_ReadWriteStrobePolaritySelect(  
    PMP_MODULE_ID index,  
    PMP_POLARITY_LEVEL polarity  
);
```

Description

This function sets polarity of the read strobe to the level specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Returns

None.

Remarks

This function implements an operation of the ReadWritePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsReadWritePolarity](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ReadWriteStrobePolaritySelect( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

5.6.16.6.14.5 PLIB_PMP_WriteEnableStrobePolaritySelect Function**C**

```
void PLIB_PMP_WriteEnableStrobePolaritySelect(  
    PMP_MODULE_ID index,  
    PMP_POLARITY_LEVEL polarity  
);
```

Description

This function sets the polarity of the write enable strobe to the level specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Returns

None.

Remarks

This function implements an operation of the WriteEnablePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsWriteEnablePolarity](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_WriteEnableStrobePolaritySelect( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

5.6.16.6.15 Enhanced Polarity Configuration Functions**5.6.16.6.15.1 PLIB_PMP_ChipSelectXAckPolaritySelect Function****C**

```
void PLIB_PMP_ChipSelectXAckPolaritySelect(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect,  
    PMP_POLARITY_LEVEL polarity  
);
```

Description

This function sets the Chip Select acknowledge polarity to the level specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for the Chip Select
polarity	Possible polarity levels

Returns

None.

Remarks

This function implements an operation of the ChipXACKPolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXACKPolarity](#) in your application to automatically determine whether this feature is available.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ChipSelectXAckPolaritySelect( PMP_ID_0,
                                       chipSelect,
                                       PMP_POLARITY_ACTIVE_LOW );
```

5.6.16.6.15.2 PLIB_PMP_ChipSelectXByteEnablePortPolaritySelect Function**C**

```
void PLIB_PMP_ChipSelectXByteEnablePortPolaritySelect(
    PMP_MODULE_ID index,
    PMP_CHIP_SELECT chipSelect,
    PMP_POLARITY_LEVEL polarity
);
```

Description

This function sets the nibble/byte enable polarity to the level specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	One of the possible values from PMP_CHIP_SELECT
polarity	Possible polarity levels

Returns

None.

Remarks

This function implements an operation of the ChipXByteEnablePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXByteEnablePolarity](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipSelectXByteEnablePortPolaritySelect ( PMP_ID_0,
```



```
PMP_CHIP_SELECT_ONE,  
PMP_POLARITY_ACTIVE_LOW );
```

5.6.16.6.15.3 PLIB_PMP_ChipSelectXReadWriteStrobePolaritySelect Function

C

```
void PLIB_PMP_ChipSelectXReadWriteStrobePolaritySelect(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect,  
    PMP_POLARITY_LEVEL polarity  
);
```

Description

This function sets the read/write strobe polarity to the level specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	One of the possible values from PMP_CHIP_SELECT
polarity	Possible polarity levels

Returns

None.

Remarks

This function implements an operation of the ChipXReadWritePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXReadWritePolarity](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipSelectXReadWriteStrobePolaritySelect ( PMP_ID_0,  
                                                    PMP_CHIP_SELECT_ONE,  
                                                    PMP_POLARITY_ACTIVE_LOW );
```

5.6.16.6.15.4 PLIB_PMP_ChipSelectXWriteEnableStrobePolaritySelect Function

C

```
void PLIB_PMP_ChipSelectXWriteEnableStrobePolaritySelect(  
    PMP_MODULE_ID index,  
    PMP_CHIP_SELECT chipSelect,  
    PMP_POLARITY_LEVEL polarity  
);
```

Description

This function sets the write/enable strobe polarity to the level specified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	One of the possible values from PMP_CHIP_SELECT

polarity	Possible polarity levels

Returns

None.

Remarks

This function implements an operation of the ChipXWriteEnablePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PMP_ExistsChipXWriteEnablePolarity](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_PMP_ChipSelectXWriteEnableStrobePolaritySelect ( PMP_ID_0,
                                                    PMP_CHIP_SELECT_ONE,
                                                    PMP_POLARITY_ACTIVE_LOW );
```

5.6.16.6.16 Data Types and Constants

5.6.16.6.16.1 PMP_ACK_MODE Enumeration

C

```
typedef enum {
    PMP_USE_ACK_WITH_TIMEOUT,
    PMP_USE_ACK,
    PMP_ACK_DISABLED
} PMP_ACK_MODE;
```

Description

PMP Acknowledge Modes

This data type defines the different configurations by which the PMP can be enabled.

Members

Members	Description
PMP_USE_ACK_WITH_TIMEOUT	Acknowledge used with a timeout
PMP_USE_ACK	Acknowledge is used
PMP_ACK_DISABLED	Acknowledge is not used

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.2 PMP_ADDRESS_HOLD_LATCH_WAIT_STATES Enumeration

C

```
typedef enum {
    PMP_ADDRESS_HOLD_ONE_AND_ONE_QUARTER,
    PMP_ADDRESS_HOLD_ONE_QUARTER
} PMP_ADDRESS_HOLD_LATCH_WAIT_STATES;
```

Description

PMP Address Latch Strobe Wait States

This data type defines the different configurations by which the PMP holds after address latch strobe wait states can be configured. Refer to the specific device data sheet for the exact clock cycle timing.

Members

Members	Description
PMP_ADDRESS_HOLD_ONE_AND_ONE_QUARTER	Data Wait of 1 1/4 Peripheral Bus Clock Cycles
PMP_ADDRESS_HOLD_ONE_QUARTER	Data Wait of 1/4 Peripheral Bus Clock Cycles

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.3 PMP_ADDRESS_LATCH Enumeration**C**

```
typedef enum {
    PMP_ADDRESS_LATCH_UPPER,
    PMP_ADDRESS_LATCH_HIGH,
    PMP_ADDRESS_LATCH_LOW
} PMP_ADDRESS_LATCH;
```

Description

PMP Address Latch

This data type defines the different configurations by which the PMP address latch strobes can be configured.

Members

Members	Description
PMP_ADDRESS_LATCH_UPPER	Address latch upper
PMP_ADDRESS_LATCH_HIGH	Address latch high
PMP_ADDRESS_LATCH_LOW	Address latch low

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.4 PMP_ADDRESS_LATCH_WAIT_STATES Enumeration**C**

```
typedef enum {
    PMP_ADDRESS_WAIT_THREE_AND_HALF,
    PMP_ADDRESS_WAIT_TWO_AND_HALF,
    PMP_ADDRESS_WAIT_ONE_AND_HALF,
    PMP_ADDRESS_WAIT_HALF
} PMP_ADDRESS_LATCH_WAIT_STATES;
```

Description

PMP Address Latch Strobe Wait States

This data type defines the different configurations by which the PMP address latch strobe wait states can be configured. Refer to the specific device data sheet for the exact clock cycle timing.

Members

Members	Description
PMP_ADDRESS_WAIT_THREE_AND_HALF	Data Wait of 3 1/2 Peripheral Bus Clock Cycles
PMP_ADDRESS_WAIT_TWO_AND_HALF	Data Wait of 2 1/2 Peripheral Bus Clock Cycles
PMP_ADDRESS_WAIT_ONE_AND_HALF	Data Wait of 1 1/2 Peripheral Bus Clock Cycles
PMP_ADDRESS_WAIT_HALF	Data Wait of 1/2 Peripheral Bus Clock Cycles

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.5 PMP_ADDRESS_PORT Enumeration**C**

```
typedef enum {
    PMP_PMA0_PORT,
    PMP_PMA1_PORT,
    PMP_PMA2_TO_PMA13_PORTS,
    PMP_PMA2_PORT,
    PMP_PMA3_PORT,
    PMP_PMA4_PORT,
    PMP_PMA5_PORT,
    PMP_PMA6_PORT,
    PMP_PMA7_PORT,
    PMP_PMA8_PORT,
    PMP_PMA9_PORT,
    PMP_PMA10_PORT,
    PMP_PMA11_PORT,
    PMP_PMA12_PORT,
    PMP_PMA13_PORT,
    PMP_PMA14_PORT,
    PMP_PMA15_PORT,
    PMP_PMA16_TO_PMA22_PORTS
} PMP_ADDRESS_PORT;
```

Description

PMP Address Port Pins

This data type defines the different address that can be configured by the PMP module. The user application can make each pin as a general purpose I/O or part of PMP functionality.

Members

Members	Description
PMP_PMA0_PORT	Address line 0 port (make as general purpose I/O or PMP address line)
PMP_PMA1_PORT	Address line 1 port (make as general purpose I/O or PMP address line)
PMP_PMA2_TO_PMA13_PORTS	Address line 2 to 13 ports (make as general purpose I/O or PMP address lines)
PMP_PMA2_PORT	Address line 2 port (make as general purpose I/O or PMP address line)
PMP_PMA3_PORT	Address line 3 port (make as general purpose I/O or PMP address line)
PMP_PMA4_PORT	Address line 4 port (make as general purpose I/O or PMP address line)
PMP_PMA5_PORT	Address line 5 port (make as general purpose I/O or PMP address line)
PMP_PMA6_PORT	Address line 6 port (make as general purpose I/O or PMP address line)
PMP_PMA7_PORT	Address line 7 port (make as general purpose I/O or PMP address line)
PMP_PMA8_PORT	Address line 8 port (make as general purpose I/O or PMP address line)
PMP_PMA9_PORT	Address line 9 port (make as general purpose I/O or PMP address line)
PMP_PMA10_PORT	Address line 10 port (make as general purpose I/O or PMP address line)
PMP_PMA11_PORT	Address line 11 port (make as general purpose I/O or PMP address line)
PMP_PMA12_PORT	Address line 12 port (make as general purpose I/O or PMP address line)
PMP_PMA13_PORT	Address line 13 port (make as general purpose I/O or PMP address line)
PMP_PMA14_PORT	Address line 14 port (make as general purpose I/O or PMP address line)
PMP_PMA15_PORT	Address line 15 port (make as general purpose I/O or PMP address line)
PMP_PMA16_TO_PMA22_PORTS	Address line 16 to 22 ports (make as general purpose I/O or PMP address lines)

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h). These enumerator values can be passed in a bitwise ORed fashion to select multiple ports at a time.

5.6.16.6.16.6 PMP_ALTERNATE_MASTER_WAIT_STATES Enumeration**C**

```
typedef enum {  
    PMP_ALTERNATE_MASTER_WAIT_10,  
    PMP_ALTERNATE_MASTER_WAIT_9,  
    PMP_ALTERNATE_MASTER_WAIT_8,  
    PMP_ALTERNATE_MASTER_WAIT_7,  
    PMP_ALTERNATE_MASTER_WAIT_6,  
    PMP_ALTERNATE_MASTER_WAIT_5,  
    PMP_ALTERNATE_MASTER_WAIT_4,  
    PMP_ALTERNATE_MASTER_WAIT_3  
} PMP_ALTERNATE_MASTER_WAIT_STATES;
```

Description

PMP Alternate Master Wait States

This data type defines the different configurations by which the PMP alternate master wait states can be configured.

Members

Members	Description
PMP_ALTERNATE_MASTER_WAIT_10	Wait of 10 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_9	Wait of 9 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_8	Wait of 8 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_7	Wait of 7 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_6	Wait of 6 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_5	Wait of 5 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_4	Wait of 4 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_3	Wait of 3 Alternate Master Cycles

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.7 PMP_CHIP_SELECT Enumeration**C**

```
typedef enum {  
    PMP_CHIP_SELECT_ONE,  
    PMP_CHIP_SELECT_TWO  
} PMP_CHIP_SELECT;
```

Description

PMP Chip Select

This data type defines the different Chip Select lines of the PMP module.

Members

Members	Description
PMP_CHIP_SELECT_ONE	chip select One

PMP_CHIP_SELECT_TWO	chip select Two

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.8 PMP_CHIPSELECT_FUNCTION Enumeration**C**

```
typedef enum {
    PMCS1_AS_CHIP_SELECT,
    PMCS1_AS_ADDRESS_LINE,
    PMCS1_PMCS2_AS_ADDRESS_LINES,
    PMCS1_AS_ADDRESS_LINE_PMCS2_AS_CHIP_SELECT,
    PMCS1_AND_PMCS2_AS_CHIP_SELECT
} PMP_CHIPSELECT_FUNCTION;
```

Description

PMP Chip Select pin functions

This data type defines different functions of Chip Select pins.

Members

Members	Description
PMCS1_AS_CHIP_SELECT	CS1 used as Chip Select
PMCS1_AS_ADDRESS_LINE	CS1 used as address line
PMCS1_PMCS2_AS_ADDRESS_LINES	CS1 and CS2 used as address lines
PMCS1_AS_ADDRESS_LINE_PMCS2_AS_CHIP_SELECT	CS1 used as address line and CS2 as Chip Select
PMCS1_AND_PMCS2_AS_CHIP_SELECT	Both CS1 and CS2 used as Chip Selects

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.9 PMP_DATA_HOLD_STATES Enumeration**C**

```
typedef enum {
    PMP_DATA_HOLD_FOUR,
    PMP_DATA_HOLD_THREE,
    PMP_DATA_HOLD_TWO,
    PMP_DATA_HOLD_ONE
} PMP_DATA_HOLD_STATES;
```

Description

PMP Data Hold after strobe wait state

This data type defines the different data Hold after strobe wait states.

Members

Members	Description
PMP_DATA_HOLD_FOUR	Data Hold of 4 Peripheral Bus Clock Cycles
PMP_DATA_HOLD_THREE	Data Hold of 3 Peripheral Bus Clock Cycles
PMP_DATA_HOLD_TWO	Data Hold of 2 Peripheral Bus Clock Cycles
PMP_DATA_HOLD_ONE	Data Hold of 1 Peripheral Bus Clock Cycles

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.10 PMP_DATA_LENGTH Enumeration**C**

```
typedef enum {  
    PMP_DATA_8_BITS,  
    PMP_DATA_16_BITS,  
    PMP_DATA_32_BITS  
} PMP_DATA_LENGTH;
```

Description

PMP data length

This data type defines the possible data lengths handled by the PMP module.

Members

Members	Description
PMP_DATA_8_BITS	8-bit data length
PMP_DATA_16_BITS	16-bit data length
PMP_DATA_32_BITS	32-bit data length

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.11 PMP_DATA_SIZE Enumeration**C**

```
typedef enum {  
    PMP_DATA_SIZE_SIXTEEN_BITS,  
    PMP_DATA_SIZE_EIGHT_BITS,  
    PMP_DATA_SIZE_FOUR_BITS  
} PMP_DATA_SIZE;
```

Description

PMP DATA Size

This data type defines the different configurations for the data lengths handled by the PMP module.

Members

Members	Description
PMP_DATA_SIZE_SIXTEEN_BITS	Data length of 16-bits
PMP_DATA_SIZE_EIGHT_BITS	Data length of 8-bits
PMP_DATA_SIZE_FOUR_BITS	Data length of 4-bits

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.12 PMP_DATA_WAIT_STATES Enumeration**C**

```
typedef enum {
```

```

PMP_DATA_WAIT_FOUR,
PMP_DATA_WAIT_THREE,
PMP_DATA_WAIT_TWO,
PMP_DATA_WAIT_ONE
} PMP_DATA_WAIT_STATES;

```

Description

PMP Data setup to read/write/address phase wait state configuration

This data type defines the different wait state configurations for setup to read/write/address phase.

Members

Members	Description
PMP_DATA_WAIT_FOUR	Data Wait of 4 Peripheral Bus Clock Cycles
PMP_DATA_WAIT_THREE	Data Wait of 3 Peripheral Bus Clock Cycles
PMP_DATA_WAIT_TWO	Data Wait of 2 Peripheral Bus Clock Cycles
PMP_DATA_WAIT_ONE	Data Wait of 1 Peripheral Bus Clock Cycles

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.13 PMP_INCREMENT_MODE Enumeration**C**

```

typedef enum {
    PMP_BUFFERS_AUTO_INCREMENT,
    PMP_ADDRESS_AUTO_DECREMENT,
    PMP_ADDRESS_AUTO_INCREMENT,
    PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE
} PMP_INCREMENT_MODE;

```

Description

PMP Increment Modes

This data type defines the different configurations by which the PMP address incrementing can be configured.

Members

Members	Description
PMP_BUFFERS_AUTO_INCREMENT	Read and write buffers auto-increment
PMP_ADDRESS_AUTO_DECREMENT	Decrement PMP Address by one every read/write cycle
PMP_ADDRESS_AUTO_INCREMENT	Increment PMP Address by one every read/write cycle
PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE	PMP Address does not change

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.14 PMP_INPUT_BUFFER_TYPE Enumeration**C**

```

typedef enum {
    PMP_INPUT_BUFFER_TTL,
    PMP_INPUT_BUFFER_SCHMITT
} PMP_INPUT_BUFFER_TYPE;

```


Description

PMP Input Buffers type

This data type defines the input buffer types.

Members

Members	Description
PMP_INPUT_BUFFER_TTL	TTL input buffer
PMP_INPUT_BUFFER_SCHMITT	schmitt trigger input buffer

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.15 PMP_INTERRUPT_MODE Enumeration**C**

```
typedef enum {
    PMP_INTERRUPT_BUFFER_3_IS_ACCESSED,
    PMP_INTERRUPT_EVERY_RW_CYCLE,
    PMP_INTERRUPT_NONE
} PMP_INTERRUPT_MODE;
```

Description

PMP Interrupt Mode

This data type defines the different configurations by which the PMP can be configured for interrupt requests.

Members

Members	Description
PMP_INTERRUPT_BUFFER_3_IS_ACCESSED	Interrupt generated when Read/Write Buffer 3 is read/written
PMP_INTERRUPT_EVERY_RW_CYCLE	Interupt Occurs Every Read/Write Cycle
PMP_INTERRUPT_NONE	No interrupt generated

Remarks

None.

5.6.16.6.16.16 PMP_MASTER_MODE Enumeration**C**

```
typedef enum {
    PMP_ALTERNATE_MASTER_DIRECT,
    PMP_ALTERNATE_MASTER,
    PMP_CPU_MASTER
} PMP_MASTER_MODE;
```

Description

PMP Master Modes

This data type defines the different configurations by which the PMP can be enabled.

Members

Members	Description
PMP_ALTERNATE_MASTER_DIRECT	Alternate Master has PMP control with Direct I/O access
PMP_ALTERNATE_MASTER	Alternate Master has PMP control

PMP_CPU_MASTER	CPU has PMP control

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.17 PMP_MODULE_ID Enumeration**C**

```
typedef enum {
    PMP_ID_0
} PMP_MODULE_ID;
```

Description

PMP module ID

This data type defines the possible instances of the PMP module.

Members

Members	Description
PMP_ID_0	first instance of the PMP

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.18 PMP_MUX_MODE Enumeration**C**

```
typedef enum {
    PMP_MUX_ADDRESS_PHASES_3,
    PMP_MUX_ADDRESS_PHASES_2,
    PMP_MUX_ADDRESS_PHASES_1,
    PMP_MUX_LINES_16_ADDRESS_16_DATA,
    PMP_MUX_LINES_16_ADDRESS_8_DATA,
    PMP_MUX_LINES_8_ADDRESS_8_DATA,
    PMP_MUX_NONE
} PMP_MUX_MODE;
```

Description

PMP Multiplex Modes

This data type defines the different configurations by which the PMP can be enabled.

Members

Members	Description
PMP_MUX_ADDRESS_PHASES_3	Lower address bits are multiplexed with data bits using 3 address phase
PMP_MUX_ADDRESS_PHASES_2	Lower address bits are multiplexed with data bits using 2 address phases
PMP_MUX_ADDRESS_PHASES_1	Lower address bits are multiplexed with data bits using 1 address phase
PMP_MUX_LINES_16_ADDRESS_16_DATA	16 address lines are multiplexed on 16 data lines
PMP_MUX_LINES_16_ADDRESS_8_DATA	16 address lines multiplexed on 8 data lines
PMP_MUX_LINES_8_ADDRESS_8_DATA	8 bits of address is multiplexed on 8 data lines
PMP_MUX_NONE	No multiplexing of address and data lines

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.19 PMP_OPERATION_MODE Enumeration

C

```
typedef enum {  
    PMP_MASTER_READ_WRITE_STROBES_MULTIPLEXED,  
    PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT,  
    PMP_BUFFERED_SLAVE,  
    PMP_ENHANCED_SLAVE,  
    PMP_LEGACY_SLAVE  
} PMP_OPERATION_MODE;
```

Description

PMP Operation Modes

This data type defines the different configurations by which the PMP can be enabled.

Members

Members	Description
PMP_MASTER_READ_WRITE_STROBES_MULTIPLEXED	Master mode 1, the read and the write strobes share a single line. The enable strobe is used to decode the info sent on read/write strobe line
PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT	Master mode 2, the read and write strobes are on independent lines
PMP_BUFFERED_SLAVE	Buffered Slave mode
PMP_ENHANCED_SLAVE	Enhanced Slave mode
PMP_LEGACY_SLAVE	Legacy Parallel Slave mode

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.20 PMP_PMBE_PORT Enumeration

C

```
typedef enum {  
    PMBE_0,  
    PMBE_1  
} PMP_PMBE_PORT;
```

Description

PMP Byte Enable port.

This data type defines the available Byte Enable ports.

Members

Members	Description
PMBE_0	Byte Enable Port-0
PMBE_1	Byte Enable Port-1

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.16.21 PMP_POLARITY_LEVEL Enumeration

C

```
typedef enum {
    PMP_POLARITY_ACTIVE_HIGH,
    PMP_POLARITY_ACTIVE_LOW
} PMP_POLARITY_LEVEL;
```

Description

PMP Pins Polarity

This data type defines the possible polarity levels for the PMP pins.

Members

Members	Description
PMP_POLARITY_ACTIVE_HIGH	Active high polarity
PMP_POLARITY_ACTIVE_LOW	Active low polarity

Remarks

None.

5.6.16.6.16.22 PMP_STROBE_WAIT_STATES Enumeration

C

```
typedef enum {
    PMP_STROBE_WAIT_FIFTEEN,
    PMP_STROBE_WAIT_FOURTEEN,
    PMP_STROBE_WAIT_THIRTEEN,
    PMP_STROBE_WAIT_TWELVE,
    PMP_STROBE_WAIT_ELEVEN,
    PMP_STROBE_WAIT_TEN,
    PMP_STROBE_WAIT_NINE,
    PMP_STROBE_WAIT_EIGHT,
    PMP_STROBE_WAIT_SEVEN,
    PMP_STROBE_WAIT_SIX,
    PMP_STROBE_WAIT_FIVE,
    PMP_STROBE_WAIT_FOUR,
    PMP_STROBE_WAIT_THREE,
    PMP_STROBE_WAIT_TWO,
    PMP_STROBE_WAIT_ONE,
    PMP_STROBE_WAIT_NONE
} PMP_STROBE_WAIT_STATES;
```

Description

PMP Read to Byte enable strobe configuration

This data type defines the different configurations by which the PMP strobe wait signal time can be configured.

Members

Members	Description
PMP_STROBE_WAIT_FIFTEEN	Strobe Wait of 15 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_FOURTEEN	Strobe Wait of 14 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_THIRTEEN	Strobe Wait of 13 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_TWELVE	Strobe Wait of 12 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_ELEVEN	Strobe Wait of 11 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_TEN	Strobe Wait of 10 Peripheral Bus Clock Cycles

PMP_STROBE_WAIT_NINE	Strobe Wait of 9 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_EIGHT	Strobe Wait of 8 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_SEVEN	Strobe Wait of 7 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_SIX	Strobe Wait of 6 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_FIVE	Strobe Wait of 5 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_FOUR	Strobe Wait of 4 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_THREE	Strobe Wait of 3 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_TWO	Strobe Wait of 2 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_ONE	Strobe Wait of 1 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_NONE	No wait states

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

5.6.16.6.17 Feature Existence Functions

5.6.16.6.17.1 PLIB_PMP_ExistsAddressControl Function

C

```
bool PLIB_PMP_ExistsAddressControl(
    PMP_MODULE_ID index
);
```

Description

This interface identifies that the AddressControl feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_AddressSet](#)
- [PLIB_PMP_AddressLinesA0A1Set](#)
- [PLIB_PMP_AddressLinesA0A1Get](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AddressControl feature is supported on the device
- false - The AddressControl feature is not supported on the device

Remarks

None.

5.6.16.6.17.2 PLIB_PMP_ExistsAddressLatchPolarity Function

C

```
bool PLIB_PMP_ExistsAddressLatchPolarity(
    PMP_MODULE_ID index
);
```

```
);
```

Description

This interface identifies that the AddressLatchPolarity feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_AddressLatchPolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AddressLatchPolarity feature is supported on the device
- false - The AddressLatchPolarity feature is not supported on the device

Remarks

None.

5.6.16.6.17.3 PLIB_PMP_ExistsAddressLatchStrobePortControl Function

C

```
bool PLIB_PMP_ExistsAddressLatchStrobePortControl(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the AddressLatchStrobePortControl feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_AddressLatchStrobeEnable](#)
- [PLIB_PMP_AddressLatchStrobeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AddressLatchStrobePortControl feature is supported on the device
- false - The AddressLatchStrobePortControl feature is not supported on the device

Remarks

None.

5.6.16.6.17.4 PLIB_PMP_ExistsAddressPortPinControl Function

C

```
bool PLIB_PMP_ExistsAddressPortPinControl(  

```

```
    PMP_MODULE_ID index
);
```

Description

This interface identifies that the AddressPortPinControl feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_AddressPortEnable](#)
- [PLIB_PMP_AddressPortDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AddressPortPinControl feature is supported on the device
- false - The AddressPortPinControl feature is not supported on the device

Remarks

None.

5.6.16.6.17.5 PLIB_PMP_ExistsAltMasterRequest Function

C

```
bool PLIB_PMP_ExistsAltMasterRequest(
    PMP_MODULE_ID index
);
```

Description

This interface identifies that the AltMasterRequest feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_AlternateMasterRequestStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AltMasterRequest feature is supported on the device
- false - The AltMasterRequest feature is not supported on the device

Remarks

None.

5.6.16.6.17.6 PLIB_PMP_ExistsBufferOverflow Function

C

```
bool PLIB_PMP_ExistsBufferOverflow(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the BufferOverflow feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_InputOverflowHasOccurred](#)
- [PLIB_PMP_InputOverflowClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferOverflow feature is supported on the device
- false - The BufferOverflow feature is not supported on the device

Remarks

None.

5.6.16.6.17.7 PLIB_PMP_ExistsBufferRead Function

C

```
bool PLIB_PMP_ExistsBufferRead(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the BufferRead feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_InputBufferXByteReceive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferRead feature is supported on the device
- false - The BufferRead feature is not supported on the device

Remarks

None.

5.6.16.6.17.8 PLIB_PMP_ExistsBufferType Function

C

```
bool PLIB_PMP_ExistsBufferType(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the BufferType feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_InputBufferTypeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferType feature is supported on the device
- false - The BufferType feature is not supported on the device

Remarks

None.

5.6.16.6.17.9 PLIB_PMP_ExistsBufferUnderFlow Function

C

```
bool PLIB_PMP_ExistsBufferUnderFlow(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the BufferUnderFlow feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_OutputUnderflowHasOccurred](#)
- [PLIB_PMP_OutputUnderflowClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferUnderFlow feature is supported on the device
- false - The BufferUnderFlow feature is not supported on the device

Remarks

None.

5.6.16.6.17.10 PLIB_PMP_ExistsBufferWrite Function

C

```
bool PLIB_PMP_ExistsBufferWrite(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the BufferWrite feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_OutputBufferXByteSend](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferWrite feature is supported on the device
- false - The BufferWrite feature is not supported on the device

Remarks

None.

5.6.16.6.17.11 PLIB_PMP_ExistsBusKeeper Function

C

```
bool PLIB_PMP_ExistsBusKeeper(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the BusKeeper feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_BusKeeperEnable](#)
- [PLIB_PMP_BusKeeperDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BusKeeper feature is supported on the device
- false - The BusKeeper feature is not supported on the device

Remarks

None.

5.6.16.6.17.12 PLIB_PMP_ExistsBusyStatus Function

C

```
bool PLIB_PMP_ExistsBusyStatus(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the BusyStatus feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_PortsBusy](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BusyStatus feature is supported on the device
- false - The BusyStatus feature is not supported on the device

Remarks

None.

5.6.16.6.17.13 PLIB_PMP_ExistsByteEnablePolarity Function

C

```
bool PLIB_PMP_ExistsByteEnablePolarity(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ByteEnablePolarity feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ByteEnablePolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ByteEnablePolarity feature is supported on the device
- false - The ByteEnablePolarity feature is not supported on the device

Remarks

None.

5.6.16.6.17.14 PLIB_PMP_ExistsByteEnablePortControl Function

C

```
bool PLIB_PMP_ExistsByteEnablePortControl(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ByteEnablePortControl feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_ByteEnablePortEnable](#)
- [PLIB_PMP_ByteEnablePortDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ByteEnablePortControl feature is supported on the device
- false - The ByteEnablePortControl feature is not supported on the device

Remarks

None.

5.6.16.6.17.15 PLIB_PMP_ExistsChipSelectEnable Function

C

```
bool PLIB_PMP_ExistsChipSelectEnable(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipSelectEnable feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_ChipSelectXEnable](#)
- [PLIB_PMP_ChipSelectXDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipSelectEnable feature is supported on the device
- false - The ChipSelectEnable feature is not supported on the device

Remarks

None.

5.6.16.6.17.16 PLIB_PMP_ExistsChipSelectoperation Function**C**

```
bool PLIB_PMP_ExistsChipSelectoperation(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipSelectoperation feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipSelectFunctionSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipSelectoperation feature is supported on the device
- false - The ChipSelectoperation feature is not supported on the device

Remarks

None.

5.6.16.6.17.17 PLIB_PMP_ExistsChipXACKMode Function**C**

```
bool PLIB_PMP_ExistsChipXACKMode(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXACKMode feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipXAckModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXACKMode feature is supported on the device
- false - The ChipXACKMode feature is not supported on the device

Remarks

None.

5.6.16.6.17.18 PLIB_PMP_ExistsChipXACKPolarity Function**C**

```
bool PLIB_PMP_ExistsChipXACKPolarity(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXACKPolarity feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipSelectXAckPolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXACKPolarity feature is supported on the device
- false - The ChipXACKPolarity feature is not supported on the device

Remarks

None.

5.6.16.6.17.19 PLIB_PMP_ExistsChipXAltMasterWaitStates Function**C**

```
bool PLIB_PMP_ExistsChipXAltMasterWaitStates(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXAltMasterWaitStates feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipXWaitStatesAlternateMasterSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXAltMasterWaitStates feature is supported on the device
- false - The ChipXAltMasterWaitStates feature is not supported on the device

Remarks

None.

5.6.16.6.17.20 PLIB_PMP_ExistsChipXBaseAddress Function**C**

```
bool PLIB_PMP_ExistsChipXBaseAddress(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXBaseAddress feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipXBaseAddressSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXBaseAddress feature is supported on the device
- false - The ChipXBaseAddress feature is not supported on the device

Remarks

None.

5.6.16.6.17.21 PLIB_PMP_ExistsChipXByteEnablePolarity Function**C**

```
bool PLIB_PMP_ExistsChipXByteEnablePolarity(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXByteEnablePolarity feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipSelectXByteEnablePortPolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXByteEnablePolarity feature is supported on the device
- false - The ChipXByteEnablePolarity feature is not supported on the device

Remarks

None.

5.6.16.6.17.22 PLIB_PMP_ExistsChipXDataHoldWaitStates Function**C**

```
bool PLIB_PMP_ExistsChipXDataHoldWaitStates(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXDataHoldWaitStates feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipXWaitStatesDataHoldSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXDataHoldWaitStates feature is supported on the device
- false - The ChipXDataHoldWaitStates feature is not supported on the device

Remarks

None.

5.6.16.6.17.23 PLIB_PMP_ExistsChipXDataSetupWaitStates Function**C**

```
bool PLIB_PMP_ExistsChipXDataSetupWaitStates(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXDataSetupWaitStates feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipXWaitStatesDataSetupSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXDataSetupWaitStates feature is supported on the device
- false - The ChipXDataSetupWaitStates feature is not supported on the device

Remarks

None.

5.6.16.6.17.24 PLIB_PMP_ExistsChipXDataSize Function**C**

```
bool PLIB_PMP_ExistsChipXDataSize(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXDataSize feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipXDataSizeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXDataSize feature is supported on the device
- false - The ChipXDataSize feature is not supported on the device

Remarks

None.

5.6.16.6.17.25 PLIB_PMP_ExistsChipXEnableStrobeSelect Function**C**

```
bool PLIB_PMP_ExistsChipXEnableStrobeSelect(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXEnableStrobeSelect feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_ChipSelectXEnableStrobeSelect](#)
- [PLIB_PMP_ChipSelectXEnableStrobeDeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXEnableStrobeSelect feature is supported on the device

- false - The ChipXEnableStrobeSelect feature is not supported on the device

Remarks

None.

5.6.16.6.17.26 PLIB_PMP_ExistsChipXPolarity Function**C**

```
bool PLIB_PMP_ExistsChipXPolarity(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXPolarity feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipSelectXPolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXPolarity feature is supported on the device
- false - The ChipXPolarity feature is not supported on the device

Remarks

None.

5.6.16.6.17.27 PLIB_PMP_ExistsChipXReadWritePolarity Function**C**

```
bool PLIB_PMP_ExistsChipXReadWritePolarity(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXReadWritePolarity feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipSelectXReadWriteStrobePolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXReadWritePolarity feature is supported on the device
- false - The ChipXReadWritePolarity feature is not supported on the device

Remarks

None.

5.6.16.6.17.28 PLIB_PMP_ExistsChipXStrobeWaitStates Function**C**

```
bool PLIB_PMP_ExistsChipXStrobeWaitStates(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXStrobeWaitStates feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipXWaitStatesStrobeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXStrobeWaitStates feature is supported on the device
- false - The ChipXStrobeWaitStates feature is not supported on the device

Remarks

None.

5.6.16.6.17.29 PLIB_PMP_ExistsChipXWriteEnablePolarity Function**C**

```
bool PLIB_PMP_ExistsChipXWriteEnablePolarity(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ChipXWriteEnablePolarity feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipSelectXWriteEnableStrobePolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipXWriteEnablePolarity feature is supported on the device
- false - The ChipXWriteEnablePolarity feature is not supported on the device

Remarks

None.

5.6.16.6.17.30 PLIB_PMP_ExistsCSXActiveStatus Function**C**

```
bool PLIB_PMP_ExistsCSXActiveStatus(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the CSXActiveStatus feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ChipSelectXIsActive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CSXActiveStatus feature is supported on the device
- false - The CSXActiveStatus feature is not supported on the device

Remarks

None.

5.6.16.6.17.31 PLIB_PMP_ExistsCSXPortControl Function**C**

```
bool PLIB_PMP_ExistsCSXPortControl(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the CSXPortControl feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_ChipSelectXPortEnable](#)
- [PLIB_PMP_ChipSelectXPortDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CSXPortControl feature is supported on the device

- false - The CSXPortControl feature is not supported on the device

Remarks

None.

5.6.16.6.17.32 PLIB_PMP_ExistsCurrentMaster Function**C**

```
bool PLIB_PMP_ExistsCurrentMaster(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the CurrentMaster feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_AlternateMasterHasAccess](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CurrentMaster feature is supported on the device
- false - The CurrentMaster feature is not supported on the device

Remarks

None.

5.6.16.6.17.33 PLIB_PMP_ExistsDataHoldWaitStates Function**C**

```
bool PLIB_PMP_ExistsDataHoldWaitStates(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the DataHoldWaitStates feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_WaitStatesDataHoldSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataHoldWaitStates feature is supported on the device
- false - The DataHoldWaitStates feature is not supported on the device

Remarks

None.

5.6.16.6.17.34 PLIB_PMP_ExistsDataSetUpWaitStates Function**C**

```
bool PLIB_PMP_ExistsDataSetUpWaitStates(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the DataSetUpWaitStates feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_WaitStatesDataSetUpSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataSetUpWaitStates feature is supported on the device
- false - The DataSetUpWaitStates feature is not supported on the device

Remarks

None.

5.6.16.6.17.35 PLIB_PMP_ExistsDataStrobeWaitStates Function**C**

```
bool PLIB_PMP_ExistsDataStrobeWaitStates(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the DataStrobeWaitStates feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_WaitStatesStrobeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataStrobeWaitStates feature is supported on the device
- false - The DataStrobeWaitStates feature is not supported on the device

Remarks

None.

5.6.16.6.17.36 PLIB_PMP_ExistsDataTransferSize Function**C**

```
bool PLIB_PMP_ExistsDataTransferSize(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the DataTransferSize feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_DataSizeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataTransferSize feature is supported on the device
- false - The DataTransferSize feature is not supported on the device

Remarks

None.

5.6.16.6.17.37 PLIB_PMP_ExistsEnableControl Function**C**

```
bool PLIB_PMP_ExistsEnableControl(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the EnableControl feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_Disable](#)
- [PLIB_PMP_Enable](#)
- [PLIB_PMP_IsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Remarks

None.

5.6.16.6.17.38 PLIB_PMP_ExistsEnhancedMasterMode Function**C**

```
bool PLIB_PMP_ExistsEnhancedMasterMode(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the EnhancedMasterMode feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_EnhancedMasterModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnhancedMasterMode feature is supported on the device
- false - The EnhancedMasterMode feature is not supported on the device

Remarks

None.

5.6.16.6.17.39 PLIB_PMP_ExistsIncrementMode Function**C**

```
bool PLIB_PMP_ExistsIncrementMode(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the IncrementMode feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_AddressIncrementModeSelect](#)
- [PLIB_PMP_AddressIncrementModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The IncrementMode feature is supported on the device
- false - The IncrementMode feature is not supported on the device

Remarks

None.

5.6.16.6.17.40 PLIB_PMP_ExistsInputBufferFull Function**C**

```
bool PLIB_PMP_ExistsInputBufferFull(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the InputBufferFull feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_InputBuffersAreFull](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InputBufferFull feature is supported on the device
- false - The InputBufferFull feature is not supported on the device

Remarks

None.

5.6.16.6.17.41 PLIB_PMP_ExistsInputBufferXStatus Function**C**

```
bool PLIB_PMP_ExistsInputBufferXStatus(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the InputBufferXStatus feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_InputBufferXIsFull](#)
- [PLIB_PMP_IsDataReceived](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InputBufferXStatus feature is supported on the device
- false - The InputBufferXStatus feature is not supported on the device

Remarks

None.

5.6.16.6.17.42 PLIB_PMP_ExistsInterruptMode Function**C**

```
bool PLIB_PMP_ExistsInterruptMode(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the InterruptMode feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_InterruptModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InterruptMode feature is supported on the device
- false - The InterruptMode feature is not supported on the device

Remarks

None.

5.6.16.6.17.43 PLIB_PMP_ExistsMasterRXTX Function**C**

```
bool PLIB_PMP_ExistsMasterRXTX(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the MasterRXTX feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_MasterSend](#)

- [PLIB_PMP_MasterReceive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MasterRXTX feature is supported on the device
- false - The MasterRXTX feature is not supported on the device

Remarks

None.

5.6.16.6.17.44 PLIB_PMP_ExistsMUXModeSelect Function**C**

```
bool PLIB_PMP_ExistsMUXModeSelect(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the MUXModeSelect feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_MultiplexModeSelect](#)
- [PLIB_PMP_MultiplexModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MUXModeSelect feature is supported on the device
- false - The MUXModeSelect feature is not supported on the device

Remarks

None.

5.6.16.6.17.45 PLIB_PMP_ExistsOperationMode Function**C**

```
bool PLIB_PMP_ExistsOperationMode(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the OperationMode feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_OperationModeSelect](#)
- [PLIB_PMP_OperationModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OperationMode feature is supported on the device
- false - The OperationMode feature is not supported on the device

Remarks

None.

5.6.16.6.17.46 PLIB_PMP_ExistsOutPutBufferEmpty Function**C**

```
bool PLIB_PMP_ExistsOutPutBufferEmpty(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the OutPutBufferEmpty feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_OutputBuffersAreEmpty](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OutPutBufferEmpty feature is supported on the device
- false - The OutPutBufferEmpty feature is not supported on the device

Remarks

None.

5.6.16.6.17.47 PLIB_PMP_ExistsOutputBufferXStatus Function**C**

```
bool PLIB_PMP_ExistsOutputBufferXStatus(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the OutputBufferXStatus feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_OutputBufferXIsEmpty](#)
- [PLIB_PMP_IsDataTransmitted](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OutputBufferXStatus feature is supported on the device
- false - The OutputBufferXStatus feature is not supported on the device

Remarks

None.

5.6.16.6.17.48 PLIB_PMP_ExistsReadWritePolarity Function

C

```
bool PLIB_PMP_ExistsReadWritePolarity(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ReadWritePolarity feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_ReadWriteStrobePolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReadWritePolarity feature is supported on the device
- false - The ReadWritePolarity feature is not supported on the device

Remarks

None.

5.6.16.6.17.49 PLIB_PMP_ExistsReadWriteStrobePortControl Function

C

```
bool PLIB_PMP_ExistsReadWriteStrobePortControl(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ReadWriteStrobePortControl feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_ReadWriteStrobePortEnable](#)
- [PLIB_PMP_ReadWriteStrobePortDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReadWriteStrobePortControl feature is supported on the device
- false - The ReadWriteStrobePortControl feature is not supported on the device

Remarks

None.

5.6.16.6.17.50 PLIB_PMP_ExistsReservedAddrSpace Function**C**

```
bool PLIB_PMP_ExistsReservedAddrSpace(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the ReservedAddrSpace feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_ReservedAddressSpaceBitsSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReservedAddrSpace feature is supported on the device
- false - The ReservedAddrSpace feature is not supported on the device

Remarks

None.

5.6.16.6.17.51 PLIB_PMP_ExistsSlaveRX Function**C**

```
bool PLIB_PMP_ExistsSlaveRX(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the SlaveRX feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_SlaveReceive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SlaveRX feature is supported on the device
- false - The SlaveRX feature is not supported on the device

Remarks

None.

5.6.16.6.17.52 PLIB_PMP_ExistsSlaveTX Function**C**

```
bool PLIB_PMP_ExistsSlaveTX(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the SlaveTX feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_SlaveSend](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SlaveTX feature is supported on the device
- false - The SlaveTX feature is not supported on the device

Remarks

None.

5.6.16.6.17.53 PLIB_PMP_ExistsSmartAddress Function**C**

```
bool PLIB_PMP_ExistsSmartAddress(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the SmartAddress feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_SmartAddressStrobeEnable](#)

- [PLIB_PMP_SmartAddressStrobeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SmartAddress feature is supported on the device
- false - The SmartAddress feature is not supported on the device

Remarks

None.

5.6.16.6.17.54 PLIB_PMP_ExistsStopInIdleControl Function**C**

```
bool PLIB_PMP_ExistsStopInIdleControl(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the StopInIdleControl feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_StopInIdleEnable](#)
- [PLIB_PMP_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdleControl feature is supported on the device
- false - The StopInIdleControl feature is not supported on the device

Remarks

None.

5.6.16.6.17.55 PLIB_PMP_ExistsTransactionError Function**C**

```
bool PLIB_PMP_ExistsTransactionError(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the TransactionError feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_TransactionErrorHasOccurred](#)
- [PLIB_PMP_TransactionErrorClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransactionError feature is supported on the device
- false - The TransactionError feature is not supported on the device

Remarks

None.

5.6.16.6.17.56 PLIB_PMP_ExistsTransactionTimeOut Function**C**

```
bool PLIB_PMP_ExistsTransactionTimeOut(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the TransactionTimeOut feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_TransactionHasTimedOut](#)
- [PLIB_PMP_TransactionTimeoutClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransactionTimeOut feature is supported on the device
- false - The TransactionTimeOut feature is not supported on the device

Remarks

None.

5.6.16.6.17.57 PLIB_PMP_ExistsWaitStatesAddrHoldStrobe Function**C**

```
bool PLIB_PMP_ExistsWaitStatesAddrHoldStrobe(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the WaitStatesAddrHoldStrobe feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_WaitStatesAddressHoldStrobeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WaitStatesAddrHoldStrobe feature is supported on the device
- false - The WaitStatesAddrHoldStrobe feature is not supported on the device

Remarks

None.

5.6.16.6.17.58 PLIB_PMP_ExistsWaitStatesAddrLatchStrobe Function**C**

```
bool PLIB_PMP_ExistsWaitStatesAddrLatchStrobe(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the WaitStatesAddrLatchStrobe feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_WaitStatesAddressLatchStrobeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WaitStatesAddrLatchStrobe feature is supported on the device
- false - The WaitStatesAddrLatchStrobe feature is not supported on the device

Remarks

None.

5.6.16.6.17.59 PLIB_PMP_ExistsWriteEnablePolarity Function**C**

```
bool PLIB_PMP_ExistsWriteEnablePolarity(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the WriteEnablePolarity feature is available on the PMP module. When this interface returns true, this function is supported on the device:

- [PLIB_PMP_WriteEnableStrobePolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WriteEnablePolarity feature is supported on the device
- false - The WriteEnablePolarity feature is not supported on the device

Remarks

None.

5.6.16.6.17.60 PLIB_PMP_ExistsWriteEnablePortControl Function**C**

```
bool PLIB_PMP_ExistsWriteEnablePortControl(  
    PMP_MODULE_ID index  
);
```

Description

This interface identifies that the WriteEnablePortControl feature is available on the PMP module. When this interface returns true, these functions are supported on the device:

- [PLIB_PMP_WriteEnableStrobePortEnable](#)
- [PLIB_PMP_WriteEnableStrobePortDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WriteEnablePortControl feature is supported on the device
- false - The WriteEnablePortControl feature is not supported on the device

Remarks

None.

5.6.16.7 Files

Files

Name	Description
plib_pmp.h	Parallel Master Port (PMP) Peripheral Library Interface Header.

Description

5.6.16.7.1 [plib_pmp.h](#)

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the PMP Peripheral Library.

Functions

	Name	Description
≡	PLIB_PMP_AddressGet	Gets the current address of the PMP module.
≡	PLIB_PMP_AddressIncrementModeGet	Gets the increment mode being used with the address of the PMP module.
≡	PLIB_PMP_AddressIncrementModeSelect	Configures the increment mode being used with the address of the PMP module.
≡	PLIB_PMP_AddressLatchPolaritySelect	Sets the address latch strobe polarity.
≡	PLIB_PMP_AddressLatchStrobeDisable	Disables the specific address latch strobe.
≡	PLIB_PMP_AddressLatchStrobeEnable	Enables the specific address latch strobe.
≡	PLIB_PMP_AddressLinesA0A1Get	Gets the value of the address lines PMA0 and PMA1.
≡	PLIB_PMP_AddressLinesA0A1Set	Sets the address lines PMA0 and PMA1 with the value specified.
≡	PLIB_PMP_AddressPortDisable	Disables the port lines specified as PMP address lines.
≡	PLIB_PMP_AddressPortEnable	Enables the port lines specified as PMP address lines.
≡	PLIB_PMP_AddressSet	Sets the current address of the PMP module to the specified address.
≡	PLIB_PMP_AlternateMasterHasAccess	Gets the status of which alternate masters have gained access to the PMP module.
≡	PLIB_PMP_AlternateMasterRequestStatus	Gets the status of the request from the alternate master.
≡	PLIB_PMP_BusKeeperDisable	Disables the bus keeper (high-impedance) state.
≡	PLIB_PMP_BusKeeperEnable	Enables the bus keeper (high-impedance) state.
≡	PLIB_PMP_ByteEnablePolaritySelect	Sets the byte enable polarity.
≡	PLIB_PMP_ByteEnablePortDisable	Disables the byte enable port.
≡	PLIB_PMP_ByteEnablePortEnable	Enables the byte enable port.
≡	PLIB_PMP_ChipSelectFunctionSelect	Defines the functionality of the pins intended to be used as Chip Select.
≡	PLIB_PMP_ChipSelectXAckPolaritySelect	Sets the Chip Select acknowledge polarity.
≡	PLIB_PMP_ChipSelectXByteEnablePortPolaritySelect	Sets the nibble/byte enable polarity.
≡	PLIB_PMP_ChipSelectXDisable	Configures the Chip Select.
≡	PLIB_PMP_ChipSelectXEnable	Configures the Chip Select.
≡	PLIB_PMP_ChipSelectXEnableStrobeDeSelect	Selects the read and write strobes. The enable strobe is not selected (not required).

	PLIB_PMP_ChipSelectXEnableStrobeSelect	Selects the read/write and enable strobes.
	PLIB_PMP_ChipSelectXIsActive	Gets the current status of the specified Chip Select.
	PLIB_PMP_ChipSelectXPolaritySelect	Sets the specified Chip Select polarity.
	PLIB_PMP_ChipSelectXPortDisable	Disables the specified Chip Select's port.
	PLIB_PMP_ChipSelectXPortEnable	Enables the specified Chip Select's port.
	PLIB_PMP_ChipSelectXReadWriteStrobePolaritySelect	Sets the read/write strobe polarity.
	PLIB_PMP_ChipSelectXWriteEnableStrobePolaritySelect	Sets the write/enable strobe polarity.
	PLIB_PMP_ChipXAckModeSelect	Selects the acknowledge mode for the PMP module.
	PLIB_PMP_ChipXBaseAddressSet	Sets the base address of the specified device.
	PLIB_PMP_ChipXDataSizeSelect	Enables data transfer size for the PMP for the specified device.
	PLIB_PMP_ChipXWaitStatesAlternateMasterSelect	Selects the wait states of the alternate master for Chip Select-X.
	PLIB_PMP_ChipXWaitStatesDataHoldSelect	Configures the data hold states (after data transfer) needed to be used with the PMP module for the specified device.
	PLIB_PMP_ChipXWaitStatesDataSetupSelect	Configures the data wait states (before the data transfer) needed to be used with the PMP module for the specified device.
	PLIB_PMP_ChipXWaitStatesStrobeSelect	Configures the strobe wait states (during the data transfer) needed to be used with the PMP module for the specified device.
	PLIB_PMP_DataSizeSelect	Enables data transfer size for the PMP module.
	PLIB_PMP_Disable	Disables the specific PMP module.
	PLIB_PMP_Enable	Enables the specific PMP module.
	PLIB_PMP_EnhancedMasterModeSelect	Selects Master mode for the PMP module.
	PLIB_PMP_ExistsAddressControl	Identifies that the AddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchPolarity	Identifies that the AddressLatchPolarity feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchStrobePortControl	Identifies that the AddressLatchStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressPortPinControl	Identifies that the AddressPortPinControl feature exists on the PMP module.
	PLIB_PMP_ExistsAltMasterRequest	Identifies that the AltMasterRequest feature exists on the PMP module.
	PLIB_PMP_ExistsBufferOverFlow	Identifies that the BufferOverFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferRead	Identifies that the BufferRead feature exists on the PMP module.
	PLIB_PMP_ExistsBufferType	Identifies that the BufferType feature exists on the PMP module.
	PLIB_PMP_ExistsBufferUnderFlow	Identifies that the BufferUnderFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferWrite	Identifies that the BufferWrite feature exists on the PMP module.
	PLIB_PMP_ExistsBusKeeper	Identifies that the BusKeeper feature exists on the PMP module.
	PLIB_PMP_ExistsBusyStatus	Identifies that the BusyStatus feature exists on the PMP module.

	<u>PLIB_PMP_ExistsByteEnablePolarity</u>	Identifies that the ByteEnablePolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsByteEnablePortControl</u>	Identifies that the ByteEnablePortControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipSelectEnable</u>	Identifies that the ChipSelectEnable feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipSelectoperation</u>	Identifies that the ChipSelectoperation feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXACKMode</u>	Identifies that the ChipXACKMode feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXACKPolarity</u>	Identifies that the ChipXACKPolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXAltMasterWaitStates</u>	Identifies that the ChipXAltMasterWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXBaseAddress</u>	Identifies that the ChipXBaseAddress feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXByteEnablePolarity</u>	Identifies that the ChipXByteEnablePolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXDataHoldWaitStates</u>	Identifies that the ChipXDataHoldWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXDataSetupWaitStates</u>	Identifies that the ChipXDataSetupWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXDataSize</u>	Identifies that the ChipXDataSize feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXEnableStrobeSelect</u>	Identifies that the ChipXEnableStrobeSelect feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXPolarity</u>	Identifies that the ChipXPolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXReadWritePolarity</u>	Identifies that the ChipXReadWritePolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXStrobeWaitStates</u>	Identifies that the ChipXStrobeWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsChipXWriteEnablePolarity</u>	Identifies that the ChipXWriteEnablePolarity feature exists on the PMP module.
	<u>PLIB_PMP_ExistsCSXActiveStatus</u>	Identifies that the CSXActiveStatus feature exists on the PMP module.
	<u>PLIB_PMP_ExistsCSXPortControl</u>	Identifies that the CSXPortControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsCurrentMaster</u>	Identifies that the CurrentMaster feature exists on the PMP module.
	<u>PLIB_PMP_ExistsDataHoldWaitStates</u>	Identifies that the DataHoldWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsDataSetUpWaitStates</u>	Identifies that the DataSetUpWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsDataStrobeWaitStates</u>	Identifies that the DataStrobeWaitStates feature exists on the PMP module.
	<u>PLIB_PMP_ExistsDataTransferSize</u>	Identifies that the DataTransferSize feature exists on the PMP module.
	<u>PLIB_PMP_ExistsEnableControl</u>	Identifies that the EnableControl feature exists on the PMP module.
	<u>PLIB_PMP_ExistsEnhancedMasterMode</u>	Identifies that the EnhancedMasterMode feature exists on the PMP module.

	PLIB_PMP_ExistsIncrementMode	Identifies that the IncrementMode feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferFull	Identifies that the InputBufferFull feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferXStatus	Identifies that the InputBufferXStatus feature exists on the PMP module.
	PLIB_PMP_ExistsInterruptMode	Identifies that the InterruptMode feature exists on the PMP module.
	PLIB_PMP_ExistsMasterRXTX	Identifies that the MasterRXTX feature exists on the PMP module.
	PLIB_PMP_ExistsMUXModeSelect	Identifies that the MUXModeSelect feature exists on the PMP module.
	PLIB_PMP_ExistsOperationMode	Identifies that the OperationMode feature exists on the PMP module.
	PLIB_PMP_ExistsOutPutBufferEmpty	Identifies that the OutPutBufferEmpty feature exists on the PMP module.
	PLIB_PMP_ExistsOutputBufferXStatus	Identifies that the OutputBufferXStatus feature exists on the PMP module.
	PLIB_PMP_ExistsReadWritePolarity	Identifies that the ReadWritePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsReadWriteStrobePortControl	Identifies that the ReadWriteStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsReservedAddrSpace	Identifies that the ReservedAddrSpace feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveRX	Identifies that the SlaveRX feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveTX	Identifies that the SlaveTX feature exists on the PMP module.
	PLIB_PMP_ExistsSmartAddress	Identifies that the SmartAddress feature exists on the PMP module.
	PLIB_PMP_ExistsStopInIdleControl	Identifies that the StopInIdleControl feature exists on the PMP module.
	PLIB_PMP_ExistsTransactionError	Identifies that the TransactionError feature exists on the PMP module.
	PLIB_PMP_ExistsTransactionTimeOut	Identifies that the TransactionTimeOut feature exists on the PMP module.
	PLIB_PMP_ExistsWaitStatesAddrHoldStrobe	Identifies that the WaitStatesAddrHoldStrobe feature exists on the PMP module.
	PLIB_PMP_ExistsWaitStatesAddrLatchStrobe	Identifies that the WaitStatesAddrLatchStrobe feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePolarity	Identifies that the WriteEnablePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePortControl	Identifies that the WriteEnablePortControl feature exists on the PMP module.
	PLIB_PMP_InputBuffersAreFull	Gets the state of the input buffers.
	PLIB_PMP_InputBufferTypeSelect	Selects the input buffer based on the input passed.
	PLIB_PMP_InputBufferXByteReceive	Data to be received in Byte mode.
	PLIB_PMP_InputBufferXIsFull	Gets the state of the identified input buffer.
	PLIB_PMP_InputOverflowClear	Clears a PMP Overflow error.
	PLIB_PMP_InputOverflowHasOccurred	Identifies if there was a receiver overflow error.

	PLIB_PMP_InterruptModeGet	Gets the current configured interrupt mode being used with the PMP module.
	PLIB_PMP_InterruptModeSelect	Configures the interrupt request mode being used with the PMP module.
	PLIB_PMP_IsDataReceived	Checks and returns if the data has been received in the specified buffer in Slave mode.
	PLIB_PMP_IsDataTransmitted	Checks and returns if the data has been transmitted from the specified buffer in Slave mode.
	PLIB_PMP_IsEnabled	Checks whether or not the PMP module is enabled.
	PLIB_PMP_MasterReceive	Receives the data in the master mode.
	PLIB_PMP_MasterSend	Sends the specified data in Master mode.
	PLIB_PMP_MultiplexModeGet	Gets the current multiplexing mode configured between the address and data of the PMP module.
	PLIB_PMP_MultiplexModeSelect	Configures the multiplexing between the address and data of the PMP module.
	PLIB_PMP_OperationModeGet	Gets the current operation mode of the PMP module.
	PLIB_PMP_OperationModeSelect	Configures the operation mode of the PMP module.
	PLIB_PMP_OutputBuffersAreEmpty	Gets the state of the output buffers.
	PLIB_PMP_OutputBufferXByteSend	Data to be transmitted in Byte mode.
	PLIB_PMP_OutputBufferXIsEmpty	Gets the state of the input buffer.
	PLIB_PMP_OutputUnderflowClear	Clears a PMP output underflow error.
	PLIB_PMP_OutputUnderflowHasOccurred	Identifies if there was an output buffer sent out with no data.
	PLIB_PMP_PortIsBusy	Identifies if the (Master mode) PMP port is busy.
	PLIB_PMP_ReadWriteStrobePolaritySelect	Sets the polarity of the read strobe.
	PLIB_PMP_ReadWriteStrobePortDisable	Disables the PMP module read strobe port.
	PLIB_PMP_ReadWriteStrobePortEnable	Enables the PMP module read strobe port.
	PLIB_PMP_ReservedAddressSpaceBitsSet	Sets the address bits of the reserved address space of the EPMP. The end address of the Chip Select-2 (if present).
	PLIB_PMP_SlaveReceive	Receives the data in Slave mode.
	PLIB_PMP_SlaveSend	Sends the specified data in Slave mode.
	PLIB_PMP_SmartAddressStrobeDisable	Disables "smart" address strobes.
	PLIB_PMP_SmartAddressStrobeEnable	Enables "smart" address strobes.
	PLIB_PMP_StopInIdleDisable	Enables the PMP module to continue operation in Idle mode.
	PLIB_PMP_StopInIdleEnable	Discontinues PMP module operation when the device enters Idle mode.
	PLIB_PMP_TransactionErrorClear	Clears the transaction error flag.
	PLIB_PMP_TransactionErrorHasOccurred	Returns the status of a transaction error.
	PLIB_PMP_TransactionHasTimedOut	Returns the time-out status.
	PLIB_PMP_TransactionTimeoutClear	Clears the time-out flag.
	PLIB_PMP_WaitStatesAddressHoldStrobeSelect	Configures the address wait states needed to be used with the PMP module.
	PLIB_PMP_WaitStatesAddressLatchStrobeSelect	Configures the address wait states needed to be used with the PMP module.
	PLIB_PMP_WaitStatesDataHoldSelect	Configures the data hold states (after data transfer) needed to be used with the PMP module.

	PLIB_PMP_WaitStatesDataSetUpSelect	Configures the data wait states (before the data transfer) needed to be used with the PMP module.
	PLIB_PMP_WaitStatesStrobeSelect	Configures the strobe wait states (during the data transfer) needed to be used with the PMP module.
	PLIB_PMP_WriteEnableStrobePolaritySelect	Sets the polarity of the write enable strobe.
	PLIB_PMP_WriteEnableStrobePortDisable	Disables the PMP module write strobe port.
	PLIB_PMP_WriteEnableStrobePortEnable	Enables the PMP module write strobe port.

File Name

plib_pmp.h

Company

Microchip Technology Inc.

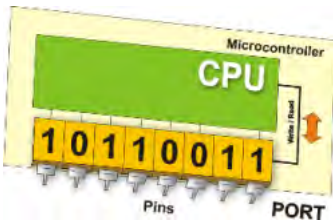
5.6.17 Ports Peripheral Library

5.6.17.1 Introduction

This library provides a low-level abstraction of the I/O Ports module on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

Ports Peripheral Library
for
Microchip Microcontrollers



I/O Ports Overview

The general purpose I/O pins can be considered the simplest of peripherals. These I/O pins allow the microcontroller to monitor and control other devices. To add flexibility and functionality to a device, some pins are multiplexed with alternate function(s). These functions depend on which peripheral features are on the device. In general, when a peripheral is functioning, that pin may not be used as a general purpose I/O pin.

A major challenge in general purpose devices is providing the largest possible set of peripheral features while minimizing the conflict of features on I/O pins. The Peripheral Pin Select (PPS) feature simplifies this challenge by enabling the user's peripheral set selection and their placement on a wide range of I/O pins. By increasing the pin out options available on a particular device, users can better tailor the microcontroller to their entire application. The PPS feature operates over a fixed subset of digital I/O

pins. Users may independently map the input and/or output of any one of many digital peripherals to any one of these I/O pins. PPS is performed in software and generally does not require the device to be reprogrammed. Hardware safeguards are included that prevent accidental or spurious changes to the peripheral mapping once it has been established.

Note: Peripheral Pin Select is not available on all devices. Please refer to the specific device data sheet to determine availability of this feature.

Following are some of the key features of the I/O Ports module:

- Individual output pin/port open-drain control
- Individual input pin/port pull-up/down control
- Monitor select inputs and generate interrupt on mismatch condition (Change Notification)
- Operation during CPU Sleep and Idle modes
- Port line analog/digital selection
- Port slew rate control
- PPS peripheral function remapping

5.6.17.2 Release Notes

MPLAB Harmony Version: v0.70b **Ports Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

None to report at this time.

5.6.17.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED AS IS MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.17.4 Using the Library

This topic describes the basic architecture of the Ports Peripheral Library and provides information and examples on how to use it.

Description

Interface Header File: [plib_ports.h](#)

The interface to the Ports Peripheral Library is defined in the "[plib_ports.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the Ports Peripheral Library must include "peripheral.h".

Library File:

The Ports Peripheral Library is part of the processor-specific peripheral library installed with the compiler. This library is automatically available (in the default search path) for any project built using the Microchip compilers.

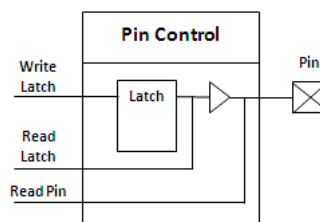
5.6.17.4.1 Hardware Abstraction Model

This topic provides detailed information for general purpose I/O and pin mapping.

Description

General Purpose I/O

All port pins have three controls directly associated with their operation as digital I/O. The data direction control determines whether the pin is an input or an output. All port pins are defined as inputs after a Reset. Writes to a port are latched and, if programmed as an output, driven by a buffer to the pin. The value of pin can be read directly from an input buffer, (after it has been synchronized to the clock), regardless of whether the pin direction has been selected as input or output. Additionally, the value being driven by the latch before the output buffer, can also be read.



Internal pull-up resistors are available on selected port pins to eliminate the need to use external pull-up resistors. These pull-ups can be controlled by this library. The open-drain feature allows the generation of outputs higher than VDD on any desired digital-only pins by using external pull-up resistors.

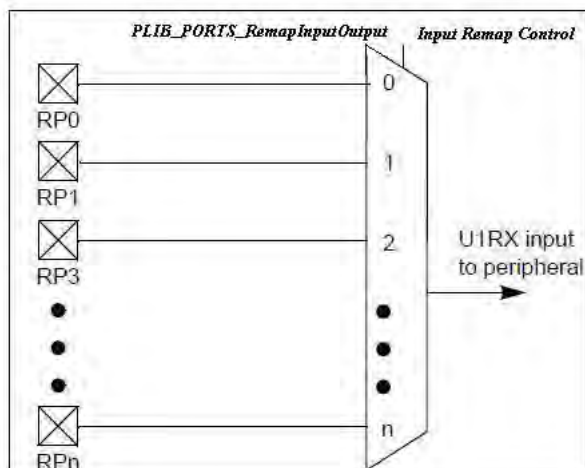
Note: For detailed specifications on the maximum allowed open-drain voltage, refer to the "**Electrical Characteristics**" chapter in the specific device data sheet.

The input change notification feature of the I/O ports allows the microcontrollers to generate interrupt requests to the processor in response to a change of state on selected input pins. This feature is capable of detecting input change of states even in Sleep mode, when the clocks are disabled.

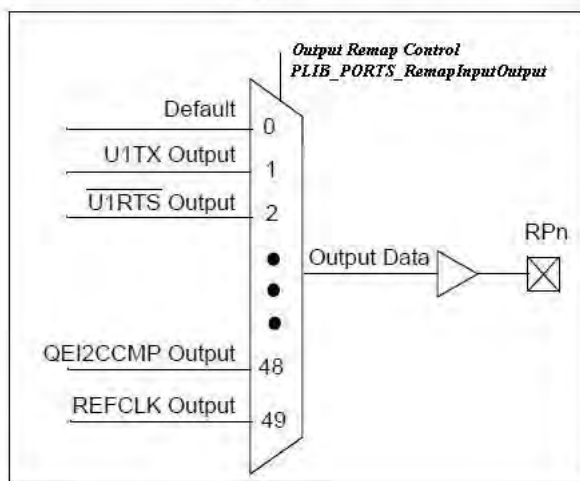
The alternate pin function selections are used to steer specific peripheral input and output functions between different pins.

The output slew rate of some port pins are programmable to select either the standard transition rate or a reduced transition rate of 'x' times the standard to minimize EMI.

Peripheral Pin Select

Remappable Input Example for USART 1 Receive**Output Mapping**

In contrast to inputs, the outputs of the peripheral pin select options are mapped on the basis of the pin. In this case, a control associated with a particular pin dictates the peripheral output to be mapped.

Multiplexing of Remappable Output for RPn**Mapping Limitations**

The control schema of the peripheral select pins is not limited to a small range of fixed peripheral configurations. There are no mutual-exclusion or hardware-enforced lockouts between any of the peripheral mapping SFRs. Literally any combination of peripheral mappings across any or all of the RPn pins is possible. This includes both many-to-one and one-to-many mappings of peripheral inputs and outputs to pins. While such mappings may be technically possible from a configuration point of view, they may not be supportable from an electrical point of view and may cause damage to the part.

Note 1: For detailed specifications on the mapping limitations, refer to the "I/O Ports" chapter in the specific device data sheet.

Note 2: Regarding pin priorities, the priority of the pins are assigned from left to right, left being the highest priority and the right most being the least priority.

5.6.17.4.2 Library Overview

The Ports Peripheral Library supports the following set of interface routines:

Library Interface Section	Description
Port Functions	Port access read/write/toggle/clear interfaces with 8-/16-/32-bit widths based on the selected device.
Port Pin Functions	Port bit/pin read/write/toggle/clear interfaces.
Change Notification Functions	Port change notification.
Peripheral Pin Select Functions	Interface routines for mapping the digital input/output to a specific the PPS remappable input/output pin.

5.6.17.4.3 How the Library Works

How the Library Works

The following topics discuss the processes involved while using a Ports module:

- [Ports Control](#)
- [Ports Function Remap](#)
- [Ports Change Notification](#)
- [Special Considerations](#)

5.6.17.4.3.1 Ports Control

Port Functions Usage:

- Port Read - Ports can be read at byte/word level using [PLIB_PORTS_Read](#), and at the bit/pin level using [PLIB_PORTS_PinGet](#)
- Port Write - Ports can be written to at byte/word level using [PLIB_PORTS_Write](#) and at the bit/pin level using [PLIB_PORTS_PinWrite](#). The function [PLIB_PORTS_PinSet](#) can be used to set a specific bit/pin of a port. A mask-based PORTx write can be accomplished using the function [PLIB_PORTS_Set](#) with the appropriate mask as a parameter along with the value.
- Port Clear - Ports can be cleared at byte/word level using [PLIB_PORTS_Clear](#) with appropriate mask as a parameter and at the bit/pin level using [PLIB_PORTS_PinClear](#)
- Port Direction - The ports read direction can be set at byte/word level using [PLIB_PORTS_DirectionInputSet](#) and at the bit/pin level using [PLIB_PORTS_PinDirectionInputSet](#). Similarly, the ports write direction can be set at the byte/word level using [PLIB_PORTS_DirectionOutputSet](#) and at the bit/pin level using [PLIB_PORTS_PinDirectionOutputSet](#). The direction information can be read using [PLIB_PORTS_DirectionGet](#).
- Port Toggle - Ports can be toggled at the byte/word level using [PLIB_PORTS_Toggle](#) with the appropriate mask as a parameter and at the bit/pin level using [PLIB_PORTS_PinToggle](#)
- Port Open Drain - Ports can be enabled for open-drain functionality at the byte/word level using the interface [PLIB_PORTS_OpenDrainEnable](#) and at the bit/pin level using [PLIB_PORTS_PinOpenDrainEnable](#). Similarly, the ports can be disabled for open-drain functionality at the byte/word level using [PLIB_PORTS_OpenDrainDisable](#) and at the bit/pin level using [PLIB_PORTS_PinOpenDrainDisable](#).
- Port Pull-up - Ports can be enabled for pull-up functionality at the byte/word level using [PLIB_PORTS_PullUpEnable](#) and at the bit/pin level using [PLIB_PORTS_PinPullUpEnable](#). Similarly, the ports can be disabled for pull-up functionality at the byte/word level using the interface [PLIB_PORTS_PullUpDisable](#) and at the bit/pin level using [PLIB_PORTS_PinPullUpDisable](#).

Example:

```

// PORT Direction setting for output
PLIB_PORTS_DirectionOutputSet(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0xFFFF);
// Writing a value into a PORT
PLIB_PORTS_Write(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_TYPE)0x1234);
// PORT Direction setting for input
PLIB_PORTS_DirectionInputSet(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0xFFFF);
// Reading back the previously written value
PORTS_DATA_TYPE readData = PLIB_PORTS_Read(MY_PORTS_INSTANCE, MY_CHANNEL);
// Clearing the PORT
PLIB_PORTS_Clear(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
// Writing the value based on the mask, only 0x34 gets written to the PORT
PLIB_PORTS_Set(MY_PORTS_INSTANCE, MY_CHANNEL, 0x1234, (PORTS_DATA_MASK)0x00FF);
// Toggling a PORT
PLIB_PORTS_Toggle(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);

```

Note: In certain devices, disabling pull-ups will enable the pull-down feature and vice-versa. For more information, refer to the "I/O Ports" chapter in the specific device data sheet or the related family reference manual section.

5.6.17.4.3.2 Ports Function Remap

Using the PPS Feature:

1. Ensure that any fixed digital peripherals on the pins to be used are disabled.
2. Ensure that pins with analog functionality are set to Digital mode.
3. Check if the Input/Output lock is active using `PLIB_PORTS_IOLockIsActive`.
4. Enable writing to the Input/Output control register using `PLIB_PORTS_IOLockDisable`.
5. Remap the input and/or output pins using the internal PPS module. This can be achieved with the function `PLIB_PORTS_RemapInputOutput` with the respective parameters defined in the processor specific headers.
6. Lock the Input/Output control register using `PLIB_PORTS_IOLockEnable`.
7. Configure and enable newly mapped PPS peripherals.

Example:

```

// Remapping input function 'Input Capture 1' to the Remappable input pin 'RPI1'
PLIB_PORTS_RemapInputOutput(MY_PORTS_INSTANCE, PORTS_REMAP_FUNCTION_IC1, PORTS_REMAP_PIN_RPI1);
// Remapping output function 'UART1 Transmit' to the Remappable output pin 'RP3'
PLIB_PORTS_RemapInputOutput(MY_PORTS_INSTANCE, PORTS_REMAP_FUNCTION_U1TX, PORTS_REMAP_PIN_RP3);

```

Note: For more information, refer to the "I/O Ports" chapter in the specific device data sheet or the related family reference manual section.

5.6.17.4.3.3 Ports Change Notification

Change Notification Feature Usage

1. Ensure that the change notify pin is configured as a digital input by setting the associated bit in the direction register using the `PLIB_PORTS_PinDirectionInputSet` or `PLIB_PORTS_DirectionInputSet` functions.
2. Select the desired change notification pin using `PLIB_PORTS_PinChangeNoticeEnable`.
3. Turn on the weak pull-up devices (if desired) for the selected change notification pins using `PLIB_PORTS_ChangeNoticePullUpEnable` (weak pull-ups can be disabled using `PLIB_PORTS_ChangeNoticePullUpDisable`).
4. Clear the selected change notification interrupt flag.
5. Select the desired interrupt priority for change notification pin.
6. Enable change notification interrupts.

7. Change notification can be disabled after the successful usage using [PLIB_PORTS_PinChangeNoticeDisable](#). Certain microcontrollers support the global control over the change notification feature using the following functions: [PLIB_PORTS_ChangeNoticeEnable](#) and [PLIB_PORTS_ChangeNoticeDisable](#). If there are any requirements to control the pull-downs, the following two functions could be used: [PLIB_PORTS_ChangeNoticePullDownEnable](#) and [PLIB_PORTS_ChangeNoticePullDownDisable](#).

Change Notification Operation in Sleep and Idle Modes

The change notification feature continues to operate during Sleep or Idle mode. Its operation can be enabled or disabled using [PLIB_PORTS_ChangeNoticeInIdleEnable](#) or [PLIB_PORTS_ChangeNoticeInIdleDisable](#). If one of the enabled change notification pins changes states, the respective status bit will be set, which can be monitored using [PLIB_PORTS_ChangeNoticeHasOccurred](#).

Example:

```
// Enabling the global change notification
PLIB_PORTS_ChangeNoticeEnable(MY_PORTS_INSTANCE);
// Enabling weak pull-ups for the change notification PIN 10
PLIB_PORTS_ChangeNoticePullUpEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10);
// Enabling change notification on PIN 10
PLIB_PORTS_GlobalChangeNoticeEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10);
```

Note: For more information, refer to the "I/O Ports" chapter in the specific device data sheet or the related family reference manual section.

5.6.17.4.3.4 Special Considerations

Considerations on Ports Usage

When reading a port register, all pins configured as analog input channels will read as cleared.

Pins configured as digital inputs will not operate correctly with analog peripherals. Analog levels on any pin that is defined as a digital input may cause the input buffer of analog peripherals to consume current that exceeds the device specifications.

Pull-ups and pull-downs on change notification pins should always be disabled when the port pin is configured as a digital output.

One instruction cycle is required between a port direction change or port write operation and a read operation of the same port. Typically, this instruction would be a NOP.

Considerations for PPS

The ability to control PPS options introduces several considerations into application design that could be overlooked. This is particularly true for several common peripherals that are available only as remappable peripherals.

The main consideration is that the peripheral pin selects are not available on default pins in the device's default (i.e., Reset) state, all PPS inputs are tied to VSS and all PPS outputs are disconnected. This condition requires the user to initialize the device with the proper peripheral configuration before any other application code is executed. For application safety, however, it is best to lock the configuration after writing to the PPS control.

The assignment of a peripheral signal to a particular pin does not automatically perform any other configuration of the pin's I/O circuitry. In theory, this means adding a pin-selectable output to a pin may mean inadvertently driving an existing peripheral input when the output is driven. Users must be familiar with the behavior of other fixed peripherals that share a remappable pin and know when to enable or disable them. To be safe, fixed digital peripherals that share the same pin should be disabled when not in use. In addition, configuring a remappable pin for a specific peripheral does not automatically turn on that feature. The peripheral must be specifically configured for operation and enabled, as if it were tied to a fixed pin. Where this occurs in the application code (immediately following a device Reset and peripheral configuration or inside the main application routine) depends on the peripheral and its use in the application.

A final consideration is that PPS functions neither override analog inputs, nor reconfigure pins with analog functions for digital I/O. If a pin is configured as an analog input on a device Reset, it must be explicitly reconfigured as a digital I/O when used with

PPS.

Note: For more information, refer to the "I/O Ports" chapter in the specific device data sheet or the related family reference manual section.

5.6.17.5 Configuring the Library

The library is configured for the supported Ports module when the supported processor is chosen in the MPLAB IDE.

5.6.17.6 Library Interface

Data Types and Constants

	Name	Description
	<u>PORTS_ANALOG_PIN</u>	Data type defining the different Analog input pins
	<u>PORTS_BIT_POS</u>	Lists the constants which holds different bit positions of PORTS.
	<u>PORTS_CHANGE_NOTICE_PIN</u>	Data type defining the different Change Notification Pins enumeration
	<u>PORTS_CHANNEL</u>	Identifies the PORT Channels Supported.
	<u>PORTS_DATA_MASK</u>	Data type defining the PORTS data mask
	<u>PORTS_DATA_TYPE</u>	Data type defining the PORTS data type.
	<u>PORTS_MODULE_ID</u>	Identifies the PORT Modules Supported.
	<u>PORTS_PERIPHERAL_OD</u>	Data type defining the different Peripherals available for Open drain Configuration
	<u>PORTS_PIN</u>	Data type defining the different PORTS IO Pins enumeration
	<u>PORTS_PIN_MODE</u>	Identifies the ports pin mode
	<u>PORTS_REMAP_FUNCTION</u>	Data type defining the different remap function enumeration
	<u>PORTS_REMAP_INPUT_FUNCTION</u>	Data type defining the different remap input function enumeration.
	<u>PORTS_REMAP_INPUT_PIN</u>	Data type defining the different Ports I/O input pins enumeration.
	<u>PORTS_REMAP_OUTPUT_FUNCTION</u>	Data type defining the different remap output function enumeration.
	<u>PORTS_REMAP_OUTPUT_PIN</u>	Data type defining the different Ports I/O output pins enumeration.
	<u>PORTS_REMAP_PIN</u>	Data type defining the different remappable input/output enumeration

Change Notification Functions

	Name	Description
	<u>PLIB_PORTS_ChangeNoticeDisable</u>	Global Change Notice disable.
	<u>PLIB_PORTS_ChangeNoticeEnable</u>	Global Change Notice enable.
	<u>PLIB_PORTS_ChangeNoticeInIdleDisable</u>	CPU Idle does not affect Change Notice operation.
	<u>PLIB_PORTS_ChangeNoticeInIdleEnable</u>	CPU Idle mode halts Change Notice operation.
	<u>PLIB_PORTS_ChangeNoticeInIdlePerPortDisable</u>	Enables the selected pin as analog or digital.
	<u>PLIB_PORTS_ChangeNoticeInIdlePerPortEnable</u>	Enables the selected pin as analog or digital.
	<u>PLIB_PORTS_ChangeNoticePerPortHasOccurred</u>	Enables the selected pin as analog or digital.
	<u>PLIB_PORTS_ChangeNoticePerPortTurnOff</u>	Enables the selected pin as analog or digital.
	<u>PLIB_PORTS_ChangeNoticePerPortTurnOn</u>	Enables the selected pin as analog or digital.
	<u>PLIB_PORTS_ChangeNoticePullDownPerPortDisable</u>	Enables the selected pin as analog or digital.
	<u>PLIB_PORTS_ChangeNoticePullDownPerPortEnable</u>	Enables the selected pin as analog or digital.
	<u>PLIB_PORTS_ChangeNoticePullUpDisable</u>	Disable pull-up on input change.
	<u>PLIB_PORTS_ChangeNoticePullUpEnable</u>	Enable pull-up on input change.

	PLIB_PORTS_ChangeNoticePullUpPerPortDisable	Enables the selected pin as analog or digital.
	PLIB_PORTS_ChangeNoticePullUpPerPortEnable	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinChangeNoticeDisable	Port pin Change Notice disable.
	PLIB_PORTS_PinChangeNoticeEnable	Port pin Change Notice enable.
	PLIB_PORTS_PinChangeNoticePerPortDisable	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinChangeNoticePerPortEnable	Enables the selected pin as analog or digital.

Feature Existence Functions

	Name	Description
	PLIB_PORTS_ExistsChangeNotice	Identifies whether the ChangeNotice feature exists on the PORTS module
	PLIB_PORTS_ExistsChangeNoticeInIdle	Identifies whether the ChangeNoticeInIdle feature exists on the PORTS module
	PLIB_PORTS_ExistsChangeNoticePerPortInIdle	Identifies whether the ChangeNoticeInIdlePerPort feature exists on the PORTS module
	PLIB_PORTS_ExistsChangeNoticePerPortStatus	Identifies whether the ChangeNoticePerPortStatus feature exists on the PORTS module
	PLIB_PORTS_ExistsChangeNoticePerPortTurnOn	Identifies whether the ChangeNoticePerPortTurnOn feature exists on the PORTS module
	PLIB_PORTS_ExistsChangeNoticePullDownPerPort	Identifies whether the ChangeNoticePullDownPerPort feature exists on the PORTS module
	PLIB_PORTS_ExistsChangeNoticePullUp	Identifies whether the ChangeNoticePullup feature exists on the PORTS module
	PLIB_PORTS_ExistsChangeNoticePullUpPerPort	Identifies whether the ChangeNoticePullUpPerPort feature exists on the PORTS module
	PLIB_PORTS_ExistsPinChangeNotice	Identifies whether the PinChangeNotice feature exists on the PORTS module
	PLIB_PORTS_ExistsPinChangeNoticePerPort	Identifies whether the PinChangeNoticePerPort feature exists on the PORTS module
	PLIB_PORTS_ExistsPinMode	Identifies whether the PinMode feature exists on the PORTS module
	PLIB_PORTS_ExistsPinModePerPort	Identifies whether the PinModePerPort feature exists on the PORTS module
	PLIB_PORTS_ExistsPortsDirection	Identifies whether the PortsDirection feature exists on the PORTS module
	PLIB_PORTS_ExistsPortsOpenDrain	Identifies whether the PortsOpenDrain feature exists on the PORTS module
	PLIB_PORTS_ExistsPortsRead	Identifies whether the PortsRead feature exists on the PORTS module
	PLIB_PORTS_ExistsPortsWrite	Identifies whether the PortsWrite feature exists on the PORTS module
	PLIB_PORTS_ExistsRemapInput	Identifies whether the RemapInput feature exists on the PORTS module
	PLIB_PORTS_ExistsRemapOutput	Identifies whether the RemapOutput feature exists on the PORTS module

Peripheral Pin Select Functions

	Name	Description
	PLIB_PORTS_RemapInput	Input/Output (I/O) function remapping.
	PLIB_PORTS_RemapOutput	Input/Output (I/O) function remapping.

Port Functions

	Name	Description
≡	PLIB_PORTS_Clear	Clears the selected digital port/latch.
≡	PLIB_PORTS_DirectionGet	Reads the direction of the selected digital port.
≡	PLIB_PORTS_DirectionInputSet	Enables the read (input) direction for the selected port.
≡	PLIB_PORTS_DirectionOutputSet	Enables the write (output) direction for the selected port.
≡	PLIB_PORTS_PinOpenDrainDisable	Disables the open drain functionality for the selected pin.
≡	PLIB_PORTS_PinOpenDrainEnable	Enables the open drain functionality for the selected pin.
≡	PLIB_PORTS_Read	Reads the selected digital port.
≡	PLIB_PORTS_Set	Writes the selected digital port/latch based on the mask.
≡	PLIB_PORTS_Toggle	Toggles the selected digital port/latch.
≡	PLIB_PORTS_Write	Writes the selected digital port/latch.
≡	PLIB_PORTS_OpenDrainDisable	Disables the open drain functionality for the selected port.
≡	PLIB_PORTS_OpenDrainEnable	Enables the open drain functionality for the selected port.

Port Pin Functions

	Name	Description
≡	PLIB_PORTS_PinClear	Clears the selected digital pin/latch.
≡	PLIB_PORTS_PinDirectionInputSet	Enables the read (input) direction for the selected pin.
≡	PLIB_PORTS_PinDirectionOutputSet	Enables the write (output) direction for the selected pin.
≡	PLIB_PORTS_PinGet	Reads/Gets data from the selected digital pin.
≡	PLIB_PORTS_PinModeSelect	Enables the selected pin as analog or digital.
≡	PLIB_PORTS_PinSet	Sets the selected digital pin/latch.
≡	PLIB_PORTS_PinToggle	Toggles the selected digital pin/latch.
≡	PLIB_PORTS_PinWrite	Writes the selected digital pin/latch.
≡	PLIB_PORTS_PinModePerPortSelect	Enables the selected pin as analog or digital.

Description

This section lists the interface routines, data types, constants, and macros required for the Ports Peripheral Library.

5.6.17.6.1 Port Pin Functions**5.6.17.6.1.1 PLIB_PORTS_PinClear Function****C**

```
void PLIB_PORTS_PinClear(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function clears the selected digital pin/latch.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsWrite](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
PLIB_PORTS_PinClear(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.6.17.6.1.2 PLIB_PORTS_PinDirectionInputSet Function**C**

```
void PLIB_PORTS_PinDirectionInputSet(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function enables the read (input) direction for the selected pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsDirection](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
PLIB_PORTS_PinDirectionInputSet(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.6.17.6.1.3 PLIB_PORTS_PinDirectionOutputSet Function

C

```
void PLIB_PORTS_PinDirectionOutputSet(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function enables the write (output) direction for the selected pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsDirection](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// MY_PINNUM - PORTS_PIN_10  
PLIB_PORTS_PinDirectionOutputSet(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.6.17.6.1.4 PLIB_PORTS_PinGet Function

C

```
bool PLIB_PORTS_PinGet(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function reads/gets data from the selected digital pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

Port pin read data.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsRead](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// MY_PINNUM - PORTS_PIN_10  
bool bitStatus = PLIB_PORTS_PinGet(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.6.17.6.1.5 PLIB_PORTS_PinModeSelect Function**C**

```
void PLIB_PORTS_PinModeSelect(  
    PORTS_MODULE_ID index,  
    PORTS_ANALOG_PIN pin,  
    PORTS_PIN_MODE mode  
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	Possible values of PORTS_ANALOG_PIN
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// MY_PIN - PORTS_ANALOG_PIN_0  
// MY_PIN_MODE - PORTS_PIN_MODE_ANALOG  
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.1.6 PLIB_PORTS_PinSet Function**C**

```
void PLIB_PORTS_PinSet(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function sets the selected digital pin/latch.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsWrite](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
PLIB_PORTS_PinSet(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.6.17.6.1.7 PLIB_PORTS_PinToggle Function**C**

```
void PLIB_PORTS_PinToggle(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function toggles the selected digital pin/latch.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsWrite](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
PLIB_PORTS_PinToggle(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.6.17.6.1.8 PLIB_PORTS_PinWrite Function**C**

```
void PLIB_PORTS_PinWrite(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos,
    bool value
);
```

Description

This function writes to the selected digital pin/latch.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS
value	Value to be written to the specific pin/latch
true	sets the bit, false - clears the bit

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsWrite](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
PLIB_PORTS_PinWrite(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.6.17.6.1.9 PLIB_PORTS_PinModePerPortSelect Function**C**

```
void PLIB_PORTS_PinModePerPortSelect(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos,
    PORTS_PIN_MODE mode
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN      - PORTS_ANALOG_PIN_0
// MY_PIN_MODE - PORTS_PIN_MODE_ANALOG
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.2 Port Functions

5.6.17.6.2.1 PLIB_PORTS_Clear Function

C

```
void PLIB_PORTS_Clear(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_DATA_MASK clearMask
);
```

Description

This function clears the selected digital port/latch.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
clearMask	Identifies the bits to be cleared

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsWrite](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PORTS_DATA_MASK clearMask = (PORTS_DATA_MASK)0x00FF;
PLIB_PORTS_Clear(MY_PORTS_INSTANCE, MY_CHANNEL, clearMask);
```

5.6.17.6.2.2 PLIB_PORTS_DirectionGet Function**C**

```
PORTS_DATA_MASK PLIB_PORTS_DirectionGet(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel
);
```

Description

This function reads the direction of the selected digital port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.

Returns

Direction of the selected port of type [PORTS_DATA_MASK](#)

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsDirection](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PORTS_DATA_MASK readDir = PLIB_PORTS_DirectionGet(MY_PORTS_INSTANCE, MY_CHANNEL);
```

5.6.17.6.2.3 PLIB_PORTS_DirectionInputSet Function**C**

```
void PLIB_PORTS_DirectionInputSet(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_DATA_MASK mask
);
```

Description

This function enables the read (input) direction for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

channel	Identifier for the PORTS channel A, B, C, etc.
mask	mask for the direction of width PORTS_DATA_MASK

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsDirection](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PLIB_PORTS_DirectionInputSet(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
```

5.6.17.6.2.4 PLIB_PORTS_DirectionOutputSet Function**C**

```
void PLIB_PORTS_DirectionOutputSet(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_DATA_MASK mask
);
```

Description

This function enables the write (output) direction for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
mask	mask for the direction of width PORTS_DATA_MASK

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsDirection](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PLIB_PORTS_DirectionOutputSet(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
```

5.6.17.6.2.5 PLIB_PORTS_PinOpenDrainDisable Function**C**

```
void PLIB_PORTS_PinOpenDrainDisable(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function disables the open drain functionality for the selected pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsOpenDrain](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
PLIB_PORTS_PinOpenDrainDisable(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.6.17.6.2.6 PLIB_PORTS_PinOpenDrainEnable Function**C**

```
void PLIB_PORTS_PinOpenDrainEnable(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function enables the open drain functionality for the selected pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsOpenDrain](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// MY_PINNUM - PORTS_PIN_10  
PLIB_PORTS_PinOpenDrainEnable(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.6.17.6.2.7 PLIB_PORTS_Read Function**C**

```
PORTS_DATA_TYPE PLIB_PORTS_Read(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel  
);
```

Description

This function reads from the selected digital port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.

Returns

data on a port with width [PORTS_DATA_TYPE](#)

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsRead](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
PORTS_DATA_TYPE readData = PLIB_PORTS_Read(MY_PORTS_INSTANCE, MY_CHANNEL);
```

5.6.17.6.2.8 PLIB_PORTS_Set Function**C**

```
void PLIB_PORTS_Set(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_DATA_TYPE value,  
    PORTS_DATA_MASK mask  
);
```

Description

This function writes to the selected digital port/latch relative to the mask.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

channel	Identifier for the PORTS channel A, B, C, etc.
value	Value to be written into a port of width PORTS_DATA_TYPE
mask	Identifies the bits to be written to

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsWrite](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_VALUE - 0x1234
PORTS_DATA_MASK myMask = (PORTS_DATA_MASK)0x00FF;
PLIB_PORTS_Set(MY_PORTS_INSTANCE, MY_CHANNEL, MY_VALUE, myMask);
```

5.6.17.6.2.9 PLIB_PORTS_Toggle Function**C**

```
void PLIB_PORTS_Toggle(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_DATA_MASK toggleMask
);
```

Description

This function toggles the selected digital port/latch.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
toggleMask	Identifies the bits to be toggled

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsWrite](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PORTS_DATA_MASK toggleMask = (PORTS_DATA_MASK)0x00FF;
PLIB_PORTS_Toggle(MY_PORTS_INSTANCE, MY_CHANNEL, toggleMask);
```

5.6.17.6.2.10 PLIB_PORTS_Write Function**C**

```
void PLIB_PORTS_Write(
```

```
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_DATA_TYPE value  
);
```

Description

This function writes to the selected digital port/latch.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
value	Value to be written into a port of width PORTS_DATA_TYPE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsWrite](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// MY_VALUE - 0x1234  
PLIB_PORTS_Write(MY_PORTS_INSTANCE, MY_CHANNEL, MY_VALUE);
```

5.6.17.6.2.11 PLIB_PORTS_OpenDrainDisable Function

C

```
void PLIB_PORTS_OpenDrainDisable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_DATA_MASK mask  
);
```

Description

This function disables the open drain functionality for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
mask	mask of type PORTS_DATA_MASK

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_PORTS_ExistsPortsOpenDrain](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PLIB_PORTS_OpenDrainDisable(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
```

5.6.17.6.2.12 PLIB_PORTS_OpenDrainEnable Function

C

```
void PLIB_PORTS_OpenDrainEnable(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_DATA_MASK mask
);
```

Description

This function enables the open drain functionality for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORTS channel A, B, C, etc.
mask	mask of type PORTS_DATA_MASK

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPortsOpenDrain](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PLIB_PORTS_OpenDrainEnable(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
```

5.6.17.6.3 Peripheral Pin Select Functions

5.6.17.6.3.1 PLIB_PORTS_RemapInput Function

C

```
void PLIB_PORTS_RemapInput(
    PORTS_MODULE_ID index,
    PORTS_REMAP_INPUT_FUNCTION function,
    PORTS_REMAP_INPUT_PIN remapPin
);
```

Description

This function controls the I/O function remapping.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_PORTS_ExistsRemapInputOutput` in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// Remapping input function 'Input Capture 1' to the Remappable pin 'RPD2'  
PLIB_PORTS_RemapInput(MY_PORTS_INSTANCE, INPUT_FUNC_IC1, INPUT_PIN_RPD2 );
```

5.6.17.6.3.2 PLIB_PORTS_RemapOutput Function**C**

```
void PLIB_PORTS_RemapOutput(  
    PORTS_MODULE_ID index,  
    PORTS_REMAP_OUTPUT_FUNCTION function,  
    PORTS_REMAP_OUTPUT_PIN remapPin  
);
```

Description

This function controls the I/O function remapping.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_PORTS_ExistsRemapInputOutput` in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// Remapping output function 'UART3 Transmit' to the Remappable pin 'RPA14'  
PLIB_PORTS_RemapInputOutput(MY_PORTS_INSTANCE, OUTPUT_FUNC_U3TX, OUTPUT_PIN_RPA14);
```

5.6.17.6.4 Change Notification Functions

5.6.17.6.4.1 PLIB_PORTS_ChangeNoticeDisable Function

C

```
void PLIB_PORTS_ChangeNoticeDisable(  
    PORTS_MODULE_ID index  
);
```

Description

This function disables the global Change Notice feature.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsChangeNotice](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
PLIB_PORTS_ChangeNoticeDisable(MY_PORTS_INSTANCE);
```

5.6.17.6.4.2 PLIB_PORTS_ChangeNoticeEnable Function

C

```
void PLIB_PORTS_ChangeNoticeEnable(  
    PORTS_MODULE_ID index  
);
```

Description

This function enables the global Change Notice feature.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsChangeNotice](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
```

```
// application developer.  
PLIB_PORTS_ChangeNoticeEnable(MY_PORTS_INSTANCE);
```

5.6.17.6.4.3 PLIB_PORTS_ChangeNoticeInIdleDisable Function

C

```
void PLIB_PORTS_ChangeNoticeInIdleDisable(  
    PORTS_MODULE_ID index  
);
```

Description

This function allows change notice operation to continue upon CPU Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsChangeNoticeInIdle](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
PLIB_PORTS_ChangeNoticeInIdleDisable(MY_PORTS_INSTANCE);
```

5.6.17.6.4.4 PLIB_PORTS_ChangeNoticeInIdleEnable Function

C

```
void PLIB_PORTS_ChangeNoticeInIdleEnable(  
    PORTS_MODULE_ID index  
);
```

Description

This function halts Change Notice operation on CPU Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsChangeNoticeInIdle](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
PLIB_PORTS_ChangeNoticeInIdleEnable(MY_PORTS_INSTANCE);
```

5.6.17.6.4.5 PLIB_PORTS_ChangeNoticeInIdlePerPortDisable Function**C**

```
void PLIB_PORTS_ChangeNoticeInIdlePerPortDisable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel  
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// MY_PIN - PORTS_ANALOG_PIN_0  
// MY_PIN_MODE - PORTS_PIN_MODE_ANALOG  
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.6 PLIB_PORTS_ChangeNoticeInIdlePerPortEnable Function**C**

```
void PLIB_PORTS_ChangeNoticeInIdlePerPortEnable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel  
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN          - PORTS_ANALOG_PIN_0
// MY_PIN_MODE     - PORTS_PIN_MODE_ANALOG
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.7 PLIB_PORTS_ChangeNoticePerPortHasOccured Function**C**

```
bool PLIB_PORTS_ChangeNoticePerPortHasOccured(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN          - PORTS_ANALOG_PIN_0
```

```
// MY_PIN_MODE - PORTS_PIN_MODE_ANALOG
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.8 PLIB_PORTS_ChangeNoticePerPortTurnOff Function

C

```
void PLIB_PORTS_ChangeNoticePerPortTurnOff(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN - PORTS_ANALOG_PIN_0
// MY_PIN_MODE - PORTS_PIN_MODE_ANALOG
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.9 PLIB_PORTS_ChangeNoticePerPortTurnOn Function

C

```
void PLIB_PORTS_ChangeNoticePerPortTurnOn(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN          - PORTS_ANALOG_PIN_0
// MY_PIN_MODE     - PORTS_PIN_MODE_ANALOG
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.10 PLIB_PORTS_ChangeNoticePullDownPerPortDisable Function**C**

```
void PLIB_PORTS_ChangeNoticePullDownPerPortDisable(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN          - PORTS_ANALOG_PIN_0
// MY_PIN_MODE     - PORTS_PIN_MODE_ANALOG
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.11 PLIB_PORTS_ChangeNoticePullDownPerPortEnable Function

C

```
void PLIB_PORTS_ChangeNoticePullDownPerPortEnable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// MY_PIN - PORTS_ANALOG_PIN_0  
// MY_PIN_MODE - PORTS_PIN_MODE_ANALOG  
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.12 PLIB_PORTS_ChangeNoticePullUpDisable Function

C

```
void PLIB_PORTS_ChangeNoticePullUpDisable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANGE_NOTICE_PIN pinNum  
);
```

Description

This function disables pull-up on input change.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsChangeNoticePullUp](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_CHANGE_NOTICE_PIN_10
PLIB_PORTS_ChangeNoticePullUpDisable(MY_PORTS_INSTANCE, MY_PINNUM);
```

5.6.17.6.4.13 PLIB_PORTS_ChangeNoticePullUpEnable Function**C**

```
void PLIB_PORTS_ChangeNoticePullUpEnable(
    PORTS_MODULE_ID index,
    PORTS_CHANGE_NOTICE_PIN pinNum
);
```

Description

This function enables pull-up on input change.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsChangeNoticePullUp](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_CHANGE_NOTICE_PIN_10
PLIB_PORTS_ChangeNoticePullUpEnable(MY_PORTS_INSTANCE, MY_PINNUM);
```

5.6.17.6.4.14 PLIB_PORTS_ChangeNoticePullUpPerPortDisable Function**C**

```
void PLIB_PORTS_ChangeNoticePullUpPerPortDisable(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN          - PORTS_ANALOG_PIN_0
// MY_PIN_MODE     - PORTS_PIN_MODE_ANALOG
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.15 PLIB_PORTS_ChangeNoticePullUpPerPortEnable Function**C**

```
void PLIB_PORTS_ChangeNoticePullUpPerPortEnable(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN - PORTS_ANALOG_PIN_0
// MY_PIN_MODE - PORTS_PIN_MODE_ANALOG
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.16 PLIB_PORTS_PinChangeNoticeDisable Function**C**

```
void PLIB_PORTS_PinChangeNoticeDisable(
    PORTS_MODULE_ID index,
    PORTS_CHANGE_NOTICE_PIN pinNum
);
```

Description

This function disables the port pin Change Notice feature.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinChangeNotice](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_CHANGE_NOTICE_PIN_10
PLIB_PORTS_PinChangeNoticeDisable(MY_PORTS_INSTANCE, MY_PINNUM);
```

5.6.17.6.4.17 PLIB_PORTS_PinChangeNoticeEnable Function**C**

```
void PLIB_PORTS_PinChangeNoticeEnable(
    PORTS_MODULE_ID index,
    PORTS_CHANGE_NOTICE_PIN pinNum
);
```

Description

This function enables the port pin Change Notice feature.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinChangeNotice](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_CHANGE_NOTICE_PIN_10
PLIB_PORTS_PinChangeNoticeEnable(MY_PORTS_INSTANCE, MY_PINNUM);
```

5.6.17.6.4.18 PLIB_PORTS_PinChangeNoticePerPortDisable Function**C**

```
void PLIB_PORTS_PinChangeNoticePerPortDisable(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN - PORTS_ANALOG_PIN_0
// MY_PIN_MODE - PORTS_PIN_MODE_ANALOG
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.4.19 PLIB_PORTS_PinChangeNoticePerPortEnable Function**C**

```
void PLIB_PORTS_PinChangeNoticePerPortEnable(
    PORTS_MODULE_ID index,
```

```
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_PORTS_ExistsPinMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// MY_PIN      - PORTS_ANALOG_PIN_0  
// MY_PIN_MODE - PORTS_PIN_MODE_ANALOG  
PLIB_PORTS_PinModeSelect(MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE);
```

5.6.17.6.5 Data Types and Constants

5.6.17.6.5.1 PORTS_ANALOG_PIN Enumeration

C

```
typedef enum {  
    PORTS_ANALOG_PIN_0,  
    PORTS_ANALOG_PIN_1,  
    PORTS_ANALOG_PIN_2,  
    PORTS_ANALOG_PIN_3,  
    PORTS_ANALOG_PIN_4,  
    PORTS_ANALOG_PIN_5,  
    PORTS_ANALOG_PIN_6,  
    PORTS_ANALOG_PIN_7,  
    PORTS_ANALOG_PIN_8,  
    PORTS_ANALOG_PIN_9,  
    PORTS_ANALOG_PIN_10,  
    PORTS_ANALOG_PIN_11,  
    PORTS_ANALOG_PIN_12,  
    PORTS_ANALOG_PIN_13,  
    PORTS_ANALOG_PIN_14,  
    PORTS_ANALOG_PIN_15,  
    PORTS_ANALOG_PIN_16,  
    PORTS_ANALOG_PIN_17,  
    PORTS_ANALOG_PIN_18,  
    PORTS_ANALOG_PIN_19,  
    PORTS_ANALOG_PIN_20,
```

```

PORTS_ANALOG_PIN_21,
PORTS_ANALOG_PIN_22,
PORTS_ANALOG_PIN_23,
PORTS_ANALOG_PIN_24,
PORTS_ANALOG_PIN_25,
PORTS_ANALOG_PIN_26,
PORTS_ANALOG_PIN_27,
PORTS_ANALOG_PIN_28,
PORTS_ANALOG_PIN_29,
PORTS_ANALOG_PIN_30,
PORTS_ANALOG_PIN_31,
PORTS_ANALOG_PINS_ALL
} PORTS_ANALOG_PIN;

```

Description

PORTS Analog input pins

This data type defines the different Analog input pins

Members

Members	Description
PORTS_ANALOG_PIN_0	Analog Input Pin 0
PORTS_ANALOG_PIN_1	Analog Input Pin 1
PORTS_ANALOG_PIN_2	Analog Input Pin 2
PORTS_ANALOG_PIN_3	Analog Input Pin 3
PORTS_ANALOG_PIN_4	Analog Input Pin 4
PORTS_ANALOG_PIN_5	Analog Input Pin 5
PORTS_ANALOG_PIN_6	Analog Input Pin 6
PORTS_ANALOG_PIN_7	Analog Input Pin 7
PORTS_ANALOG_PIN_8	Analog Input Pin 8
PORTS_ANALOG_PIN_9	Analog Input Pin 9
PORTS_ANALOG_PIN_10	Analog Input Pin 10
PORTS_ANALOG_PIN_11	Analog Input Pin 11
PORTS_ANALOG_PIN_12	Analog Input Pin 12
PORTS_ANALOG_PIN_13	Analog Input Pin 13
PORTS_ANALOG_PIN_14	Analog Input Pin 14
PORTS_ANALOG_PIN_15	Analog Input Pin 15
PORTS_ANALOG_PIN_16	Analog Input Pin 16
PORTS_ANALOG_PIN_17	Analog Input Pin 17
PORTS_ANALOG_PIN_18	Analog Input Pin 18
PORTS_ANALOG_PIN_19	Analog Input Pin 19
PORTS_ANALOG_PIN_20	Analog Input Pin 20
PORTS_ANALOG_PIN_21	Analog Input Pin 21
PORTS_ANALOG_PIN_22	Analog Input Pin 22
PORTS_ANALOG_PIN_23	Analog Input Pin 23
PORTS_ANALOG_PIN_24	Analog Input Pin 8
PORTS_ANALOG_PIN_25	Analog Input Pin 9
PORTS_ANALOG_PIN_26	Analog Input Pin 10
PORTS_ANALOG_PIN_27	Analog Input Pin 11
PORTS_ANALOG_PIN_28	Analog Input Pin 12
PORTS_ANALOG_PIN_29	Analog Input Pin 13

PORTS_ANALOG_PIN_30	Analog Input Pin 14
PORTS_ANALOG_PIN_31	Analog Input Pin 15
PORTS_ANALOG_PINS_ALL	All Analog Pins

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.17.6.5.2 PORTS_BIT_POS Enumeration**C**

```
typedef enum {
    PORTS_BIT_POS_0,
    PORTS_BIT_POS_1,
    PORTS_BIT_POS_2,
    PORTS_BIT_POS_3,
    PORTS_BIT_POS_4,
    PORTS_BIT_POS_5,
    PORTS_BIT_POS_6,
    PORTS_BIT_POS_7,
    PORTS_BIT_POS_8,
    PORTS_BIT_POS_9,
    PORTS_BIT_POS_10,
    PORTS_BIT_POS_11,
    PORTS_BIT_POS_12,
    PORTS_BIT_POS_13,
    PORTS_BIT_POS_14,
    PORTS_BIT_POS_15
} PORTS_BIT_POS;
```

Description

PORTS bit positions

Lists the constants which hold different bit positions of PORTS.

Members

Members	Description
PORTS_BIT_POS_0	PORT bit position 0
PORTS_BIT_POS_1	PORT bit position 1
PORTS_BIT_POS_2	PORT bit position 2
PORTS_BIT_POS_3	PORT bit position 3
PORTS_BIT_POS_4	PORT bit position 4
PORTS_BIT_POS_5	PORT bit position 5
PORTS_BIT_POS_6	PORT bit position 6
PORTS_BIT_POS_7	PORT bit position 7
PORTS_BIT_POS_8	PORT bit position 8
PORTS_BIT_POS_9	PORT bit position 9
PORTS_BIT_POS_10	PORT bit position 10
PORTS_BIT_POS_11	PORT bit position 11
PORTS_BIT_POS_12	PORT bit position 12
PORTS_BIT_POS_13	PORT bit position 13
PORTS_BIT_POS_14	PORT bit position 14
PORTS_BIT_POS_15	PORT bit position 15

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

5.6.17.6.5.3 PORTS_CHANGE_NOTICE_PIN Enumeration**C**

```
typedef enum {  
    PORTS_CHANGE_NOTICE_PIN_0,  
    PORTS_CHANGE_NOTICE_PIN_1,  
    PORTS_CHANGE_NOTICE_PIN_2,  
    PORTS_CHANGE_NOTICE_PIN_3,  
    PORTS_CHANGE_NOTICE_PIN_4,  
    PORTS_CHANGE_NOTICE_PIN_5,  
    PORTS_CHANGE_NOTICE_PIN_6,  
    PORTS_CHANGE_NOTICE_PIN_7,  
    PORTS_CHANGE_NOTICE_PIN_8,  
    PORTS_CHANGE_NOTICE_PIN_9,  
    PORTS_CHANGE_NOTICE_PIN_10,  
    PORTS_CHANGE_NOTICE_PIN_11,  
    PORTS_CHANGE_NOTICE_PIN_12,  
    PORTS_CHANGE_NOTICE_PIN_13,  
    PORTS_CHANGE_NOTICE_PIN_14,  
    PORTS_CHANGE_NOTICE_PIN_15,  
    PORTS_CHANGE_NOTICE_PIN_16,  
    PORTS_CHANGE_NOTICE_PIN_17,  
    PORTS_CHANGE_NOTICE_PIN_18,  
    PORTS_CHANGE_NOTICE_PIN_19,  
    PORTS_CHANGE_NOTICE_PIN_20,  
    PORTS_CHANGE_NOTICE_PIN_21,  
    PORTS_CHANGE_NOTICE_PIN_22,  
    PORTS_CHANGE_NOTICE_PIN_23,  
    PORTS_CHANGE_NOTICE_PIN_24,  
    PORTS_CHANGE_NOTICE_PIN_25,  
    PORTS_CHANGE_NOTICE_PIN_26,  
    PORTS_CHANGE_NOTICE_PIN_27,  
    PORTS_CHANGE_NOTICE_PIN_28,  
    PORTS_CHANGE_NOTICE_PIN_29,  
    PORTS_CHANGE_NOTICE_PIN_30,  
    PORTS_CHANGE_NOTICE_PIN_31,  
    PORTS_CHANGE_NOTICE_PIN_32,  
    PORTS_CHANGE_NOTICE_PIN_33,  
    PORTS_CHANGE_NOTICE_PIN_34,  
    PORTS_CHANGE_NOTICE_PIN_35,  
    PORTS_CHANGE_NOTICE_PIN_36,  
    PORTS_CHANGE_NOTICE_PIN_37,  
    PORTS_CHANGE_NOTICE_PIN_38,  
    PORTS_CHANGE_NOTICE_PIN_39,  
    PORTS_CHANGE_NOTICE_PIN_40,  
    PORTS_CHANGE_NOTICE_PIN_41,  
    PORTS_CHANGE_NOTICE_PIN_42,  
    PORTS_CHANGE_NOTICE_PIN_43,  
    PORTS_CHANGE_NOTICE_PIN_44,  
    PORTS_CHANGE_NOTICE_PIN_45,  
    PORTS_CHANGE_NOTICE_PIN_46,  
    PORTS_CHANGE_NOTICE_PIN_47,  
    PORTS_CHANGE_NOTICE_PIN_48,  
    PORTS_CHANGE_NOTICE_PIN_49,  
    PORTS_CHANGE_NOTICE_PIN_50,  
    PORTS_CHANGE_NOTICE_PIN_51,  
    PORTS_CHANGE_NOTICE_PIN_52,  
    PORTS_CHANGE_NOTICE_PIN_53,  
    PORTS_CHANGE_NOTICE_PIN_54,  
    PORTS_CHANGE_NOTICE_PIN_55,  
}
```



```

PORTS_CHANGE_NOTICE_PIN_56,
PORTS_CHANGE_NOTICE_PIN_57,
PORTS_CHANGE_NOTICE_PIN_58,
PORTS_CHANGE_NOTICE_PIN_59,
PORTS_CHANGE_NOTICE_PIN_60,
PORTS_CHANGE_NOTICE_PIN_61,
PORTS_CHANGE_NOTICE_PIN_62,
PORTS_CHANGE_NOTICE_PIN_63,
PORTS_CHANGE_NOTICE_PIN_64,
PORTS_CHANGE_NOTICE_PIN_65,
PORTS_CHANGE_NOTICE_PIN_66,
PORTS_CHANGE_NOTICE_PIN_67,
PORTS_CHANGE_NOTICE_PIN_68,
PORTS_CHANGE_NOTICE_PIN_69,
PORTS_CHANGE_NOTICE_PIN_70,
PORTS_CHANGE_NOTICE_PIN_71,
PORTS_CHANGE_NOTICE_PIN_72,
PORTS_CHANGE_NOTICE_PIN_73,
PORTS_CHANGE_NOTICE_PIN_74,
PORTS_CHANGE_NOTICE_PIN_75,
PORTS_CHANGE_NOTICE_PIN_76,
PORTS_CHANGE_NOTICE_PIN_77,
PORTS_CHANGE_NOTICE_PIN_78,
PORTS_CHANGE_NOTICE_PIN_79,
PORTS_CHANGE_NOTICE_PIN_80,
PORTS_CHANGE_NOTICE_PIN_81,
PORTS_CHANGE_NOTICE_PIN_82,
PORTS_CHANGE_NOTICE_PIN_83,
PORTS_CHANGE_NOTICE_PIN_84
} PORTS_CHANGE_NOTICE_PIN;

```

Description

Change Notification Pins enumeration

This data type defines the different Change Notification Pins enumeration

Members

Members	Description
PORTS_CHANGE_NOTICE_PIN_0	Change Notification Pin 0
PORTS_CHANGE_NOTICE_PIN_1	Change Notification Pin 1
PORTS_CHANGE_NOTICE_PIN_2	Change Notification Pin 2
PORTS_CHANGE_NOTICE_PIN_3	Change Notification Pin 3
PORTS_CHANGE_NOTICE_PIN_4	Change Notification Pin 4
PORTS_CHANGE_NOTICE_PIN_5	Change Notification Pin 5
PORTS_CHANGE_NOTICE_PIN_6	Change Notification Pin 6
PORTS_CHANGE_NOTICE_PIN_7	Change Notification Pin 7
PORTS_CHANGE_NOTICE_PIN_8	Change Notification Pin 8
PORTS_CHANGE_NOTICE_PIN_9	Change Notification Pin 9
PORTS_CHANGE_NOTICE_PIN_10	Change Notification Pin 10
PORTS_CHANGE_NOTICE_PIN_11	Change Notification Pin 11
PORTS_CHANGE_NOTICE_PIN_12	Change Notification Pin 12
PORTS_CHANGE_NOTICE_PIN_13	Change Notification Pin 13
PORTS_CHANGE_NOTICE_PIN_14	Change Notification Pin 14
PORTS_CHANGE_NOTICE_PIN_15	Change Notification Pin 15
PORTS_CHANGE_NOTICE_PIN_16	Change Notification Pin 16
PORTS_CHANGE_NOTICE_PIN_17	Change Notification Pin 17

PORTS_CHANGE_NOTICE_PIN_18	Change Notification Pin 18
PORTS_CHANGE_NOTICE_PIN_19	Change Notification Pin 19
PORTS_CHANGE_NOTICE_PIN_20	Change Notification Pin 20
PORTS_CHANGE_NOTICE_PIN_21	Change Notification Pin 21
PORTS_CHANGE_NOTICE_PIN_22	Change Notification Pin 22
PORTS_CHANGE_NOTICE_PIN_23	Change Notification Pin 23
PORTS_CHANGE_NOTICE_PIN_24	Change Notification Pin 24
PORTS_CHANGE_NOTICE_PIN_25	Change Notification Pin 25
PORTS_CHANGE_NOTICE_PIN_26	Change Notification Pin 26
PORTS_CHANGE_NOTICE_PIN_27	Change Notification Pin 27
PORTS_CHANGE_NOTICE_PIN_28	Change Notification Pin 28
PORTS_CHANGE_NOTICE_PIN_29	Change Notification Pin 29
PORTS_CHANGE_NOTICE_PIN_30	Change Notification Pin 30
PORTS_CHANGE_NOTICE_PIN_31	Change Notification Pin 31
PORTS_CHANGE_NOTICE_PIN_32	Change Notification Pin 32
PORTS_CHANGE_NOTICE_PIN_33	Change Notification Pin 33
PORTS_CHANGE_NOTICE_PIN_34	Change Notification Pin 34
PORTS_CHANGE_NOTICE_PIN_35	Change Notification Pin 35
PORTS_CHANGE_NOTICE_PIN_36	Change Notification Pin 36
PORTS_CHANGE_NOTICE_PIN_37	Change Notification Pin 37
PORTS_CHANGE_NOTICE_PIN_38	Change Notification Pin 38
PORTS_CHANGE_NOTICE_PIN_39	Change Notification Pin 39
PORTS_CHANGE_NOTICE_PIN_40	Change Notification Pin 40
PORTS_CHANGE_NOTICE_PIN_41	Change Notification Pin 41
PORTS_CHANGE_NOTICE_PIN_42	Change Notification Pin 42
PORTS_CHANGE_NOTICE_PIN_43	Change Notification Pin 43
PORTS_CHANGE_NOTICE_PIN_44	Change Notification Pin 44
PORTS_CHANGE_NOTICE_PIN_45	Change Notification Pin 45
PORTS_CHANGE_NOTICE_PIN_46	Change Notification Pin 46
PORTS_CHANGE_NOTICE_PIN_47	Change Notification Pin 47
PORTS_CHANGE_NOTICE_PIN_48	Change Notification Pin 48
PORTS_CHANGE_NOTICE_PIN_49	Change Notification Pin 49
PORTS_CHANGE_NOTICE_PIN_50	Change Notification Pin 50
PORTS_CHANGE_NOTICE_PIN_51	Change Notification Pin 51
PORTS_CHANGE_NOTICE_PIN_52	Change Notification Pin 52
PORTS_CHANGE_NOTICE_PIN_53	Change Notification Pin 53
PORTS_CHANGE_NOTICE_PIN_54	Change Notification Pin 54
PORTS_CHANGE_NOTICE_PIN_55	Change Notification Pin 55
PORTS_CHANGE_NOTICE_PIN_56	Change Notification Pin 56
PORTS_CHANGE_NOTICE_PIN_57	Change Notification Pin 57
PORTS_CHANGE_NOTICE_PIN_58	Change Notification Pin 58
PORTS_CHANGE_NOTICE_PIN_59	Change Notification Pin 59
PORTS_CHANGE_NOTICE_PIN_60	Change Notification Pin 60
PORTS_CHANGE_NOTICE_PIN_61	Change Notification Pin 61
PORTS_CHANGE_NOTICE_PIN_62	Change Notification Pin 62

PORTS_CHANGE_NOTICE_PIN_63	Change Notification Pin 63
PORTS_CHANGE_NOTICE_PIN_64	Change Notification Pin 64
PORTS_CHANGE_NOTICE_PIN_65	Change Notification Pin 65
PORTS_CHANGE_NOTICE_PIN_66	Change Notification Pin 66
PORTS_CHANGE_NOTICE_PIN_67	Change Notification Pin 67
PORTS_CHANGE_NOTICE_PIN_68	Change Notification Pin 68
PORTS_CHANGE_NOTICE_PIN_69	Change Notification Pin 69
PORTS_CHANGE_NOTICE_PIN_70	Change Notification Pin 70
PORTS_CHANGE_NOTICE_PIN_71	Change Notification Pin 71
PORTS_CHANGE_NOTICE_PIN_72	Change Notification Pin 72
PORTS_CHANGE_NOTICE_PIN_73	Change Notification Pin 73
PORTS_CHANGE_NOTICE_PIN_74	Change Notification Pin 74
PORTS_CHANGE_NOTICE_PIN_75	Change Notification Pin 75
PORTS_CHANGE_NOTICE_PIN_76	Change Notification Pin 76
PORTS_CHANGE_NOTICE_PIN_77	Change Notification Pin 77
PORTS_CHANGE_NOTICE_PIN_78	Change Notification Pin 78
PORTS_CHANGE_NOTICE_PIN_79	Change Notification Pin 79
PORTS_CHANGE_NOTICE_PIN_80	Change Notification Pin 8
PORTS_CHANGE_NOTICE_PIN_81	Change Notification Pin 81
PORTS_CHANGE_NOTICE_PIN_82	Change Notification Pin 82
PORTS_CHANGE_NOTICE_PIN_83	Change Notification Pin 83
PORTS_CHANGE_NOTICE_PIN_84	Change Notification Pin 84

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.17.6.5.4 PORTS_CHANNEL Enumeration**C**

```
typedef enum {
    PORTS_CHANNEL_A,
    PORTS_CHANNEL_B,
    PORTS_CHANNEL_C,
    PORTS_CHANNEL_D,
    PORTS_CHANNEL_E,
    PORTS_CHANNEL_F,
    PORTS_CHANNEL_G,
    PORTS_CHANNEL_H,
    PORTS_CHANNEL_J,
    PORTS_NUMBER_OF_CHANNELS
} PORTS_CHANNEL;
```

Description

PORT Channel ID

This enumeration identifies the available PORT Channels.

Members

Members	Description
PORTS_CHANNEL_A	Port A
PORTS_CHANNEL_B	Port B

PORTS_CHANNEL_C	Port C
PORTS_CHANNEL_D	Port D
PORTS_CHANNEL_E	Port E
PORTS_CHANNEL_F	Port F
PORTS_CHANNEL_G	Port G
PORTS_CHANNEL_H	Port H
PORTS_CHANNEL_J	Port J
PORTS_NUMBER_OF_CHANNELS	Max number of Ports

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

5.6.17.6.5.5 PORTS_DATA_MASK Type**C**

```
typedef uint16_t PORTS_DATA_MASK;
```

Description

PORTS data mask definition

This data type defines the PORTS data mask

Remarks

This data type changes sizes, depending on if it is being used on an 8- 16- or 32-bit part.

5.6.17.6.5.6 PORTS_DATA_TYPE Type**C**

```
typedef uint32_t PORTS_DATA_TYPE;
```

Description

PORTS data type definition

This data type defines the PORTS data type.

Remarks

This data type changes sizes, depending on if it is being used on an 8-, 16-, or 32-bit device.

5.6.17.6.5.7 PORTS_MODULE_ID Enumeration**C**

```
typedef enum {
    PORTS_ID_0,
    PORT_NUMBER_OF_MODULES
} PORTS_MODULE_ID;
```

Description

PORT Module ID

This enumeration identifies the available PORT modules.

Members

Members	Description
PORT_NUMBER_OF_MODULES	Max number of Instances

Remarks

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

5.6.17.6.5.8 PORTS_PERIPHERAL_OD Enumeration**C**

```
typedef enum {
    PORTS_PERIPHERAL_OD_MSSP1,
    PORTS_PERIPHERAL_OD_MSSP2,
    PORTS_PERIPHERAL_OD_ECCP1,
    PORTS_PERIPHERAL_OD_ECCP2,
    PORTS_PERIPHERAL_OD_ECCP3,
    PORTS_PERIPHERAL_OD_CCP10,
    PORTS_PERIPHERAL_OD_CCP9,
    PORTS_PERIPHERAL_OD_CCP8,
    PORTS_PERIPHERAL_OD_CCP7,
    PORTS_PERIPHERAL_OD_CCP6,
    PORTS_PERIPHERAL_OD_CCP5,
    PORTS_PERIPHERAL_OD_CCP4,
    PORTS_PERIPHERAL_OD_EUSART2,
    PORTS_PERIPHERAL_OD_EUSART1
} PORTS_PERIPHERAL_OD;
```

Description

Peripherals available for Open drain Configuration

This data type defines the different Peripherals available for Open drain Configuration

Members

Members	Description
PORTS_PERIPHERAL_OD_MSSP1	Peripheral Open Drain for MSSP1
PORTS_PERIPHERAL_OD_MSSP2	Peripheral Open Drain for MSSP2
PORTS_PERIPHERAL_OD_ECCP1	Peripheral Open Drain for ECCP1
PORTS_PERIPHERAL_OD_ECCP2	Peripheral Open Drain for ECCP2
PORTS_PERIPHERAL_OD_ECCP3	Peripheral Open Drain for ECCP3
PORTS_PERIPHERAL_OD_CCP10	Peripheral Open Drain for CCP10
PORTS_PERIPHERAL_OD_CCP9	Peripheral Open Drain for CCP9
PORTS_PERIPHERAL_OD_CCP8	Peripheral Open Drain for CCP8
PORTS_PERIPHERAL_OD_CCP7	Peripheral Open Drain for CCP7
PORTS_PERIPHERAL_OD_CCP6	Peripheral Open Drain for CCP6
PORTS_PERIPHERAL_OD_CCP5	Peripheral Open Drain for CCP5
PORTS_PERIPHERAL_OD_CCP4	Peripheral Open Drain for CCP4
PORTS_PERIPHERAL_OD_EUSART2	Peripheral Open Drain for EUSART2
PORTS_PERIPHERAL_OD_EUSART1	Peripheral Open Drain for EUSART1

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.17.6.5.9 PORTS_PIN Enumeration

C

```
typedef enum {
    PORTS_PIN_0,
    PORTS_PIN_1,
    PORTS_PIN_2,
    PORTS_PIN_3,
    PORTS_PIN_4,
    PORTS_PIN_5,
    PORTS_PIN_6,
    PORTS_PIN_7,
    PORTS_PIN_8,
    PORTS_PIN_9,
    PORTS_PIN_10,
    PORTS_PIN_11,
    PORTS_PIN_12,
    PORTS_PIN_13,
    PORTS_PIN_14,
    PORTS_PIN_15
} PORTS_PIN;
```

Description

PORTS IO Pins enumeration

This data type defines the different PORTS IO Pins enumeration

Members

Members	Description
PORTS_PIN_0	PORT Bit Position 0
PORTS_PIN_1	PORT Bit Position 1
PORTS_PIN_2	PORT Bit Position 2
PORTS_PIN_3	PORT Bit Position 3
PORTS_PIN_4	PORT Bit Position 4
PORTS_PIN_5	PORT Bit Position 5
PORTS_PIN_6	PORT Bit Position 6
PORTS_PIN_7	PORT Bit Position 7
PORTS_PIN_8	PORT Bit Position 8
PORTS_PIN_9	PORT Bit Position 9
PORTS_PIN_10	PORT Bit Position 10
PORTS_PIN_11	PORT Bit Position 11
PORTS_PIN_12	PORT Bit Position 12
PORTS_PIN_13	PORT Bit Position 13
PORTS_PIN_14	PORT Bit Position 14
PORTS_PIN_15	PORT Bit Position 15

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.17.6.5.10 PORTS_PIN_MODE Enumeration

C

```
typedef enum {
    PORTS_PIN_MODE_ANALOG,
```

```

    PORTS_PIN_MODE_DIGITAL
} PORTS_PIN_MODE;

```

Description

PORTs Pin Mode

This enumeration identifies the available pin modes.

Members

Members	Description
PORTS_PIN_MODE_ANALOG	Port Pin is in Analog Mode
PORTS_PIN_MODE_DIGITAL	Port pin is in Digital Mode

Remarks

Not all modules are available on all devices. Refer to the specific device data sheet to determine which modules are supported.

5.6.17.6.5.11 PORTS_REMAP_FUNCTION Enumeration

C

```

typedef enum {
    PORTS_REMAP_FUNCTION_EXT_INT1,
    PORTS_REMAP_FUNCTION_EXT_INT2,
    PORTS_REMAP_FUNCTION_EXT_INT3,
    PORTS_REMAP_FUNCTION_EXT_INT4,
    PORTS_REMAP_FUNCTION_IC1,
    PORTS_REMAP_FUNCTION_IC2,
    PORTS_REMAP_FUNCTION_IC3,
    PORTS_REMAP_FUNCTION_IC4,
    PORTS_REMAP_FUNCTION_IC5,
    PORTS_REMAP_FUNCTION_IC6,
    PORTS_REMAP_FUNCTION_IC7,
    PORTS_REMAP_FUNCTION_IC8,
    PORTS_REMAP_FUNCTION_IC9,
    PORTS_REMAP_FUNCTION_OC_FAULTA,
    PORTS_REMAP_FUNCTION_OC_FAULTB,
    PORTS_REMAP_FUNCTION_SPI1_CLOCK,
    PORTS_REMAP_FUNCTION_SPI1_DATA,
    PORTS_REMAP_FUNCTION_SPI1_SLAVE_SEL,
    PORTS_REMAP_FUNCTION_SPI2_CLOCK,
    PORTS_REMAP_FUNCTION_SPI2_DATA,
    PORTS_REMAP_FUNCTION_SPI2_SLAVE_SEL,
    PORTS_REMAP_FUNCTION_SPI3_CLOCK,
    PORTS_REMAP_FUNCTION_SPI3_DATA,
    PORTS_REMAP_FUNCTION_SPI3_SLAVE_SEL,
    PORTS_REMAP_FUNCTION_TMR2_EXT_CLOCK,
    PORTS_REMAP_FUNCTION_TMR3_EXT_CLOCK,
    PORTS_REMAP_FUNCTION_TMR4_EXT_CLOCK,
    PORTS_REMAP_FUNCTION_TMR5_EXT_CLOCK,
    PORTS_REMAP_FUNCTION_USART1_CTS,
    PORTS_REMAP_FUNCTION_USART1_RX,
    PORTS_REMAP_FUNCTION_USART2_CTS,
    PORTS_REMAP_FUNCTION_USART2_RX,
    PORTS_REMAP_FUNCTION_USART3_CTS,
    PORTS_REMAP_FUNCTION_USART3_RX,
    PORTS_REMAP_FUNCTION_USART4_CTS,
    PORTS_REMAP_FUNCTION_USART4_RX,
    PORTS_REMAP_FUNCTION_NULL,
    PORTS_REMAP_FUNCTION_C10OUT,
    PORTS_REMAP_FUNCTION_C20OUT,
    PORTS_REMAP_FUNCTION_U1TX,
    PORTS_REMAP_FUNCTION_U1RTS,
    PORTS_REMAP_FUNCTION_U2TX,

```

```

PORTS_REMAP_FUNCTION_U2RTS,
PORTS_REMAP_FUNCTION_SD01,
PORTS_REMAP_FUNCTION_SCK10UT,
PORTS_REMAP_FUNCTION_SS10UT,
PORTS_REMAP_FUNCTION_SD02,
PORTS_REMAP_FUNCTION_SCK20UT,
PORTS_REMAP_FUNCTION_SS20UT,
PORTS_REMAP_FUNCTION_OC1,
PORTS_REMAP_FUNCTION_OC2,
PORTS_REMAP_FUNCTION_OC3,
PORTS_REMAP_FUNCTION_OC4,
PORTS_REMAP_FUNCTION_OC5,
PORTS_REMAP_FUNCTION_OC6,
PORTS_REMAP_FUNCTION_OC7,
PORTS_REMAP_FUNCTION_OC8,
PORTS_REMAP_FUNCTION_U3TX,
PORTS_REMAP_FUNCTION_U3RTS,
PORTS_REMAP_FUNCTION_U4TX,
PORTS_REMAP_FUNCTION_U4RTS,
PORTS_REMAP_FUNCTION_SD03,
PORTS_REMAP_FUNCTION_SCK30UT,
PORTS_REMAP_FUNCTION_SS30UT,
PORTS_REMAP_FUNCTION_OC9,
PORTS_REMAP_FUNCTION_C30OUT
} PORTS_REMAP_FUNCTION;

```

Description

Remap Function Enumeration

This data type defines the different remap function enumeration

Members

Members	Description
PORTS_REMAP_FUNCTION_EXT_INT1	Input Function Name - External Interrupt 1
PORTS_REMAP_FUNCTION_EXT_INT2	Input Function Name - External Interrupt 2
PORTS_REMAP_FUNCTION_EXT_INT3	Input Function Name - External Interrupt 3
PORTS_REMAP_FUNCTION_EXT_INT4	Input Function Name - External Interrupt 4
PORTS_REMAP_FUNCTION_IC1	Input Function Name - Input Capture 1
PORTS_REMAP_FUNCTION_IC2	Input Function Name - Input Capture 2
PORTS_REMAP_FUNCTION_IC3	Input Function Name - Input Capture 3
PORTS_REMAP_FUNCTION_IC4	Input Function Name - Input Capture 4
PORTS_REMAP_FUNCTION_IC5	Input Function Name - Input Capture 5
PORTS_REMAP_FUNCTION_IC6	Input Function Name - Input Capture 6
PORTS_REMAP_FUNCTION_IC7	Input Function Name - Input Capture 7
PORTS_REMAP_FUNCTION_IC8	Input Function Name - Input Capture 8
PORTS_REMAP_FUNCTION_IC9	Input Function Name - Input Capture 9
PORTS_REMAP_FUNCTION_OC_FAULTA	Input Function Name - Output Compare Fault A
PORTS_REMAP_FUNCTION_OC_FAULTB	Input Function Name - Output Compare Fault B
PORTS_REMAP_FUNCTION_SPI1_CLOCK	Input Function Name - SPI 1 Clock
PORTS_REMAP_FUNCTION_SPI1_DATA	Input Function Name - SPI1 Data
PORTS_REMAP_FUNCTION_SPI1_SLAVE_SEL	Input Function Name - SPI1 Slave Select
PORTS_REMAP_FUNCTION_SPI2_CLOCK	Input Function Name - SPI2 Clock
PORTS_REMAP_FUNCTION_SPI2_DATA	Input Function Name - SPI2 Data
PORTS_REMAP_FUNCTION_SPI2_SLAVE_SEL	Input Function Name - SPI2 Slave Select
PORTS_REMAP_FUNCTION_SPI3_CLOCK	Input Function Name - SPI3 Clock

PORTS_REMAP_FUNCTION_SPI3_DATA	Input Function Name - SPI3 Data
PORTS_REMAP_FUNCTION_SPI3_SLAVE_SEL	Input Function Name - SPI3 Slave Select
PORTS_REMAP_FUNCTION_TMR2_EXT_CLOCK	Input Function Name - Timer 2 External Clock
PORTS_REMAP_FUNCTION_TMR3_EXT_CLOCK	Input Function Name - Timer 3 External Clock
PORTS_REMAP_FUNCTION_TMR4_EXT_CLOCK	Input Function Name - Timer 4 External Clock
PORTS_REMAP_FUNCTION_TMR5_EXT_CLOCK	Input Function Name - Timer5 External Clock
PORTS_REMAP_FUNCTION_USART1_CTS	Input Function Name - USART 1 Clear To Send
PORTS_REMAP_FUNCTION_USART1_RX	Input Function Name - USART 1 Receive
PORTS_REMAP_FUNCTION_USART2_CTS	Input Function Name - USART 2 Clear To Send
PORTS_REMAP_FUNCTION_USART2_RX	Input Function Name - USART 2 Receive
PORTS_REMAP_FUNCTION_USART3_CTS	Input Function Name - USART 3 Clear To Send
PORTS_REMAP_FUNCTION_USART3_RX	Input Function Name - USART 3 Receive
PORTS_REMAP_FUNCTION_USART4_CTS	Input Function Name - USART 4 Clear To Send
PORTS_REMAP_FUNCTION_USART4_RX	Input Function Name - USART 4 Receive
PORTS_REMAP_FUNCTION_NULL	Output Function Name - Null
PORTS_REMAP_FUNCTION_C1OUT	Output Function Name - Comparator 1 Output
PORTS_REMAP_FUNCTION_C2OUT	Output Function Name - Comparator 2 Output
PORTS_REMAP_FUNCTION_U1TX	Output Function Name - UART1 Transmit
PORTS_REMAP_FUNCTION_U1RTS	Output Function Name - UART1 Request To Send
PORTS_REMAP_FUNCTION_U2TX	Output Function Name - UART2 Transmit
PORTS_REMAP_FUNCTION_U2RTS	Output Function Name - UART2 Request To Send
PORTS_REMAP_FUNCTION_SDO1	Output Function Name - SPI1 Data Output
PORTS_REMAP_FUNCTION_SCK1OUT	Output Function Name - SPI1 Clock Output
PORTS_REMAP_FUNCTION_SS1OUT	Output Function Name - SPI1 Slave Select Output
PORTS_REMAP_FUNCTION_SDO2	Output Function Name - SPI2 Data Output
PORTS_REMAP_FUNCTION_SCK2OUT	Output Function Name - SPI2 Clock Output
PORTS_REMAP_FUNCTION_SS2OUT	Output Function Name - SPI2 Slave Select Output
PORTS_REMAP_FUNCTION_OC1	Output Function Name - Output Compare 1
PORTS_REMAP_FUNCTION_OC2	Output Function Name - Output Compare 2
PORTS_REMAP_FUNCTION_OC3	Output Function Name - Output Compare 3
PORTS_REMAP_FUNCTION_OC4	Output Function Name - Output Compare 4
PORTS_REMAP_FUNCTION_OC5	Output Function Name - Output Compare 5
PORTS_REMAP_FUNCTION_OC6	Output Function Name - Output Compare 6
PORTS_REMAP_FUNCTION_OC7	Output Function Name - Output Compare 7
PORTS_REMAP_FUNCTION_OC8	Output Function Name - Output Compare 8
PORTS_REMAP_FUNCTION_U3TX	Output Function Name - UART3 Transmit
PORTS_REMAP_FUNCTION_U3RTS	Output Function Name - UART3 Request To Send
PORTS_REMAP_FUNCTION_U4TX	Output Function Name - UART4 Transmit
PORTS_REMAP_FUNCTION_U4RTS	Output Function Name - UART4 Request To Send
PORTS_REMAP_FUNCTION_SDO3	Output Function Name - SPI3 Data Output
PORTS_REMAP_FUNCTION_SCK3OUT	Output Function Name - SPI3 Clock Output
PORTS_REMAP_FUNCTION_SS3OUT	Output Function Name - SPI3 Slave Select Output
PORTS_REMAP_FUNCTION_OC9	Output Function Name - Output Compare 9
PORTS_REMAP_FUNCTION_C3OUT	Output Function Name - Comparator 3 Output

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.17.6.5.12 PORTS_REMAP_INPUT_FUNCTION Enumeration**C**

```
typedef enum {  
    INPUT_FUNC_INT1,  
    INPUT_FUNC_INT2,  
    INPUT_FUNC_INT3,  
    INPUT_FUNC_INT4,  
    INPUT_FUNC_T2CK,  
    INPUT_FUNC_T3CK,  
    INPUT_FUNC_T4CK,  
    INPUT_FUNC_T5CK,  
    INPUT_FUNC_T6CK,  
    INPUT_FUNC_T7CK,  
    INPUT_FUNC_T8CK,  
    INPUT_FUNC_T9CK,  
    INPUT_FUNC_IC1,  
    INPUT_FUNC_IC2,  
    INPUT_FUNC_IC3,  
    INPUT_FUNC_IC4,  
    INPUT_FUNC_IC5,  
    INPUT_FUNC_IC6,  
    INPUT_FUNC_IC7,  
    INPUT_FUNC_IC8,  
    INPUT_FUNC_IC9,  
    INPUT_FUNC_OCFA,  
    INPUT_FUNC_OCFB,  
    INPUT_FUNC_U1RX,  
    INPUT_FUNC_U1CTS,  
    INPUT_FUNC_U2RX,  
    INPUT_FUNC_U2CTS,  
    INPUT_FUNC_U3RX,  
    INPUT_FUNC_U3CTS,  
    INPUT_FUNC_U4RX,  
    INPUT_FUNC_U4CTS,  
    INPUT_FUNC_U5RX,  
    INPUT_FUNC_U5CTS,  
    INPUT_FUNC_U6RX,  
    INPUT_FUNC_U6CTS,  
    INPUT_FUNC_SDI1,  
    INPUT_FUNC_SS1,  
    INPUT_FUNC_SDI2,  
    INPUT_FUNC_SS2,  
    INPUT_FUNC_SDI3,  
    INPUT_FUNC_SS3,  
    INPUT_FUNC_SDI4,  
    INPUT_FUNC_SS4,  
    INPUT_FUNC_SDI5,  
    INPUT_FUNC_SS5,  
    INPUT_FUNC_SDI6,  
    INPUT_FUNC_SS6,  
    INPUT_FUNC_C1RX,  
    INPUT_FUNC_C2RX,  
    INPUT_FUNC_REFCLKI1,  
    INPUT_FUNC_REFCLKI3,  
    INPUT_FUNC_REFCLKI4,  
    REMAP_NOT_SUPPORTED  
} PORTS_REMAP_INPUT_FUNCTION;
```

Description

Remap Input Function Enumeration

This data type defines the different remap input function enumeration.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

5.6.17.6.5.13 PORTS_REMAP_INPUT_PIN Enumeration

C

```
typedef enum {  
    INPUT_PIN_RPD2,  
    INPUT_PIN_RPG8,  
    INPUT_PIN_RPF4,  
    INPUT_PIN_RPD10,  
    INPUT_PIN_RPF1,  
    INPUT_PIN_RPB9,  
    INPUT_PIN_RPB10,  
    INPUT_PIN_RPC14,  
    INPUT_PIN_RPB5,  
    INPUT_PIN_RPC1,  
    INPUT_PIN_RPD14,  
    INPUT_PIN_RPG1,  
    INPUT_PIN_RPA14,  
    INPUT_PIN_RPD6,  
    INPUT_PIN_RPD3,  
    INPUT_PIN_RPG7,  
    INPUT_PIN_RPF5,  
    INPUT_PIN_RPD11,  
    INPUT_PIN_RPF0,  
    INPUT_PIN_RPB1,  
    INPUT_PIN_RPE5,  
    INPUT_PIN_RPC13,  
    INPUT_PIN_RPB3,  
    INPUT_PIN_RPC4,  
    INPUT_PIN_RPD15,  
    INPUT_PIN_RPG0,  
    INPUT_PIN_RPA15,  
    INPUT_PIN_RPD7,  
    INPUT_PIN_RPD9,  
    INPUT_PIN_RPG6,  
    INPUT_PIN_RPB8,  
    INPUT_PIN_RPB15,  
    INPUT_PIN_RPD4,  
    INPUT_PIN_RPB0,  
    INPUT_PIN_RPE3,  
    INPUT_PIN_RPB7,  
    INPUT_PIN_RPF12,  
    INPUT_PIN_RPD12,  
    INPUT_PIN_RPF8,  
    INPUT_PIN_RPC3,  
    INPUT_PIN_RPE9,  
    INPUT_PIN_RPD1,  
    INPUT_PIN_RPG9,  
    INPUT_PIN_RPB14,  
    INPUT_PIN_RPD0,  
    INPUT_PIN_RPB6,  
    INPUT_PIN_RPD5,  
    INPUT_PIN_RPB2,  
    INPUT_PIN_RPF3,  
    INPUT_PIN_RPF13,  
    INPUT_PIN_NC,  
}
```

```

INPUT_PIN_RPF2,
INPUT_PIN_RPC2,
INPUT_PIN_RPE8,
INPUT_PIN_RPF7,
INPUT_PIN_RPD8,
INPUT_PIN_RPA0,
INPUT_PIN_RPB4,
INPUT_PIN_RPC7,
INPUT_PIN_RPC0,
INPUT_PIN_RPC5,
INPUT_PIN_RPA1,
INPUT_PIN_RPB11,
INPUT_PIN_RPA8,
INPUT_PIN_RPC8,
INPUT_PIN_RPA9,
INPUT_PIN_RPA2,
INPUT_PIN_RPA4,
INPUT_PIN_RPB13,
INPUT_PIN_RPC6,
INPUT_PIN_RPA3,
INPUT_PIN_RPC9,
PIN_REMAP_NOT_SUPPORTED
} PORTS_REMAP_INPUT_PIN;

```

Description

PORTS IO Input Pins enumeration

This data type defines the different Ports I/O input pins enumeration.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

5.6.17.6.5.14 PORTS_REMAP_OUTPUT_FUNCTION Enumeration

C

```

typedef enum {
    OUTPUT_FUNC_U3TX,
    OUTPUT_FUNC_U4RTS,
    OUTPUT_FUNC_SD01,
    OUTPUT_FUNC_SD02,
    OUTPUT_FUNC_SD03,
    OUTPUT_FUNC_SD05,
    OUTPUT_FUNC_SS6,
    OUTPUT_FUNC_OC3,
    OUTPUT_FUNC_OC6,
    OUTPUT_FUNC_REFCLK04,
    OUTPUT_FUNC_C2OUT,
    OUTPUT_FUNC_C1TX,
    OUTPUT_FUNC_U1TX,
    OUTPUT_FUNC_U2RTS,
    OUTPUT_FUNC_U5TX,
    OUTPUT_FUNC_U6RTS,
    OUTPUT_FUNC_SD04,
    OUTPUT_FUNC_OC4,
    OUTPUT_FUNC_OC7,
    OUTPUT_FUNC_REFCLK01,
    OUTPUT_FUNC_U3RTS,
    OUTPUT_FUNC_U4TX,
    OUTPUT_FUNC_U6TX,
    OUTPUT_FUNC_SS1,
    OUTPUT_FUNC_SS3,
    OUTPUT_FUNC_SS4,
    OUTPUT_FUNC_SS5,
    OUTPUT_FUNC_SD06,

```

```

    OUTPUT_FUNC_OC5,
    OUTPUT_FUNC_OC8,
    OUTPUT_FUNC_C10OUT,
    OUTPUT_FUNC_REFCLK03,
    OUTPUT_FUNC_U1RTS,
    OUTPUT_FUNC_U2TX,
    OUTPUT_FUNC_U5RTS,
    OUTPUT_FUNC_SS2,
    OUTPUT_FUNC_OC2,
    OUTPUT_FUNC_OC1,
    OUTPUT_FUNC_OC9,
    OUTPUT_FUNC_C2TX,
    OUTPUT_FUNC_C3OUT,
    OUTPUT_FUNC_REFCLK0,
    OUTPUT_FUNC_NO_CONNECT
} PORTS_REMAP_OUTPUT_FUNCTION;

```

Description

Remap Output Function Enumeration

This data type defines the different remap output function enumeration.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

5.6.17.6.5.15 PORTS_REMAP_OUTPUT_PIN Enumeration

C

```

typedef enum {
    OUTPUT_PIN_RPA0,
    OUTPUT_PIN_RPA1,
    OUTPUT_PIN_RPA2,
    OUTPUT_PIN_RPA3,
    OUTPUT_PIN_RPA4,
    OUTPUT_PIN_RPA8,
    OUTPUT_PIN_RPA9,
    OUTPUT_PIN_RPA14,
    OUTPUT_PIN_RPA15,
    OUTPUT_PIN_RPB0,
    OUTPUT_PIN_RPB1,
    OUTPUT_PIN_RPB2,
    OUTPUT_PIN_RPB3,
    OUTPUT_PIN_RPB4,
    OUTPUT_PIN_RPB5,
    OUTPUT_PIN_RPB6,
    OUTPUT_PIN_RPB7,
    OUTPUT_PIN_RPB8,
    OUTPUT_PIN_RPB9,
    OUTPUT_PIN_RPB10,
    OUTPUT_PIN_RPB11,
    OUTPUT_PIN_RPB13,
    OUTPUT_PIN_RPB14,
    OUTPUT_PIN_RPB15,
    OUTPUT_PIN_RPC0,
    OUTPUT_PIN_RPC1,
    OUTPUT_PIN_RPC2,
    OUTPUT_PIN_RPC3,
    OUTPUT_PIN_RPC4,
    OUTPUT_PIN_RPC5,
    OUTPUT_PIN_RPC6,
    OUTPUT_PIN_RPC7,
    OUTPUT_PIN_RPC8,
    OUTPUT_PIN_RPC9,
    OUTPUT_PIN_RPC13,

```

```

OUTPUT_PIN_RPC14,
OUTPUT_PIN_RPD0,
OUTPUT_PIN_RPD1,
OUTPUT_PIN_RPD2,
OUTPUT_PIN_RPD3,
OUTPUT_PIN_RPD4,
OUTPUT_PIN_RPD5,
OUTPUT_PIN_RPD6,
OUTPUT_PIN_RPD7,
OUTPUT_PIN_RPD8,
OUTPUT_PIN_RPD9,
OUTPUT_PIN_RPD10,
OUTPUT_PIN_RPD11,
OUTPUT_PIN_RPD12,
OUTPUT_PIN_RPD14,
OUTPUT_PIN_RPD15,
OUTPUT_PIN_RPE3,
OUTPUT_PIN_RPE5,
OUTPUT_PIN_RPE8,
OUTPUT_PIN_RPE9,
OUTPUT_PIN_RPF0,
OUTPUT_PIN_RPF1,
OUTPUT_PIN_RPF2,
OUTPUT_PIN_RPF3,
OUTPUT_PIN_RPF4,
OUTPUT_PIN_RPF5,
OUTPUT_PIN_RPF6,
OUTPUT_PIN_RPF7,
OUTPUT_PIN_RPF8,
OUTPUT_PIN_RPF12,
OUTPUT_PIN_RPF13,
OUTPUT_PIN_RPG0,
OUTPUT_PIN_RPG1,
OUTPUT_PIN_RPG6,
OUTPUT_PIN_RPG7,
OUTPUT_PIN_RPG8,
OUTPUT_PIN_RPG9,
REMAP_NOT_SUPPORTED
} PORTS_REMAP_OUTPUT_PIN;

```

Description

PORTS IO Output Pins enumeration

This data type defines the different Ports I/O output pins enumeration.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

5.6.17.6.5.16 PORTS_REMAP_PIN Enumeration**C**

```

typedef enum {
    PORTS_REMAP_PIN_RPI0,
    PORTS_REMAP_PIN_RPI1,
    PORTS_REMAP_PIN_RPI2,
    PORTS_REMAP_PIN_RPI3,
    PORTS_REMAP_PIN_RPI4,
    PORTS_REMAP_PIN_RPI5,
    PORTS_REMAP_PIN_RPI6,
    PORTS_REMAP_PIN_RPI7,
    PORTS_REMAP_PIN_RPI8,
    PORTS_REMAP_PIN_RPI9,
    PORTS_REMAP_PIN_RPI10,
    PORTS_REMAP_PIN_RPI11,

```

```
PORTS_REMAP_PIN_RPI12,  
PORTS_REMAP_PIN_RPI13,  
PORTS_REMAP_PIN_RPI14,  
PORTS_REMAP_PIN_RPI15,  
PORTS_REMAP_PIN_RPI16,  
PORTS_REMAP_PIN_RPI17,  
PORTS_REMAP_PIN_RPI18,  
PORTS_REMAP_PIN_RPI19,  
PORTS_REMAP_PIN_RPI20,  
PORTS_REMAP_PIN_RPI21,  
PORTS_REMAP_PIN_RPI22,  
PORTS_REMAP_PIN_RPI23,  
PORTS_REMAP_PIN_RPI24,  
PORTS_REMAP_PIN_RPI25,  
PORTS_REMAP_PIN_RPI26,  
PORTS_REMAP_PIN_RPI27,  
PORTS_REMAP_PIN_RPI28,  
PORTS_REMAP_PIN_RPI29,  
PORTS_REMAP_PIN_RPI30,  
PORTS_REMAP_PIN_RPI31,  
PORTS_REMAP_PIN_RPI32,  
PORTS_REMAP_PIN_RPI33,  
PORTS_REMAP_PIN_RPI34,  
PORTS_REMAP_PIN_RPI35,  
PORTS_REMAP_PIN_RPI36,  
PORTS_REMAP_PIN_RPI37,  
PORTS_REMAP_PIN_RPI38,  
PORTS_REMAP_PIN_RPI39,  
PORTS_REMAP_PIN_RPI40,  
PORTS_REMAP_PIN_RPI41,  
PORTS_REMAP_PIN_RPI42,  
PORTS_REMAP_PIN_RPI43,  
PORTS_REMAP_PIN_RPI44,  
PORTS_REMAP_PIN_RPI45,  
PORTS_REMAP_PIN_RP0,  
PORTS_REMAP_PIN_RP1,  
PORTS_REMAP_PIN_RP2,  
PORTS_REMAP_PIN_RP3,  
PORTS_REMAP_PIN_RP4,  
PORTS_REMAP_PIN_RP5,  
PORTS_REMAP_PIN_RP6,  
PORTS_REMAP_PIN_RP7,  
PORTS_REMAP_PIN_RP8,  
PORTS_REMAP_PIN_RP9,  
PORTS_REMAP_PIN_RP10,  
PORTS_REMAP_PIN_RP11,  
PORTS_REMAP_PIN_RP12,  
PORTS_REMAP_PIN_RP13,  
PORTS_REMAP_PIN_RP14,  
PORTS_REMAP_PIN_RP15,  
PORTS_REMAP_PIN_RP16,  
PORTS_REMAP_PIN_RP17,  
PORTS_REMAP_PIN_RP18,  
PORTS_REMAP_PIN_RP19,  
PORTS_REMAP_PIN_RP20,  
PORTS_REMAP_PIN_RP21,  
PORTS_REMAP_PIN_RP22,  
PORTS_REMAP_PIN_RP23,  
PORTS_REMAP_PIN_RP24,  
PORTS_REMAP_PIN_RP25,  
PORTS_REMAP_PIN_RP26,  
PORTS_REMAP_PIN_RP27,  
PORTS_REMAP_PIN_RP28,  
PORTS_REMAP_PIN_RP29,  
PORTS_REMAP_PIN_RP30,  
PORTS_REMAP_PIN_RP31
```

```
} PORTS_REMAP_PIN;
```

Description

Remappable input/output Enumeration

This data type defines the different remappable input/output enumeration

Members

Members	Description
PORTS_REMAP_PIN_RPI0	Remappable input RPI0
PORTS_REMAP_PIN_RPI1	Remappable input RPI1
PORTS_REMAP_PIN_RPI2	Remappable input RPI2
PORTS_REMAP_PIN_RPI3	Remappable input RPI3
PORTS_REMAP_PIN_RPI4	Remappable input RPI4
PORTS_REMAP_PIN_RPI5	Remappable input RPI5
PORTS_REMAP_PIN_RPI6	Remappable input RPI6
PORTS_REMAP_PIN_RPI7	Remappable input RPI7
PORTS_REMAP_PIN_RPI8	Remappable input RPI8
PORTS_REMAP_PIN_RPI9	Remappable input RPI9
PORTS_REMAP_PIN_RPI10	Remappable input RPI10
PORTS_REMAP_PIN_RPI11	Remappable input RPI11
PORTS_REMAP_PIN_RPI12	Remappable input RPI12
PORTS_REMAP_PIN_RPI13	Remappable input RPI13
PORTS_REMAP_PIN_RPI14	Remappable input RPI14
PORTS_REMAP_PIN_RPI15	Remappable input RPI15
PORTS_REMAP_PIN_RPI16	Remappable input RPI16
PORTS_REMAP_PIN_RPI17	Remappable input RPI17
PORTS_REMAP_PIN_RPI18	Remappable input RPI18
PORTS_REMAP_PIN_RPI19	Remappable input RPI19
PORTS_REMAP_PIN_RPI20	Remappable input RPI20
PORTS_REMAP_PIN_RPI21	Remappable input RPI21
PORTS_REMAP_PIN_RPI22	Remappable input RPI22
PORTS_REMAP_PIN_RPI23	Remappable input RPI23
PORTS_REMAP_PIN_RPI24	Remappable input RPI24
PORTS_REMAP_PIN_RPI25	Remappable input RPI25
PORTS_REMAP_PIN_RPI26	Remappable input RPI26
PORTS_REMAP_PIN_RPI27	Remappable input RPI27
PORTS_REMAP_PIN_RPI28	Remappable input RPI28
PORTS_REMAP_PIN_RPI29	Remappable input RPI29
PORTS_REMAP_PIN_RPI30	Remappable input RPI30
PORTS_REMAP_PIN_RPI31	Remappable input RPI31
PORTS_REMAP_PIN_RPI32	Remappable input RPI32
PORTS_REMAP_PIN_RPI33	Remappable input RPI33
PORTS_REMAP_PIN_RPI34	Remappable input RPI34
PORTS_REMAP_PIN_RPI35	Remappable input RPI35
PORTS_REMAP_PIN_RPI36	Remappable input RPI36
PORTS_REMAP_PIN_RPI37	Remappable input RPI37

PORTS_REMAP_PIN_RPI38	Remappable input RPI38
PORTS_REMAP_PIN_RPI39	Remappable input RPI39
PORTS_REMAP_PIN_RPI40	Remappable input RPI40
PORTS_REMAP_PIN_RPI41	Remappable input RPI41
PORTS_REMAP_PIN_RPI42	Remappable input RPI42
PORTS_REMAP_PIN_RPI43	Remappable input RPI43
PORTS_REMAP_PIN_RPI44	Remappable input RPI44
PORTS_REMAP_PIN_RPI45	Remappable input RPI45
PORTS_REMAP_PIN_RP0	Remappable output RP0
PORTS_REMAP_PIN_RP1	Remappable output RP1
PORTS_REMAP_PIN_RP2	Remappable output RP2
PORTS_REMAP_PIN_RP3	Remappable output RP3
PORTS_REMAP_PIN_RP4	Remappable output RP4
PORTS_REMAP_PIN_RP5	Remappable output RP5
PORTS_REMAP_PIN_RP6	Remappable output RP6
PORTS_REMAP_PIN_RP7	Remappable output RP7
PORTS_REMAP_PIN_RP8	Remappable output RP8
PORTS_REMAP_PIN_RP9	Remappable output RP9
PORTS_REMAP_PIN_RP10	Remappable output RP10
PORTS_REMAP_PIN_RP11	Remappable output RP11
PORTS_REMAP_PIN_RP12	Remappable output RP12
PORTS_REMAP_PIN_RP13	Remappable output RP13
PORTS_REMAP_PIN_RP14	Remappable output RP14
PORTS_REMAP_PIN_RP15	Remappable output RP15
PORTS_REMAP_PIN_RP16	Remappable output RP16
PORTS_REMAP_PIN_RP17	Remappable output RP17
PORTS_REMAP_PIN_RP18	Remappable output RP18
PORTS_REMAP_PIN_RP19	Remappable output RP19
PORTS_REMAP_PIN_RP20	Remappable output RP20
PORTS_REMAP_PIN_RP21	Remappable output RP21
PORTS_REMAP_PIN_RP22	Remappable output RP22
PORTS_REMAP_PIN_RP23	Remappable output RP23
PORTS_REMAP_PIN_RP24	Remappable output RP24
PORTS_REMAP_PIN_RP25	Remappable output RP25
PORTS_REMAP_PIN_RP26	Remappable output RP26
PORTS_REMAP_PIN_RP27	Remappable output RP27
PORTS_REMAP_PIN_RP28	Remappable output RP28
PORTS_REMAP_PIN_RP29	Remappable output RP29
PORTS_REMAP_PIN_RP30	Remappable output RP30
PORTS_REMAP_PIN_RP31	Remappable output RP31

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all parts

5.6.17.6.6 Feature Existence Functions

5.6.17.6.6.1 PLIB_PORTS_ExistsChangeNotice Function

C

```
bool PLIB_PORTS_ExistsChangeNotice(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the ChangeNotice feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_ChangeNoticeEnable](#)
- [PLIB_PORTS_ChangeNoticeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChangeNotice feature is supported on the device
- false - The ChangeNotice feature is not supported on the device

Remarks

None.

5.6.17.6.6.2 PLIB_PORTS_ExistsChangeNoticeInIdle Function

C

```
bool PLIB_PORTS_ExistsChangeNoticeInIdle(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the ChangeNoticeInIdle feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_ChangeNoticeInIdleEnable](#)
- [PLIB_PORTS_ChangeNoticeInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChangeNoticeInIdle feature is supported on the device
- false - The ChangeNoticeInIdle feature is not supported on the device

Remarks

None.

5.6.17.6.6.3 PLIB_PORTS_ExistsChangeNoticePerPortInIdle Function**C**

```
bool PLIB_PORTS_ExistsChangeNoticePerPortInIdle(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the ChangeNoticeInIdlePerPort feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_ChangeNoticeInIdlePerPortEnable](#)
- [PLIB_PORTS_ChangeNoticeInIdlePerPortDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChangeNoticeInIdlePerPort feature is supported on the device
- false - The ChangeNoticeInIdlePerPort feature is not supported on the device

Remarks

None.

5.6.17.6.6.4 PLIB_PORTS_ExistsChangeNoticePerPortStatus Function**C**

```
bool PLIB_PORTS_ExistsChangeNoticePerPortStatus(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the ChangeNoticePerPortStatus feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_ChangeNoticePerPortHasOccured](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChangeNoticePerPortStatus feature is supported on the device

- false - The ChangeNoticePerPortStatus feature is not supported on the device

Remarks

None.

5.6.17.6.6.5 PLIB_PORTS_ExistsChangeNoticePerPortTurnOn Function**C**

```
bool PLIB_PORTS_ExistsChangeNoticePerPortTurnOn(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the ChangeNoticePerPortTurnOn feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_ChangeNoticePerPortTurnOn](#)
- [PLIB_PORTS_ChangeNoticePerPortTurnOff](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChangeNoticePerPortTurnOn feature is supported on the device
- false - The ChangeNoticePerPortTurnOn feature is not supported on the device

Remarks

None.

5.6.17.6.6.6 PLIB_PORTS_ExistsChangeNoticePullDownPerPort Function**C**

```
bool PLIB_PORTS_ExistsChangeNoticePullDownPerPort(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the ChangeNoticePullDownPerPort feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_ChangeNoticePullDownPerPortEnable](#)
- [PLIB_PORTS_ChangeNoticePullDownPerPortDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChangeNoticePullDownPerPort feature is supported on the device
- false - The ChangeNoticePullDownPerPort feature is not supported on the device

Remarks

None.

5.6.17.6.6.7 PLIB_PORTS_ExistsChangeNoticePullUp Function**C**

```
bool PLIB_PORTS_ExistsChangeNoticePullUp(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the ChangeNoticePullup feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_ChangeNoticePullUpEnable](#)
- [PLIB_PORTS_ChangeNoticePullUpDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChangeNoticePullup feature is supported on the device
- false - The ChangeNoticePullup feature is not supported on the device

Remarks

None.

5.6.17.6.6.8 PLIB_PORTS_ExistsChangeNoticePullUpPerPort Function**C**

```
bool PLIB_PORTS_ExistsChangeNoticePullUpPerPort(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the ChangeNoticePullUpPerPort feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_ChangeNoticePullUpPerPortEnable](#)
- [PLIB_PORTS_ChangeNoticePullUpPerPortDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChangeNoticePullUpPerPort feature is supported on the device
- false - The ChangeNoticePullUpPerPort feature is not supported on the device

Remarks

None.

5.6.17.6.6.9 PLIB_PORTS_ExistsPinChangeNotice Function**C**

```
bool PLIB_PORTS_ExistsPinChangeNotice(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the PinChangeNotice feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_PinChangeNoticeEnable](#)
- [PLIB_PORTS_PinChangeNoticeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PinChangeNotice feature is supported on the device
- false - The PinChangeNotice feature is not supported on the device

Remarks

None.

5.6.17.6.6.10 PLIB_PORTS_ExistsPinChangeNoticePerPort Function**C**

```
bool PLIB_PORTS_ExistsPinChangeNoticePerPort(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the PinChangeNoticePerPort feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_PinChangeNoticePerPortEnable](#)
- [PLIB_PORTS_PinChangeNoticePerPortDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PinChangeNoticePerPort feature is supported on the device
- false - The PinChangeNoticePerPort feature is not supported on the device

Remarks

None.

5.6.17.6.6.11 PLIB_PORTS_ExistsPinMode Function**C**

```
bool PLIB_PORTS_ExistsPinMode(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the PinMode feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_PinModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PinMode feature is supported on the device
- false - The PinMode feature is not supported on the device

Remarks

None.

5.6.17.6.6.12 PLIB_PORTS_ExistsPinModePerPort Function**C**

```
bool PLIB_PORTS_ExistsPinModePerPort(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the PinModePerPort feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_PinModePerPortSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PinModePerPort feature is supported on the device
- false - The PinModePerPort feature is not supported on the device

Remarks

None.

5.6.17.6.6.13 PLIB_PORTS_ExistsPortsDirection Function**C**

```
bool PLIB_PORTS_ExistsPortsDirection(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the PortsDirection feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_PinDirectionInputSet](#)
- [PLIB_PORTS_PinDirectionOutputSet](#)
- [PLIB_PORTS_DirectionInputSet](#)
- [PLIB_PORTS_DirectionOutputSet](#)
- [PLIB_PORTS_DirectionGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PortsDirection feature is supported on the device
- false - The PortsDirection feature is not supported on the device

Remarks

None.

5.6.17.6.6.14 PLIB_PORTS_ExistsPortsOpenDrain Function**C**

```
bool PLIB_PORTS_ExistsPortsOpenDrain(  
    PORTS_MODULE_ID index  
);
```


Description

This function identifies whether the PortsOpenDrain feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_PinOpenDrainEnable](#)
- [PLIB_PORTS_PinOpenDrainDisable](#)
- [PLIB_PORTS_OpenDrainEnable](#)
- [PLIB_PORTS_OpenDrainDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PortsOpenDrain feature is supported on the device
- false - The PortsOpenDrain feature is not supported on the device

Remarks

None.

5.6.17.6.6.15 PLIB_PORTS_ExistsPortsRead Function**C**

```
bool PLIB_PORTS_ExistsPortsRead(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the PortsRead feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_PinGet](#)
- [PLIB_PORTS_Read](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PortsRead feature is supported on the device
- false - The PortsRead feature is not supported on the device

Remarks

None.

5.6.17.6.6.16 PLIB_PORTS_ExistsPortsWrite Function

C

```
bool PLIB_PORTS_ExistsPortsWrite(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the PortsWrite feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_PinWrite](#)
- [PLIB_PORTS_PinSet](#)
- [PLIB_PORTS_PinClear](#)
- [PLIB_PORTS_PinToggle](#)
- [PLIB_PORTS_Write](#)
- [PLIB_PORTS_Set](#)
- [PLIB_PORTS_Toggle](#)
- [PLIB_PORTS_Clear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PortsWrite feature is supported on the device
- false - The PortsWrite feature is not supported on the device

Remarks

None.

5.6.17.6.6.17 PLIB_PORTS_ExistsRemapInput Function

C

```
bool PLIB_PORTS_ExistsRemapInput(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the RemapInput feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_RemapInput](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RemapInput feature is supported on the device
- false - The RemapInput feature is not supported on the device

Remarks

None.

5.6.17.6.6.18 PLIB_PORTS_ExistsRemapOutput Function

C

```
bool PLIB_PORTS_ExistsRemapOutput(  
    PORTS_MODULE_ID index  
);
```

Description

This function identifies whether the RemapOutput feature is available on the PORTS module. When this function returns true, these functions are supported on the device:

- [PLIB_PORTS_RemapOutput](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RemapOutput feature is supported on the device
- false - The RemapOutput feature is not supported on the device

Remarks

None.

5.6.17.7 Files

Files

Name	Description
plib_ports.h	Ports Peripheral Library Interface header for Ports function definitions.

Description

5.6.17.7.1 plib_ports.h

Ports Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Ports Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Ports peripheral.

Functions

	Name	Description
≡	<u>PLIB_PORTS_ChangeNoticeDisable</u>	Global Change Notice disable.
≡	<u>PLIB_PORTS_ChangeNoticeEnable</u>	Global Change Notice enable.
≡	<u>PLIB_PORTS_ChangeNoticeInIdleDisable</u>	CPU Idle does not affect Change Notice operation.
≡	<u>PLIB_PORTS_ChangeNoticeInIdleEnable</u>	CPU Idle mode halts Change Notice operation.
≡	<u>PLIB_PORTS_ChangeNoticeInIdlePerPortDisable</u>	Enables the selected pin as analog or digital.
≡	<u>PLIB_PORTS_ChangeNoticeInIdlePerPortEnable</u>	Enables the selected pin as analog or digital.
≡	<u>PLIB_PORTS_ChangeNoticePerPortHasOccured</u>	Enables the selected pin as analog or digital.
≡	<u>PLIB_PORTS_ChangeNoticePerPortTurnOff</u>	Enables the selected pin as analog or digital.
≡	<u>PLIB_PORTS_ChangeNoticePerPortTurnOn</u>	Enables the selected pin as analog or digital.
≡	<u>PLIB_PORTS_ChangeNoticePullDownPerPortDisable</u>	Enables the selected pin as analog or digital.
≡	<u>PLIB_PORTS_ChangeNoticePullDownPerPortEnable</u>	Enables the selected pin as analog or digital.
≡	<u>PLIB_PORTS_ChangeNoticePullUpDisable</u>	Disable pull-up on input change.
≡	<u>PLIB_PORTS_ChangeNoticePullUpEnable</u>	Enable pull-up on input change.
≡	<u>PLIB_PORTS_ChangeNoticePullUpPerPortDisable</u>	Enables the selected pin as analog or digital.
≡	<u>PLIB_PORTS_ChangeNoticePullUpPerPortEnable</u>	Enables the selected pin as analog or digital.
≡	<u>PLIB_PORTS_Clear</u>	Clears the selected digital port/latch.
≡	<u>PLIB_PORTS_DirectionGet</u>	Reads the direction of the selected digital port.
≡	<u>PLIB_PORTS_DirectionInputSet</u>	Enables the read (input) direction for the selected port.
≡	<u>PLIB_PORTS_DirectionOutputSet</u>	Enables the write (output) direction for the selected port.
≡	<u>PLIB_PORTS_ExistsChangeNotice</u>	Identifies whether the ChangeNotice feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsChangeNoticeInIdle</u>	Identifies whether the ChangeNoticeInIdle feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsChangeNoticePerPortInIdle</u>	Identifies whether the ChangeNoticeInIdlePerPort feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsChangeNoticePerPortStatus</u>	Identifies whether the ChangeNoticePerPortStatus feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsChangeNoticePerPortTurnOn</u>	Identifies whether the ChangeNoticePerPortTurnOn feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsChangeNoticePullDownPerPort</u>	Identifies whether the ChangeNoticePullDownPerPort feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsChangeNoticePullUp</u>	Identifies whether the ChangeNoticePullup feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsChangeNoticePullUpPerPort</u>	Identifies whether the ChangeNoticePullUpPerPort feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsPinChangeNotice</u>	Identifies whether the PinChangeNotice feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsPinChangeNoticePerPort</u>	Identifies whether the PinChangeNoticePerPort feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsPinMode</u>	Identifies whether the PinMode feature exists on the PORTS module
≡	<u>PLIB_PORTS_ExistsPinModePerPort</u>	Identifies whether the PinModePerPort feature exists on the PORTS module

	PLIB_PORTS_ExistsPortsDirection	Identifies whether the PortsDirection feature exists on the PORTS module
	PLIB_PORTS_ExistsPortsOpenDrain	Identifies whether the PortsOpenDrain feature exists on the PORTS module
	PLIB_PORTS_ExistsPortsRead	Identifies whether the PortsRead feature exists on the PORTS module
	PLIB_PORTS_ExistsPortsWrite	Identifies whether the PortsWrite feature exists on the PORTS module
	PLIB_PORTS_ExistsRemapInput	Identifies whether the RemapInput feature exists on the PORTS module
	PLIB_PORTS_ExistsRemapOutput	Identifies whether the RemapOutput feature exists on the PORTS module
	PLIB_PORTS_OpenDrainDisable	Disables the open drain functionality for the selected port.
	PLIB_PORTS_OpenDrainEnable	Enables the open drain functionality for the selected port.
	PLIB_PORTS_PinChangeNoticeDisable	Port pin Change Notice disable.
	PLIB_PORTS_PinChangeNoticeEnable	Port pin Change Notice enable.
	PLIB_PORTS_PinChangeNoticePerPortDisable	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinChangeNoticePerPortEnable	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinClear	Clears the selected digital pin/latch.
	PLIB_PORTS_PinDirectionInputSet	Enables the read (input) direction for the selected pin.
	PLIB_PORTS_PinDirectionOutputSet	Enables the write (output) direction for the selected pin.
	PLIB_PORTS_PinGet	Reads/Gets data from the selected digital pin.
	PLIB_PORTS_PinModePerPortSelect	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinModeSelect	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinOpenDrainDisable	Disables the open drain functionality for the selected pin.
	PLIB_PORTS_PinOpenDrainEnable	Enables the open drain functionality for the selected pin.
	PLIB_PORTS_PinSet	Sets the selected digital pin/latch.
	PLIB_PORTS_PinToggle	Toggles the selected digital pin/latch.
	PLIB_PORTS_PinWrite	Writes the selected digital pin/latch.
	PLIB_PORTS_Read	Reads the selected digital port.
	PLIB_PORTS_RemapInput	Input/Output (I/O) function remapping.
	PLIB_PORTS_RemapOutput	Input/Output (I/O) function remapping.
	PLIB_PORTS_Set	Writes the selected digital port/latch based on the mask.
	PLIB_PORTS_Toggle	Toggles the selected digital port/latch.
	PLIB_PORTS_Write	Writes the selected digital port/latch.

Types

	Name	Description
	PORTS_DATA_MASK	Data type defining the PORTS data mask
	PORTS_DATA_TYPE	Data type defining the PORTS data type.

File Name

plib_ports.h

Company

Microchip Technology Inc.

5.6.18 Power Peripheral Library

5.6.18.1 Introduction

Power Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Power Controller modules on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

Power Controller Overview

The Power Controller is a key component of a microcontroller. Power control features allow the user to obtain the balance of power consumption and performance that is best suited to their application.

The features provided in this library discuss specific software commands that are used dynamically to place the device in low-power (i.e., power-saving) modes, which disable power and clocking of selected features. When implementing a system to achieve the lowest possible power consumption while maintain the needed performance, the oscillator configuration and clocking of all peripherals must also be considered. Oscillator configuration and peripheral clocking are addressed in the Oscillator Peripheral Library. In addition, each peripheral may have its own low-power mode settings that are controlled by the library of that peripheral. The user is advised to refer to the libraries of the peripherals used for their particular application.

Key features present on a power controller include (a microcontroller can support one or more of these power features):

- **Power Source Configuration:** Provides the ability to enable and disable a source that may regulate the power consumption in the device
- **Power Status Flags:** Indicate the status of the particular power feature in the device

Note: Please refer to the the specific device data sheet to determine the exact set of supported features.

5.6.18.2 Release Notes

MPLAB Harmony Version: v0.70b **Power Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.18.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.18.4 Using the Library

Using the Library

This topic describes the basic architecture of the Power Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_power.h](#)

The interface to the Power Peripheral Library is defined in the "[plib_power.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the Power Peripheral Library must include "peripheral.h".

Peripheral Module IDs

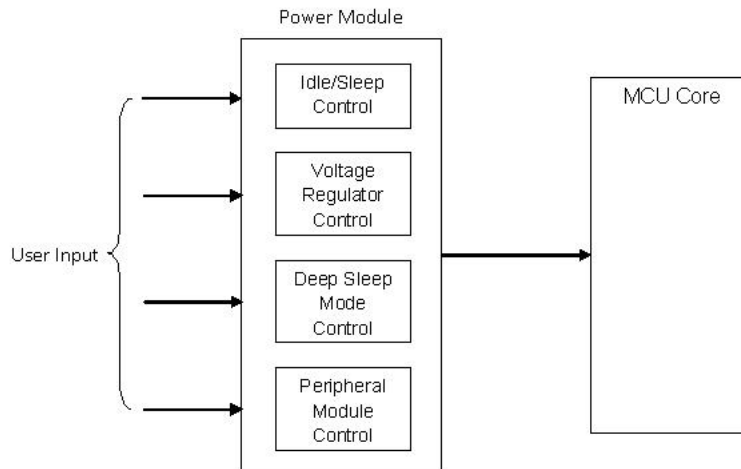
Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. To support this, the first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). Not all microcontrollers will have all instances of the module listed in this enumeration. Please refer to the specific data sheet to determine availability.

5.6.18.4.1 Hardware Abstraction Model

This library provides a low-level abstraction of the Power Controller module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Block Diagram



This section discusses the various power-saving features implemented in hardware. Combinations of these methods can be used to selectively tailor an application's power consumption, while still maintaining critical or timing-sensitive application features.

Note: Not all devices support all of the features discussed in this section. Please refer to specific device data sheet to determine the supported features for your device.

Instruction-based Power-Saving Modes

In addition to full-power operation, otherwise known as Run mode, PIC microcontrollers also provide other instruction-based power-saving modes. By powering down to different modes, different functional areas of the microcontroller allow progressive reductions of operating and idle power consumption. In addition, these modes can be tailored for more power reduction, but at a trade-off of some operating features.

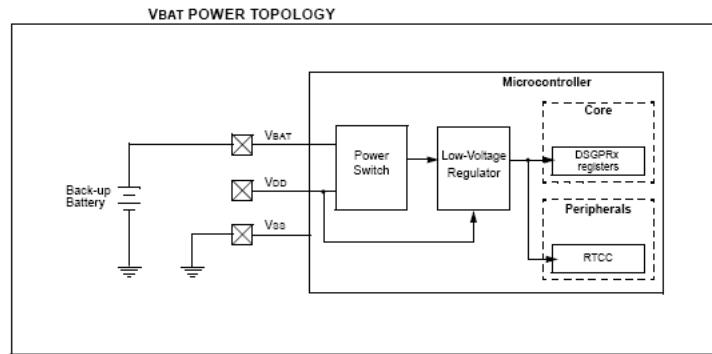
- **Idle Mode:** In Idle mode, the CPU is halted, but the System Clock (SYSCLK) source is still enabled. This allows peripherals to continue to operate when the CPU is halted. Peripherals can be individually configured to halt when entering Idle mode by setting their respective SIDL bit. Latency when exiting Idle mode is very low due to the CPU oscillator source remaining active.
- **Sleep Mode:** The CPU, SYSCLK source and any peripherals that operate on the system clock source are disabled
- **Deep Sleep Mode:** The CPU SYSCLK source, and all the peripherals, with the exception of the Real-Time Clock and Calendar (RTCC) and the Deep Sleep Watchdog Timer (DSWDT), are disabled. This is the lowest power mode for the device. The power to RAM and Flash is also disabled. Deep Sleep mode represents the lowest power mode available without removing power from the application.

The instruction-based power-saving modes are exited as a result of several different hardware triggers. When the device exits one of these three operating modes, it is said to 'wake-up'.

Hardware-based Power-Saving Mode

VBAT mode is a hardware-based power-saving mode that maintains only the most critical operations when a power loss occurs on VDD. The mode does this by powering the systems from a back-up power source connected to the VBAT pin. In this mode, the RTCC can run even when there is no power on VDD.

VBAT mode is entered whenever power is removed from VDD. An on-chip power switch detects the power loss from VDD and connects the VBAT pin to the low-voltage regulator. Entering VBAT mode requires that a power source, distinct from the main VDD power source, be available on the VBAT pin and that VDD be completely removed from the VDD pin(s). Removing VDD can be either unintentional, as in a power failure, or as part of a deliberate power reduction strategy.



Note: Not all devices support all of the features discussed in this section. Please refer to specific device data sheet to determine the supported features for your device.

Selective Peripheral Power Control

Sleep and Idle modes allow users to substantially reduce power consumption by slowing or stopping the CPU clock. Even so, peripheral modules may still remain clocked, and thus, consume some amount of power. There may be cases where the application needs what these modes do not provide: the ability to allocate limited power resources to the CPU while eliminating power consumption from the peripherals. PIC microcontrollers address this requirement by allowing peripheral modules to be selectively enabled or disabled, reducing or eliminating their power consumption.

In addition to the selective peripheral power control implemented as part of this library, devices will also include power saving features that are implemented in their specific libraries. These features include the ability to stop (shutdown) automatically when entering idle mode. On some peripherals the ability to continue operation in Sleep mode is available. Always review the libraries for the peripherals used by your application for specific and unique features that will help minimize power consumption.

Disabling Peripheral Modules

Most of the peripheral modules in the PIC microcontroller family architecture can be selectively disabled, reducing or essentially eliminating their power consumption during all operating modes.

All peripheral modules (except for I/O ports) also have a Control bit that can disable their functionality. These bits, known as the Peripheral Module Disable (PMD) bits, are generically named: "xxMD". These bits are located in the PMDx Special Function Registers (SFRs). In contrast to the module enable bits, the xxMD bit must be set to a '1' to disable the module.

While the PMD and module enable bits both disable a peripheral's functionality, the PMD bit completely shuts down the peripheral, effectively powering down all circuits and removing all clock sources. This has the additional effect of making any of the module's control and buffer registers, mapped in the SFR space, unavailable for operations. In other words, when the PMD bit is used to disable a module, the peripheral ceases to exist until the PMD bit is cleared.

The PMD bit is most useful in highly power-sensitive applications, where even tiny savings in power consumption can determine the ability of an application to function. In these cases, the bits can be set before the main body of the application to remove those peripherals that will not be needed.

5.6.18.4.2 Library Usage Model

Note: The interface provided is a superset of all power functionality available on the device. Refer to the "**Power-Saving Features**" chapter in the specific device data sheet for availability.

This library provides the low-level abstraction of the Power Controller module available on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

The library interface routines are divided into various subsections, each of the sub-sections addresses one of the blocks or the

overall operation of the Power Controller module.

Library Interface Section	Description
Initialization Functions	This section provides a set of functions for configuring the Power-Saving features and enabling or disabling the particular power feature.
Status Functions	This section provides a function to read the status of a particular power feature.

Note: Not all modes are available on all devices. Refer to the "Power-Saving Features" chapter in the specific device data sheet for availability.

5.6.18.4.3 How the Library Works

Usage of this library is partitioned into two major sections.

Initialization

The steps that are required to initialize the Power Controller module vary for different microcontrollers. Refer to the specific device data sheet to determine the correct initialization sequence. The following information provides a general overview.

Power Source Setup

- A module can be disabled for the microcontroller to be able to save power. Use the function `PLIB_POWER_ModuleDisable` to disable the particular module. `POWER_MODULE_DISABLE` provides a list of possible modules sources.
- On most microcontrollers, the voltage regulator can be controlled during Sleep mode. It can be enabled (turned on) using [PLIB_POWER_VoltageRegulatorEnable](#) and disabled (turned off) using [PLIB_POWER_VoltageRegulatorDisable](#).
- On some microcontrollers, Deep Sleep mode is enabled using [PLIB_POWER_DeepSleepModeEnable](#) and Brown-out Reset in Deep Sleep mode is enabled using [PLIB_POWER_DeepSleepBOREnable](#). The Deep Sleep mode is disabled using [PLIB_POWER_DeepSleepModeDisable](#) and Brown-out Reset in Deep Sleep mode is disabled using [PLIB_POWER_DeepSleepBORDisable](#).

Power Status

Status of the particular power feature indicates the previous status of that feature. This feature can be used to determine whether the device was in Sleep or Idle mode before waking up. It also allows software to clear the previous status of the device.

5.6.18.5 Configuring the Library

The library is configured for the supported Power Controller modules when the processor is chosen in the MPLAB IDE.

5.6.18.6 Library Interface

Data Types and Constants

	Name	Description
	POWER_MODULE	List of the modules whose power can be controlled.
	POWER_MODULE_ID	Identifies the supported Power modules.

Feature Existence Functions

	Name	Description
	PLIB_POWER_ExistsDeepSleepBOR	Identifies whether the DeepSleepBORControl feature exists on the POWER module

	PLIB_POWER_ExistsDeepSleepFaultDetect	Identifies whether the DeepSleepFaultDetectStatus feature exists on the POWER module
	PLIB_POWER_ExistsDeepSleepInterruptOnChange	Identifies whether the DeepSleepIOCStatus feature exists on the POWER module
	PLIB_POWER_ExistsDeepSleepMCLREvent	Identifies whether the DeepSleepMCLRStatus feature exists on the POWER module
	PLIB_POWER_ExistsDeepSleepMode	Identifies whether the DeepSleepModeControl feature exists on the POWER module
	PLIB_POWER_ExistsDeepSleepPowerOnReset	Identifies whether the DeepSleepPORStatus feature exists on the POWER module
	PLIB_POWER_ExistsIdleStatus	Identifies whether the IdleStatus feature exists on the POWER module
	PLIB_POWER_ExistsPeripheralModuleControl	Identifies whether the PeripheralModuleControl feature exists on the POWER module
	PLIB_POWER_ExistsSleepStatus	Identifies whether the SleepStatus feature exists on the POWER module
	PLIB_POWER_ExistsVoltageRegulatorControl	Identifies whether the VoltageRegulatorControl feature exists on the POWER module

Initialization Functions

	Name	Description
	PLIB_POWER_DeepSleepBORDisable	Disables the Deep Sleep BOR.
	PLIB_POWER_DeepSleepBOREnable	Enables the Deep Sleep BOR mode.
	PLIB_POWER_DeepSleepModeDisable	Disables the Deep Sleep mode.
	PLIB_POWER_DeepSleepModeEnable	Enables the Deep Sleep mode.
	PLIB_POWER_PeripheralModuleDisable	Disable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleEnable	Enable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleIsEnabled	Check if selected peripheral is enabled or not
	PLIB_POWER_VoltageRegulatorDisable	Disables the voltage regulator during Sleep mode.
	PLIB_POWER_VoltageRegulatorEnable	Enable the voltage regulator during Sleep mode.

Status Functions

	Name	Description
	PLIB_POWER_ClearIdleStatus	Clears the Idle mode status of the device.
	PLIB_POWER_ClearSleepStatus	Clear the Sleep mode status bit of the device.
	PLIB_POWER_DeepSleepFaultDetectStatus	Returns the status of Deep Sleep Fault Detect bit.
	PLIB_POWER_DeepSleepInterruptOnChangeStatus	Returns the status of the Deep Sleep Interrupt-on-Change bit.
	PLIB_POWER_DeepSleepMCLREventStatus	Returns the status of the Deep Sleep MCLR Event bit.
	PLIB_POWER_DeepSleepPowerOnResetStatus	Returns the status of Deep Sleep Power-on Reset bit.
	PLIB_POWER_DeviceWasInIdleMode	Returns the Idle mode status of the device.
	PLIB_POWER_DeviceWasInSleepMode	Returns the Sleep mode status of the device.
	PLIB_POWER_VoltageRegulatorIsEnabled	Provides the status of the voltage regulator during Sleep mode.

Description

This section lists and describes the functions, data types, and constants provided in the Power Peripheral Library.

5.6.18.6.1 Initialization Functions

5.6.18.6.1.1 PLIB_POWER_DeepSleepBORDisable Function

C

```
void PLIB_POWER_DeepSleepBORDisable(  
    POWER_MODULE_ID index  
);
```

Description

This function disables the Deep Sleep BOR. The Deep Sleep BOR is independent of the standard BOR and a Deep Sleep BOR event will not cause a wake-up from Deep Sleep.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsDeepSleepBOR](#) in your application to determine whether this feature is available.

Example

```
PLIB_POWER_DeepSleepBORDisable(POWER_ID_0);
```

5.6.18.6.1.2 PLIB_POWER_DeepSleepBOREnable Function

C

```
void PLIB_POWER_DeepSleepBOREnable(  
    POWER_MODULE_ID index  
);
```

Description

This function enables the Deep Sleep BOR mode. The Deep Sleep BOR is independent of the standard BOR and a Deep Sleep BOR event will not cause a wake-up from Deep Sleep.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsDeepSleepBOR](#) in your application to determine whether this feature is available.

Example

```
PLIB_POWER_DeepSleepBOREnable(POWER_ID_0);
```

5.6.18.6.1.3 PLIB_POWER_DeepSleepModeDisable Function**C**

```
void PLIB_POWER_DeepSleepModeDisable(  
    POWER_MODULE_ID index  
);
```

Description

This function disables the Deep Sleep mode. Deep Sleep mode is intended to provide the lowest levels of power consumption available without requiring the use of external switches to completely remove all power from the device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsDeepSleepMode](#) in your application to determine whether this feature is available.

Example

```
PLIB_POWER_DeepSleepModeDisable(POWER_ID_0);
```

5.6.18.6.1.4 PLIB_POWER_DeepSleepModeEnable Function**C**

```
void PLIB_POWER_DeepSleepModeEnable(  
    POWER_MODULE_ID index  
);
```

Description

This function enables the Deep Sleep mode. Deep Sleep mode is intended to provide the lowest levels of power consumption available without requiring the use of external switches to completely remove all power from the device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_POWER_ExistsDeepSleepMode](#) in your application to determine whether this feature is available.

Example

```
PLIB_POWER_DeepSleepModeEnable(POWER_ID_0);
```

5.6.18.6.1.5 PLIB_POWER_PeripheralModuleDisable Function**C**

```
void PLIB_POWER_PeripheralModuleDisable(  
    POWER_MODULE_ID index,  
    POWER_MODULE source  
);
```

Description

This function completely shuts down the selected peripheral, effectively powering down all circuits and removing all clock sources. This has the additional effect of making any of the module's control and buffer registers, which are mapped in the SFR space, unavailable for operations.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the module to be disabled from POWER_MODULE enum

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsPeripheralModuleControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_POWER_PeripheralModuleDisable (POWER_ID_0, POWER_MODULE_ADC);
```

5.6.18.6.1.6 PLIB_POWER_PeripheralModuleEnable Function**C**

```
void PLIB_POWER_PeripheralModuleEnable(  
    POWER_MODULE_ID index,  
    POWER_MODULE source  
);
```

Description

This function enables the power supply back to the selected module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the module to be enabled from POWER_MODULE enum

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsPeripheralModuleControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_POWER_PeripheralModuleEnable (POWER_ID_0, POWER_MODULE_ADC);
```

5.6.18.6.1.7 PLIB_POWER_PeripheralModuleIsEnabled Function**C**

```
bool PLIB_POWER_PeripheralModuleIsEnabled(  
    POWER_MODULE_ID index,  
    POWER_MODULE source  
);
```

Description

This function checks if selected peripheral is enabled or not.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the module to be enabled from POWER_MODULE enum

Returns

- true - Power of the selected peripheral is enabled
- false - Power of the selected peripheral is disabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsPeripheralModuleControl](#) in your application to determine whether this feature is available.

Example

```
if (PLIB_POWER_PeripheralModuleIsEnabled ( POWER_MODULE_0, POWER_MODULE_ADC ))  
{  
  
}
```

5.6.18.6.1.8 PLIB_POWER_VoltageRegulatorDisable Function**C**

```
void PLIB_POWER_VoltageRegulatorDisable(  
    POWER_MODULE_ID index  
);
```

Description

This function disables the voltage regulator into during Sleep mode for the selected device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsVoltageRegulatorControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_POWER_VoltageRegulatorDisable(POWER_ID_0);
```

5.6.18.6.1.9 PLIB_POWER_VoltageRegulatorEnable Function

C

```
void PLIB_POWER_VoltageRegulatorEnable(  
    POWER_MODULE_ID index  
);
```

Description

This function enables the voltage regulator during Sleep mode for the selected device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsVoltageRegulatorControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_POWER_VoltageRegulatorEnable(POWER_ID_0);
```

5.6.18.6.2 Status Functions

5.6.18.6.2.1 PLIB_POWER_ClearIdleStatus Function

C

```
void PLIB_POWER_ClearIdleStatus(  
    POWER_MODULE_ID index  
);
```

Description

This function clears the Idle status bit of the device if it was previously in Idle mode. However, it does not mean that it wakes up the device from Idle.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsIdleStatus](#) in your application to determine whether this feature is available.

Example

```
if(PLIB_POWER_DeviceWasInIdleMode(POWER_ID_0))
{
    PLIB_POWER_ClearIdleStatus (POWER_ID_0);
}
```

5.6.18.6.2.2 PLIB_POWER_ClearSleepStatus Function**C**

```
void PLIB_POWER_ClearSleepStatus(
    POWER_MODULE_ID index
);
```

Description

This function clears the Sleep status bit of the device if it was previously in Sleep mode. However, it does not mean that it wakes up the device from sleep.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsSleepStatus](#) in your application to determine whether this feature is available.

Example

```
if(PLIB_POWER_DeviceWasInSleepMode(POWER_ID_0))
{
    PLIB_POWER_ClearSleepStatus (POWER_ID_0);
}
```

5.6.18.6.2.3 PLIB_POWER_DeepSleepFaultDetectStatus Function**C**

```
bool PLIB_POWER_DeepSleepFaultDetectStatus(
```

```
    POWER_MODULE_ID index
);
```

Description

This function returns the status of the Deep Sleep Fault Detect bit.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

true - A Fault occurred during Deep Sleep, and some Deep Sleep configuration settings may have been corrupted
false - No Fault was detected during Deep Sleep

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsDeepSleepFaultDetect](#) in your application to determine whether this feature is available.

Example

```
if(PLIB_POWER_DeepSleepFaultDetectStatus(POWER_ID_0))
{
}
}
```

5.6.18.6.2.4 PLIB_POWER_DeepSleepInterruptOnChangeStatus Function

C

```
bool PLIB_POWER_DeepSleepInterruptOnChangeStatus(
    POWER_MODULE_ID index
);
```

Description

This function returns the status of the Deep Sleep Interrupt-on-Change bit.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Interrupt-on-Change was asserted during Deep Sleep
- false - Interrupt-on-Change was not asserted during Deep Sleep

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsDeepSleepInterruptOnChange](#) in your application to determine whether this feature is available.

Example

```
if(PLIB_POWER_DeepSleepInterruptOnChangeStatus(POWER_ID_0))
{
}
```

```
}
```

5.6.18.6.2.5 PLIB_POWER_DeepSleepMCLREventStatus Function

C

```
bool PLIB_POWER_DeepSleepMCLREventStatus(  
    POWER_MODULE_ID index  
);
```

Description

This function returns the status of the Deep Sleep MCLR Event bit.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The MCLR pin was active and was asserted during Deep Sleep
- false - The MCLR pin was not active, or was active, but not asserted during Deep Sleep

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsDeepSleepMCLREvent](#) in your application to determine whether this feature is available.

Example

```
if(PLIB_POWER_DeepSleepMCLREventStatus(POWER_ID_0))  
{  
  
}
```

5.6.18.6.2.6 PLIB_POWER_DeepSleepPowerOnResetStatus Function

C

```
bool PLIB_POWER_DeepSleepPowerOnResetStatus(  
    POWER_MODULE_ID index  
);
```

Description

This function returns the status of the Deep Sleep Power-on Reset bit.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The VDD supply POR circuit was active and a POR event was detected
- false - The VDD supply POR circuit was not active, or was active but did not detect a POR event

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_POWER_ExistsDeepSleepPowerOnReset](#) in your application to determine whether this feature is available.

Example

```
if(PLIB_POWER_DeepSleepPowerOnResetStatus(POWER_ID_0))
{
}
```

5.6.18.6.2.7 PLIB_POWER_DeviceWasInIdleMode Function**C**

```
bool PLIB_POWER_DeviceWasInIdleMode(
    POWER_MODULE_ID index
);
```

Description

This function returns the Idle mode status of the device. It tells whether or not the device was in Idle mode before waking up.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The device was in Idle mode before waking up
- false - The device was not in Idle mode

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsIdleStatus](#) in your application to determine whether this feature is available.

Example

```
if(PLIB_POWER_DeviceWasInIdleMode(POWER_ID_0))
{
}
```

5.6.18.6.2.8 PLIB_POWER_DeviceWasInSleepMode Function**C**

```
bool PLIB_POWER_DeviceWasInSleepMode(
    POWER_MODULE_ID index
);
```

Description

This function returns the Sleep mode status of the device. It tells whether or not the device was in Sleep mode before waking up.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The device was in Sleep mode before waking up
- false - The device was not in Sleep mode

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsSleepStatus](#) in your application to determine whether this feature is available.

Example

```
if(PLIB_POWER_DeviceWasInSleepMode(POWER_ID_0))
{
}
```

5.6.18.6.2.9 PLIB_POWER_VoltageRegulatorIsEnabled Function**C**

```
bool PLIB_POWER_VoltageRegulatorIsEnabled(
    POWER_MODULE_ID index
);
```

Description

This function provides the status of the voltage regulator during Sleep mode for the selected device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The voltage regulator will be enabled in Sleep mode
- false - The voltage regulator will be disabled in Sleep mode

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_POWER_ExistsVoltageRegulatorControl](#) in your application to determine whether this feature is available.

Example

```
if (PLIB_POWER_VoltageRegulatorIsEnabled(POWER_ID_0))
{
}
```

5.6.18.6.3 Data Types and Constants**5.6.18.6.3.1 POWER_MODULE Enumeration****C**

```
typedef enum {
    POWER_MODULE_ADC1,
    POWER_MODULE_CTMU,
    POWER_MODULE_CVR,
```

```
POWER_MODULE_CMP1,  
POWER_MODULE_CMP2,  
POWER_MODULE_CMP3,  
POWER_MODULE_IC1,  
POWER_MODULE_IC2,  
POWER_MODULE_IC3,  
POWER_MODULE_IC4,  
POWER_MODULE_IC5,  
POWER_MODULE_IC6,  
POWER_MODULE_IC7,  
POWER_MODULE_IC8,  
POWER_MODULE_IC9,  
POWER_MODULE_OC1,  
POWER_MODULE_OC2,  
POWER_MODULE_OC3,  
POWER_MODULE_OC4,  
POWER_MODULE_OC5,  
POWER_MODULE_OC6,  
POWER_MODULE_OC7,  
POWER_MODULE_OC8,  
POWER_MODULE_OC9,  
POWER_MODULE_TMR1,  
POWER_MODULE_TMR2,  
POWER_MODULE_TMR3,  
POWER_MODULE_TMR4,  
POWER_MODULE_TMR5,  
POWER_MODULE_TMR6,  
POWER_MODULE_TMR7,  
POWER_MODULE_TMR8,  
POWER_MODULE_TMR9,  
POWER_MODULE_UART1,  
POWER_MODULE_UART2,  
POWER_MODULE_UART3,  
POWER_MODULE_UART4,  
POWER_MODULE_UART5,  
POWER_MODULE_UART6,  
POWER_MODULE_SPI1,  
POWER_MODULE_SPI2,  
POWER_MODULE_SPI3,  
POWER_MODULE_SPI4,  
POWER_MODULE_SPI5,  
POWER_MODULE_SPI6,  
POWER_MODULE_I2C1,  
POWER_MODULE_I2C2,  
POWER_MODULE_I2C3,  
POWER_MODULE_I2C4,  
POWER_MODULE_I2C5,  
POWER_MODULE_USB,  
POWER_MODULE_CAN1,  
POWER_MODULE_CAN2,  
POWER_MODULE_RTCC,  
POWER_MODULE_REF_CLK_OUTPUT,  
POWER_MODULE_REF_CLK_OUTPUT1,  
POWER_MODULE_REF_CLK_OUTPUT2,  
POWER_MODULE_REF_CLK_OUTPUT3,  
POWER_MODULE_REF_CLK_OUTPUT4,  
POWER_MODULE_PMP,  
POWER_MODULE_EBI,  
POWER_MODULE_SQI,  
POWER_MODULE_ETHERNET,  
POWER_MODULE_DMA,  
POWER_MODULE_RANDOM_NUM_GENERATOR,  
POWER_MODULE_CRYPT0  
} POWER_MODULE;
```

Description

POWER module enumeration

This enumeration lists the modules whose power can be controlled by PMD registers

Members

Members	Description
POWER_MODULE_ADC1	ADC module
POWER_MODULE_CTMU	Charge Time Measurement Unit module
POWER_MODULE_CVR	Comparator Voltage Reference module
POWER_MODULE_CMP1	Comparator module 1
POWER_MODULE_CMP2	Comparator module 2
POWER_MODULE_CMP3	Comparator module 3
POWER_MODULE_IC1	Input Capture 1 module
POWER_MODULE_IC2	Input Capture 2 module
POWER_MODULE_IC3	Input Capture 3 module
POWER_MODULE_IC4	Input Capture 4 module
POWER_MODULE_IC5	Input Capture 5 module
POWER_MODULE_IC6	Input Capture 6 module
POWER_MODULE_IC7	Input Capture 7 module
POWER_MODULE_IC8	Input Capture 8 module
POWER_MODULE_IC9	Input Capture 9 module
POWER_MODULE_OC1	Output Compare 1 module
POWER_MODULE_OC2	Output Compare 2 module
POWER_MODULE_OC3	Output Compare 3 module
POWER_MODULE_OC4	Output Compare 4 module
POWER_MODULE_OC5	Output Compare 5 module
POWER_MODULE_OC6	Output Compare 6 module
POWER_MODULE_OC7	Output Compare 7 module
POWER_MODULE_OC8	Output Compare 8 module
POWER_MODULE_OC9	Output Compare 9 module
POWER_MODULE_TMR1	Timer1 module
POWER_MODULE_TMR2	Timer2 module
POWER_MODULE_TMR3	Timer3 module
POWER_MODULE_TMR4	Timer4 module
POWER_MODULE_TMR5	Timer5 module
POWER_MODULE_TMR6	Timer6 module
POWER_MODULE_TMR7	Timer7 module
POWER_MODULE_TMR8	Timer8 module
POWER_MODULE_TMR9	Timer9 module
POWER_MODULE_UART1	UART1 module
POWER_MODULE_UART2	UART2 module
POWER_MODULE_UART3	UART3 module
POWER_MODULE_UART4	UART4 module
POWER_MODULE_UART5	UART5 module
POWER_MODULE_UART6	UART6 module

POWER_MODULE_SPI1	SPI1 module
POWER_MODULE_SPI2	SPI2 module
POWER_MODULE_SPI3	SPI3 module
POWER_MODULE_SPI4	SPI4 module
POWER_MODULE_SPI5	SPI5 module
POWER_MODULE_SPI6	SPI6 module
POWER_MODULE_I2C1	I2C1 module
POWER_MODULE_I2C2	I2C2 module
POWER_MODULE_I2C3	I2C3 module
POWER_MODULE_I2C4	I2C4 module
POWER_MODULE_I2C5	I2C5 module
POWER_MODULE_USB	USB module
POWER_MODULE_CAN1	CAN1 module
POWER_MODULE_CAN2	CAN2 module
POWER_MODULE_RTCC	RealTime Clock and Calender module
POWER_MODULE_REF_CLK_OUTPUT	Reference Clock Output module
POWER_MODULE_REF_CLK_OUTPUT1	Reference Clock Output module 1
POWER_MODULE_REF_CLK_OUTPUT2	Reference Clock Output module 2
POWER_MODULE_REF_CLK_OUTPUT3	Reference Clock Output module 3
POWER_MODULE_REF_CLK_OUTPUT4	Reference Clock Output module 4
POWER_MODULE_PMP	Parallel Master Port module
POWER_MODULE_EBI	External Bus Interface module
POWER_MODULE_SQI	Serial Quad Interface module
POWER_MODULE_ETHERNET	Ethernet module
POWER_MODULE_DMA	Data Memory Access module
POWER_MODULE_RANDOM_NUM_GENERATOR	Random Number Generator module
POWER_MODULE_CRYPT0	Cryptographic module

Remarks

Not all modules exist in all devices. Please refer to the specific device data sheet for availability.

5.6.18.6.3.2 POWER_MODULE_ID Enumeration**C**

```
typedef enum {
    POWER_ID_0,
    POWER_NUMBER_OF_MODULES
} POWER_MODULE_ID;
```

Description

Power Module ID

This enumeration identifies the Power modules that are available on the microcontroller. This is the superset of all the possible instances that might be available on a Microchip microcontroller.

Members

Members	Description
POWER_ID_0	POWER Module ID
POWER_NUMBER_OF_MODULES	Number of available POWER modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the "Power-Saving Features" chapter in the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

5.6.18.6.4 Feature Existence Functions

5.6.18.6.4.1 PLIB_POWER_ExistsDeepSleepBOR Function

C

```
bool PLIB_POWER_ExistsDeepSleepBOR(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the DeepSleepBORControl feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_DeepSleepBOREnable](#)
- [PLIB_POWER_DeepSleepBORDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DeepSleepBORControl feature is supported on the device
- false - The DeepSleepBORControl feature is not supported on the device

Remarks

None.

5.6.18.6.4.2 PLIB_POWER_ExistsDeepSleepFaultDetect Function

C

```
bool PLIB_POWER_ExistsDeepSleepFaultDetect(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the DeepSleepFaultDetectStatus feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_DeepSleepFaultDetectStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DeepSleepFaultDetectStatus feature is supported on the device
- false - The DeepSleepFaultDetectStatus feature is not supported on the device

Remarks

None.

5.6.18.6.4.3 PLIB_POWER_ExistsDeepSleepInterruptOnChange Function**C**

```
bool PLIB_POWER_ExistsDeepSleepInterruptOnChange(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the DeepSleepIOCStatus feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_DeepSleepInterruptOnChangeStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DeepSleepIOCStatus feature is supported on the device
- false - The DeepSleepIOCStatus feature is not supported on the device

Remarks

None.

5.6.18.6.4.4 PLIB_POWER_ExistsDeepSleepMCLREvent Function**C**

```
bool PLIB_POWER_ExistsDeepSleepMCLREvent(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the DeepSleepMCLRStatus feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_DeepSleepMCLREventStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DeepSleepMCLRStatus feature is supported on the device
- false - The DeepSleepMCLRStatus feature is not supported on the device

Remarks

None.

5.6.18.6.4.5 PLIB_POWER_ExistsDeepSleepMode Function**C**

```
bool PLIB_POWER_ExistsDeepSleepMode(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the DeepSleepModeControl feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_DeepSleepModeEnable](#)
- [PLIB_POWER_DeepSleepModeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DeepSleepModeControl feature is supported on the device
- false - The DeepSleepModeControl feature is not supported on the device

Remarks

None.

5.6.18.6.4.6 PLIB_POWER_ExistsDeepSleepPowerOnReset Function**C**

```
bool PLIB_POWER_ExistsDeepSleepPowerOnReset(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the DeepSleepPORStatus feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_DeepSleepPowerOnResetStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DeepSleepPORStatus feature is supported on the device
- false - The DeepSleepPORStatus feature is not supported on the device

Remarks

None.

5.6.18.6.4.7 PLIB_POWER_ExistsIdleStatus Function**C**

```
bool PLIB_POWER_ExistsIdleStatus(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the IdleStatus feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_DeviceWasInIdleMode](#)
- [PLIB_POWER_ClearIdleStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The IdleStatus feature is supported on the device
- false - The IdleStatus feature is not supported on the device

Remarks

None.

5.6.18.6.4.8 PLIB_POWER_ExistsPeripheralModuleControl Function**C**

```
bool PLIB_POWER_ExistsPeripheralModuleControl(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the PeripheralModuleControl feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_PeripheralModuleDisable](#)
- [PLIB_POWER_PeripheralModuleEnable](#)
- [PLIB_POWER_PeripheralModuleIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PeripheralModuleControl feature is supported on the device
- false - The PeripheralModuleControl feature is not supported on the device

Remarks

None.

5.6.18.6.4.9 PLIB_POWER_ExistsSleepStatus Function**C**

```
bool PLIB_POWER_ExistsSleepStatus(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the SleepStatus feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_DeviceWasInSleepMode](#)
- [PLIB_POWER_ClearSleepStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SleepStatus feature is supported on the device
- false - The SleepStatus feature is not supported on the device

Remarks

None.

5.6.18.6.4.10 PLIB_POWER_ExistsVoltageRegulatorControl Function**C**

```
bool PLIB_POWER_ExistsVoltageRegulatorControl(  
    POWER_MODULE_ID index  
);
```

Description

This function identifies whether the VoltageRegulatorControl feature is available on the POWER module. When this function returns true, these functions are supported on the device:

- [PLIB_POWER_VoltageRegulatorEnable](#)
- [PLIB_POWER_VoltageRegulatorDisable](#)
- [PLIB_POWER_VoltageRegulatorIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The VoltageRegulatorControl feature is supported on the device
- false - The VoltageRegulatorControl feature is not supported on the device

Remarks

None.

5.6.18.7 Files

Files

Name	Description
plib_power.h	Defines the Power Peripheral Library interface.

Description

5.6.18.7.1 plib_power.h

Power Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Power Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Power Controller module.

Functions

	Name	Description
≡	PLIB_POWER_ClearIdleStatus	Clears the Idle mode status of the device.
≡	PLIB_POWER_ClearSleepStatus	Clear the Sleep mode status bit of the device.
≡	PLIB_POWER_DeepSleepBORDisable	Disables the Deep Sleep BOR.
≡	PLIB_POWER_DeepSleepBOREnable	Enables the Deep Sleep BOR mode.
≡	PLIB_POWER_DeepSleepFaultDetectStatus	Returns the status of Deep Sleep Fault Detect bit.
≡	PLIB_POWER_DeepSleepInterruptOnChangeStatus	Returns the status of the Deep Sleep Interrupt-on-Change bit.
≡	PLIB_POWER_DeepSleepMCLREventStatus	Returns the status of the Deep Sleep MCLR Event bit.
≡	PLIB_POWER_DeepSleepModeDisable	Disables the Deep Sleep mode.

	PLIB_POWER_DeepSleepModeEnable	Enables the Deep Sleep mode.
	PLIB_POWER_DeepSleepPowerOnResetStatus	Returns the status of Deep Sleep Power-on Reset bit.
	PLIB_POWER_DeviceWasInIdleMode	Returns the Idle mode status of the device.
	PLIB_POWER_DeviceWasInSleepMode	Returns the Sleep mode status of the device.
	PLIB_POWER_ExistsDeepSleepBOR	Identifies whether the DeepSleepBORControl feature exists on the POWER module
	PLIB_POWER_ExistsDeepSleepFaultDetect	Identifies whether the DeepSleepFaultDetectStatus feature exists on the POWER module
	PLIB_POWER_ExistsDeepSleepInterruptOnChange	Identifies whether the DeepSleepIOCStatus feature exists on the POWER module
	PLIB_POWER_ExistsDeepSleepMCLREvent	Identifies whether the DeepSleepMCLRStatus feature exists on the POWER module
	PLIB_POWER_ExistsDeepSleepMode	Identifies whether the DeepSleepModeControl feature exists on the POWER module
	PLIB_POWER_ExistsDeepSleepPowerOnReset	Identifies whether the DeepSleepPORStatus feature exists on the POWER module
	PLIB_POWER_ExistsIdleStatus	Identifies whether the IdleStatus feature exists on the POWER module
	PLIB_POWER_ExistsPeripheralModuleControl	Identifies whether the PeripheralModuleControl feature exists on the POWER module
	PLIB_POWER_ExistsSleepStatus	Identifies whether the SleepStatus feature exists on the POWER module
	PLIB_POWER_ExistsVoltageRegulatorControl	Identifies whether the VoltageRegulatorControl feature exists on the POWER module
	PLIB_POWER_PeripheralModuleDisable	Disable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleEnable	Enable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleIsEnabled	Check if selected peripheral is enabled or not
	PLIB_POWER_VoltageRegulatorDisable	Disables the voltage regulator during Sleep mode.
	PLIB_POWER_VoltageRegulatorEnable	Enable the voltage regulator during Sleep mode.
	PLIB_POWER_VoltageRegulatorIsEnabled	Provides the status of the voltage regulator during Sleep mode.

File Name

plib_power.h

Company

Microchip Technology Inc.

5.6.19 Reset Peripheral Library

5.6.19.1 Introduction

Reset Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Reset Controller modules on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

Reset Controller Overview

The Reset Controller is a key component of a microcontroller. The Reset Controller gives the reset status of the device at any point of time.

The following are the key features present on a Reset Controller:

- **Reset Source Configuration:** Provides the ability to enable and disable a source that may reset the device
- **Reset Status:** Identifies the source of a reset and clears it

A microcontroller can support one or more of the above reset modes.

Please refer to the the "Resets" chapter in the specific device data sheet to determine the exact set of supported features.

5.6.19.2 Release Notes

MPLAB Harmony Version: v0.70b **Reset Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.19.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.19.4 Using the Library

This topic describes the basic architecture of the Reset Peripheral Library and provides information and examples on how to use it.

Description

Interface Header File: [plib_reset.h](#)

The interface to the Reset Peripheral Library is defined in the "[plib_reset.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the Reset Peripheral Library must include "peripheral.h".

Peripheral Module IDs

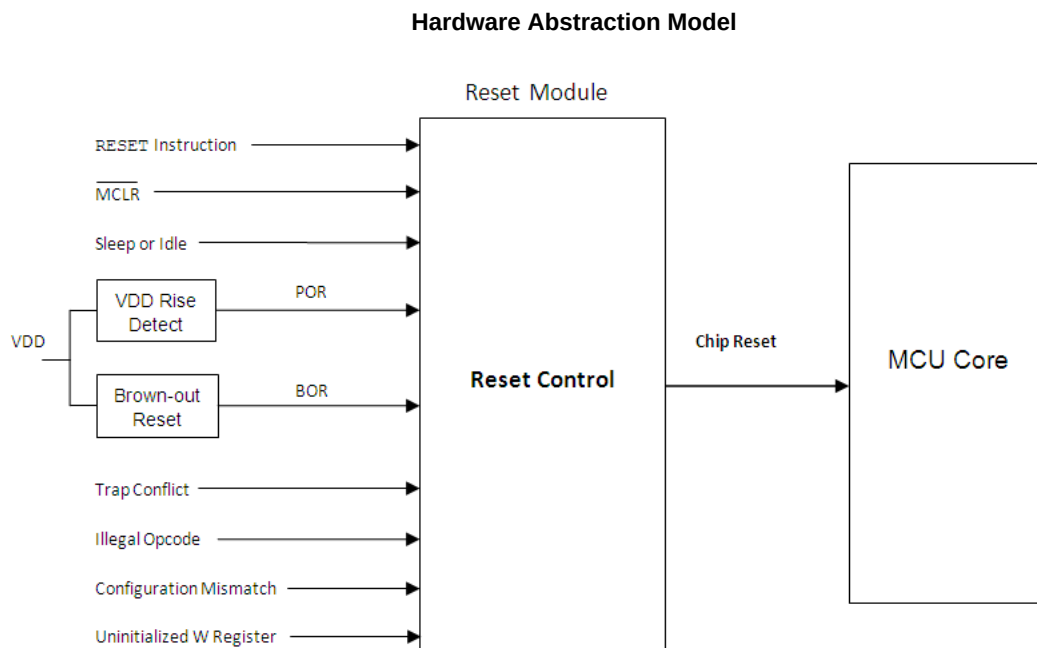
Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. To support this, the first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). Not all microcontrollers will have all instances of the module listed in this enumeration. Please refer to the "**Resets**" chapter in the specific device data sheet for availability.

5.6.19.4.1 Hardware Abstraction Model

This library provides a low-level abstraction of the Reset Controller module on Microchip PIC microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The following figure illustrates the hardware abstraction model for the Reset Peripheral Library.



The **Reset Controller** receives requests from multiple **Reset Sources**. The **Reset Controller** controls the overall operation of the reset controller.

Source Enable Configuration enables or disables a particular source of reset. Some of the reset sources such as Traps are not

controlled by the users, whereas others can be user controlled.

Note: Not all features are available on all devices. Refer to the **"Resets"** chapter in the specific device data sheet for availability

5.6.19.4.2 Library Architecture

This library provides the low-level abstraction of the Reset module available on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

The library interface routines are divided into various subsections, each of the subsection addresses one of the blocks or the overall operation of the Reset Controller module.

Library Interface Section	Description
Initialization Functions	This section provides a set of functions for configuring and enabling a reset.
Status Functions	This section provides a function to read the status of a particular reset.

Note: Not all modes are available on all devices. Refer to the **"Resets"** chapter in the specific device data sheet for availability.

5.6.19.4.3 How the Library Works

Initialization

The steps that are required to initialize the Reset Controller module vary for different microcontrollers. Refer to the **"Resets"** chapter in the specific device data sheet for the correct initialization sequence. The following information provides a general overview.

Description

Reset Source Setup

Some reset sources need to be enabled for it to be able to Reset the device. Use the function [PLIB_RESET_SoftwareResetEnable](#) to enable the reset source.

Reset Status

The status of the reset indicates which source generated the reset. The status of the reset source can be obtained using the function [PLIB_RESET_ReasonGet](#). The reason for the reset can be cleared using the function [PLIB_RESET_ReasonClear](#).

5.6.19.5 Configuring the Library

The library is configured for the supported Reset Controller modules when the processor is chosen in the MPLAB IDE.

5.6.19.6 Library Interface

Data Types and Constants

	Name	Description
	RESET_MODULE_ID	Identifies the supported Reset modules.
	RESET_CONFIG_REG_READ_ERROR	Lists the reset sources.
	RESET_NMI_COUNT_TYPE	Defines NMI counter data type
	RESET_NMI_REASON	Lists the NMI reasons.

Feature Existence Functions

	Name	Description
	PLIB_RESET_ExistsResetReasonStatus	Identifies whether the ResetReasonStatus feature exists on the RESET module
	PLIB_RESET_ExistsSoftwareResetTrigger	Identifies whether the SoftwareResetTrigger feature exists on the RESET module
	PLIB_RESET_ExistsConfigRegReadError	Identifies whether the ConfigRegReadError feature exists on the RESET module
	PLIB_RESET_ExistsNmiControl	Identifies whether the NmiControl feature exists on the RESET module
	PLIB_RESET_ExistsNmiCounter	Identifies whether the NmiCounter feature exists on the RESET module
	PLIB_RESET_ExistsWdtInSleep	Identifies whether the WdtInSleep feature exists on the RESET module

Initialization Functions

	Name	Description
	PLIB_RESET_SoftwareResetEnable	Enables the software reset.

Status Functions

	Name	Description
	PLIB_RESET_ReasonClear	Clears the RCON register.
	PLIB_RESET_ReasonGet	Returns the reason for a reset.
	PLIB_RESET_ConfigRegReadErrorGet	Get the Configuration Register Read Error
	PLIB_RESET_WdtTimeOutHasOccurredInSleep	Returns the state of the device when WDT time-out occurred

Description

This section lists and describes the functions, data types, and constants provided in the Reset Peripheral Library.

5.6.19.6.1 Initialization Functions**5.6.19.6.1.1 PLIB_RESET_SoftwareResetEnable Function****C**

```
void PLIB_RESET_SoftwareResetEnable(
    RESET_MODULE_ID index
);
```

Description

This function enables the software reset. The system unlock sequence, which must be performed to enable this feature, is done internally in this function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the SoftwareResetTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsSoftwareResetTrigger](#) function in your application to determine whether this feature is available.

Example

```
PLIB_RESET_SoftwareResetEnable( RESET_ID_0 );
```

5.6.19.6.2 Status Functions

5.6.19.6.2.1 PLIB_RESET_ReasonClear Function

C

```
void PLIB_RESET_ReasonClear(
    RESET_MODULE_ID index,
    RESET_REASON reason
);
```

Description

This function clears the particular reset bit in the RCON register. Multiple reasons for a reset can be ORed together and given as an input parameter to clear them simultaneously.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
reason	Select the reset type from enum RESET_REASON

Returns

None

Remarks

This function implements an operation of the ResetReasonStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsResetReasonStatus](#) function in your application to determine whether this feature is available.

Example

```
PLIB_RESET_ReasonClear( RESET_ID_0, RESET_REASON_MCLR | RESET_REASON_POWERON );
```

5.6.19.6.2.2 PLIB_RESET_ReasonGet Function

C

```
RESET_REASON PLIB_RESET_ReasonGet(
    RESET_MODULE_ID index
);
```

Description

This function returns the reason a reset has occurred for the selected device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

RESET_REASON - Type of reset (when there is more than one reason for a reset, this function will logically OR (bitwise) the reasons and return the value.

Remarks

This function implements an operation of the ResetReasonStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsResetReasonStatus](#) function in your application to determine whether this feature is available.

Example

```
RESET_REASON type;  
  
type = PLIB_RESET_ReasonGet ( RESET_ID_0 );
```

5.6.19.6.2.3 PLIB_RESET_ConfigRegReadErrorGet Function**C**

```
RESET_CONFIG_REG_READ_ERROR PLIB_RESET_ConfigRegReadErrorGet(  
    RESET_MODULE_ID index  
);
```

Description

This function gets the Configuration register read error, if any, after the reset

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

[RESET_CONFIG_REG_READ_ERROR](#) - Type of Config Register Read Error

Remarks

This function implements an operation of the ConfigRegReadError feature. This feature may not be available on all devices. Refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsConfigRegReadError](#) function in your application to determine whether this feature is available.

Example

```
if (PLIB_RESET_ConfigRegReadErrorGet( RESET_ID_0 )== PRIMARY_CONFIG_REG_READ_ERROR )  
{  
    //Do Something  
}
```

5.6.19.6.2.4 PLIB_RESET_WdtTimeOutHasOccurredInSleep Function**C**

```
bool PLIB_RESET_WdtTimeOutHasOccurredInSleep(  
    RESET_MODULE_ID index  
);
```

Description

This function returns if device was in sleep or not when WDT time-out event occurred.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

True - device was in sleep mode when WDT time-out occurred
False - device was not in sleep mode when WDT time-out occurred

Remarks

This function implements an operation of the WdtInSleep feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsWdtInSleep](#) function in your application to determine whether this feature is available.

Example

```
if (PLIB_RESET_WdtTimeOutHasOccurredInSleep( RESET_ID_0 ))
{
    //Do Something
}
```

5.6.19.6.3 Data Types and Constants

5.6.19.6.3.1 RESET_MODULE_ID Enumeration

C

```
typedef enum {
    RESET_ID_0
} RESET_MODULE_ID;
```

Description

Reset Module ID

This enumeration identifies the Reset modules that are available on the microcontroller. This is the superset of all the possible instances that might be available on a Microchip microcontroller.

Members

Members	Description
RESET_ID_0	RESET Module ID 0

Remarks

Caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules are available on all devices. Refer to the "Resets" chapter in the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

5.6.19.6.3.2 RESET_CONFIG_REG_READ_ERROR Enumeration

C

```
typedef enum {
    RESET_REASON_POWERON,
    RESET_REASON_BROWNOUT,
    RESET_REASON_WDT_TIMEOUT,
    RESET_REASON_SOFTWARE,
    RESET_REASON_MCLR,
    RESET_REASON_CONFIG_MISMATCH,
    RESET_REASON_HIGH_VOLTAGE_DETECT,
    PRIMARY_CONFIG_REG_READ_ERROR,
    PRIMARY_AND_ALTERNATIVE_CONFIG_REG_READ_ERROR,
    NO_CONFIG_REG_READ_ERROR
} RESET_CONFIG_REG_READ_ERROR;
```

Description

RESET source select enumeration

This enumeration lists the possible reset sources.

Members

Members	Description
RESET_REASON_POWERON	The last reset was a power on reset
RESET_REASON_BROWNOUT	The last reset was a brown out reset
RESET_REASON_WDT_TIMEOUT	The last reset was a watch dog timer time-out reset
RESET_REASON_SOFTWARE	The last reset was a software reset
RESET_REASON_MCLR	The last reset was a master clear(MCLR) reset
RESET_REASON_CONFIG_MISMATCH	The last reset was a configuration mismatch reset
RESET_REASON_HIGH_VOLTAGE_DETECT	The last reset was high voltage detect reset
PRIMARY_CONFIG_REG_READ_ERROR	An error occurred during the read of primary configuration registers
PRIMARY_AND_ALTERNATIVE_CONFIG_REG_READ_ERROR	An error occurred during the read of primary and alternate configuration register
NO_CONFIG_REG_READ_ERROR	No error occurred during the read of configuration registers

Remarks

Not all reset sources exist on all devices. Please refer to the specific device data sheet for availability.

5.6.19.6.3.3 RESET_NMI_COUNT_TYPE Type

C

```
typedef unsigned char RESET_NMI_COUNT_TYPE;
```

Description

RESET_NMI_COUNT_TYPE data type

It defines NMI counter data type

5.6.19.6.3.4 RESET_NMI_REASON Enumeration

C

```
typedef enum {
    WDTO_NMI,
```

```
DMTO_NMI,  
SOFTWARE_NMI,  
CF_NMI,  
WDTS_NMI  
} RESET_NMI_REASON;
```

Description

NMI Reason enumeration

This enumeration lists the possible NMI sources.

Members

Members	Description
WDTO_NMI	Watch Dog Timer time-out has caused NMI
DMTO_NMI	Deadman Timer time-out has caused NMI
SOFTWARE_NMI	Software trigger has caused NMI
CF_NMI	Clock Failure has caused NMI
WDTS_NMI	Watch Dog Timer in Sleep has caused NMI

Remarks

Not all NMI sources exist on all devices. Please refer to the specific device data sheet for availability.

5.6.19.6.4 Feature Existence Functions

5.6.19.6.4.1 PLIB_RESET_ExistsResetReasonStatus Function

C

```
bool PLIB_RESET_ExistsResetReasonStatus(  
    RESET_MODULE_ID index  
);
```

Description

This function identifies whether the ResetReasonStatus feature is available on the RESET module. When this function returns true, these functions are supported on the device:

- [PLIB_RESET_ReasonGet](#)
- [PLIB_RESET_ReasonClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ResetReasonStatus feature is supported on the device
- false - The ResetReasonStatus feature is not supported on the device

Remarks

None.

5.6.19.6.4.2 PLIB_RESET_ExistsSoftwareResetTrigger Function

C

```
bool PLIB_RESET_ExistsSoftwareResetTrigger(  
    RESET_MODULE_ID index  
);
```

Description

This function identifies whether the SoftwareResetTrigger feature is available on the RESET module. When this function returns true, these functions are supported on the device:

- [PLIB_RESET_SoftwareResetEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SoftwareResetTrigger feature is supported on the device
- false - The SoftwareResetTrigger feature is not supported on the device

Remarks

None.

5.6.19.6.4.3 PLIB_RESET_ExistsConfigRegReadError Function

C

```
bool PLIB_RESET_ExistsConfigRegReadError(  
    RESET_MODULE_ID index  
);
```

Description

This function identifies whether the ConfigRegReadError feature is available on the RESET module. When this function returns true, these functions are supported on the device:

- [PLIB_RESET_ConfigRegReadErrorGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ConfigRegReadError feature is supported on the device
- false - The ConfigRegReadError feature is not supported on the device

Remarks

None.

5.6.19.6.4.4 PLIB_RESET_ExistsNmiControl Function

C

```
bool PLIB_RESET_ExistsNmiControl(  
    RESET_MODULE_ID index  
);
```

Description

This function identifies whether the NmiControl feature is available on the RESET module. When this function returns true, these functions are supported on the device:

- [PLIB_RESET_NmiReasonGet](#)
- [PLIB_RESET_NmiEventTrigger](#)
- [PLIB_RESET_NmiEventClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The NmiControl feature is supported on the device
- false - The NmiControl feature is not supported on the device

Remarks

None.

5.6.19.6.4.5 PLIB_RESET_ExistsNmiCounter Function

C

```
bool PLIB_RESET_ExistsNmiCounter(  
    RESET_MODULE_ID index  
);
```

Description

This function identifies whether the NmiCounter feature is available on the RESET module. When this function returns true, these functions are supported on the device:

- [PLIB_RESET_NmiCounterValueSet](#)
- [PLIB_RESET_NmiCounterValueGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The NmiCounter feature is supported on the device
- false - The NmiCounter feature is not supported on the device

Remarks

None.

5.6.19.6.4.6 PLIB_RESET_ExistsWdtInSleep Function**C**

```
bool PLIB_RESET_ExistsWdtInSleep(
    RESET_MODULE_ID index
);
```

Description

This function identifies whether the WdtInSleep feature is available on the RESET module. When this function returns true, these functions are supported on the device:

- [PLIB_RESET_WdtTimeOutHasOccurredInSleep](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WdtInSleep feature is supported on the device
- false - The WdtInSleep feature is not supported on the device

Remarks

None.

5.6.19.6.5 NMI Events Functions**Functions**

	Name	Description
	PLIB_RESET_NmiCounterValueGet	Gets the NMI counter value
	PLIB_RESET_NmiCounterValueSet	Sets the NMI counter
	PLIB_RESET_NmiEventClear	Clears the NMI event
	PLIB_RESET_NmiEventTrigger	Triggers the NMI event
	PLIB_RESET_NmiReasonGet	Returns the reason for NMI

Description**5.6.19.6.5.1 PLIB_RESET_NmiCounterValueGet Function****C**

```
RESET_NMI_COUNT_TYPE PLIB_RESET_NmiCounterValueGet(
    RESET_MODULE_ID index
);
```

Description

This function returns the NMI Reset counter value

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

nmi_count - NMI counter value

Remarks

This function implements an operation of the NmiCounter feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsNmiCounter](#) function in your application to determine whether this feature is available.

Example

```
RESET_NMI_COUNT_TYPE nmi_count;  
nmi_count = PLIB_RESET_NmiCounterValueGet( RESET_ID_0);
```

5.6.19.6.5.2 PLIB_RESET_NmiCounterValueSet Function**C**

```
void PLIB_RESET_NmiCounterValueSet(  
    RESET_MODULE_ID index,  
    RESET_NMI_COUNT_TYPE nmi_count  
);
```

Description

This function sets the NMI counter which needs to be expired for WDT or DMT reset

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured nmi_count -NMI counter value

Returns

None

Remarks

This function implements an operation of the NmiCounter feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsNmiCounter](#) function in your application to determine whether this feature is available.

Example

```
PLIB_RESET_NmiCounterValueSet( RESET_ID_0, 0x54);
```

5.6.19.6.5.3 PLIB_RESET_NmiEventClear Function

C

```
void PLIB_RESET_NmiEventClear(  
    RESET_MODULE_ID index,  
    RESET_NMI_REASON nmi_reason  
);
```

Description

This function clears the Non-Maskable-Interrupt (NMI) event. if WDTO or DMT0 NMI event is cleared before NMI counter reaches zero, then no device reset is asserted.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nmi_reason	sets of reasons which can cause NMI

Returns

None

Remarks

This function implements an operation of the NmiControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsNmiControl](#) function in your application to determine whether this feature is available.

Example

```
PLIB_RESET_NmiEventClear( RESET_ID_0, WDTO_NMI );
```

5.6.19.6.5.4 PLIB_RESET_NmiEventTrigger Function

C

```
void PLIB_RESET_NmiEventTrigger(  
    RESET_MODULE_ID index,  
    RESET_NMI_REASON nmi_reason  
);
```

Description

This function triggers the Non-Maskable-Interrupt (NMI). In case of DMT or WDT NMI event, it will also trigger NMI counter to start the count down.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nmi_reason	sets of reasons which can cause NMI

Returns

None

Remarks

This function implements an operation of the NmiControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsNmiControl](#) function in your application to determine whether this feature is available.

Example

```
PLIB_RESET_NmiEventTrigger( RESET_ID_0, WDTO_NMI );
```

5.6.19.6.5.5 PLIB_RESET_NmiReasonGet Function**C**

```
RESET_NMI_REASON PLIB_RESET_NmiReasonGet(  
    RESET_MODULE_ID index  
);
```

Description

This function returns the reason which caused the Non-Maskable-Interrupt (NMI)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

[RESET_NMI_REASON](#) - sets of reasons which can cause NMI

Remarks

This function implements an operation of the NmiControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the [PLIB_RESET_ExistsNmiControl](#) function in your application to determine whether this feature is available.

Example

```
if (PLIB_RESET_NmiReasonGet( RESET_ID_0 )== WDTO_NMI )  
{  
    //Do Something  
}
```

5.6.19.7 Files**Files**

Name	Description
plib_reset.h	Defines the Reset Peripheral Library interface.

Description**5.6.19.7.1 plib_reset.h**

Reset Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the

Reset Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Reset Controller module.

Functions

	Name	Description
≡	PLIB_RESET_ConfigRegReadErrorGet	Get the Configuration Register Read Error
≡	PLIB_RESET_ExistsConfigRegReadError	Identifies whether the ConfigRegReadError feature exists on the RESET module
≡	PLIB_RESET_ExistsNmiControl	Identifies whether the NmiControl feature exists on the RESET module
≡	PLIB_RESET_ExistsNmiCounter	Identifies whether the NmiCounter feature exists on the RESET module
≡	PLIB_RESET_ExistsResetReasonStatus	Identifies whether the ResetReasonStatus feature exists on the RESET module
≡	PLIB_RESET_ExistsSoftwareResetTrigger	Identifies whether the SoftwareResetTrigger feature exists on the RESET module
≡	PLIB_RESET_ExistsWdtInSleep	Identifies whether the WdtInSleep feature exists on the RESET module
≡	PLIB_RESET_NmiCounterValueGet	Gets the NMI counter value
≡	PLIB_RESET_NmiCounterValueSet	Sets the NMI counter
≡	PLIB_RESET_NmiEventClear	Clears the NMI event
≡	PLIB_RESET_NmiEventTrigger	Triggers the NMI event
≡	PLIB_RESET_NmiReasonGet	Returns the reason for NMI
≡	PLIB_RESET_ReasonClear	Clears the RCON register.
≡	PLIB_RESET_ReasonGet	Returns the reason for a reset.
≡	PLIB_RESET_SoftwareResetEnable	Enables the software reset.
≡	PLIB_RESET_WdtTimeOutHasOccurredInSleep	Returns the state of the device when WDT time-out occurred

Types

	Name	Description
◆	RESET_NMI_COUNT_TYPE	Defines NMI counter data type

File Name

plib_reset.h

Company

Microchip Technology Inc.

5.6.20 RTCC Peripheral Library

5.6.20.1 Introduction

Real-Time Clock and Calendar (RTCC) Peripheral Library

for

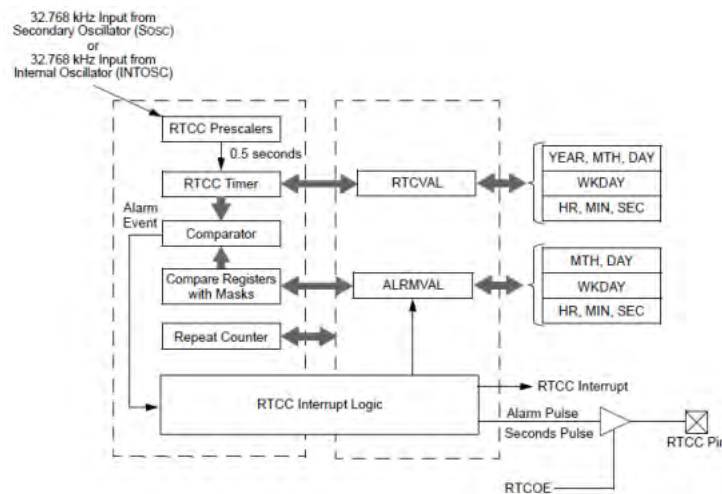
Microchip Microcontrollers

This module provides real-time clock and calendar functions. The RTCC is intended for applications where accurate time must be maintained for extended periods with minimum to no intervention from the CPU. The module is optimized for low-power usage to provide extended battery life while keeping track of time.

Description

The RTCC module is a 100-year clock and calendar with automatic leap year detection. The range of the clock is from 00:00:00 (midnight) on January 1, 2000 to 23:59:59 on December 31, 2099. The hours are available in 24-hour (military time) format. The clock provides a granularity of one second with half-second visibility to the user.

RTCC Block Diagram



Operating Modes

RTCC Mode

The RTCC module is intended to be clocked by an external real-time clock crystal oscillating at 32.768 kHz. The prescaler divides the crystal oscillator frequency to produce a 1 Hz frequency for the clock and calendar. The current Date and Time are stored in a BCD counter. Please refer to the "RTCC" chapter in the specific device data sheet and the related family reference manual section to learn more about the counter rolls, overflows and carryovers.

Alarm Mode

The RTCC module provides an alarm function that is configurable from a half-second to one year. After the alarm is enabled, the RTCC module can be configured to repeat the alarm at preconfigured intervals. The indefinite repetition of the alarm is provided through the Chime feature.

Note: Not all functionality is available on all devices. Please refer the specific device data sheet to determine availability.

5.6.20.2 Release Notes

MPLAB Harmony Version: v0.70b **RTCC Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.20.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.20.4 Using the Library

This topic describes the basic architecture of the RTCC Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_rtcc.h](#)

The interface to the RTCC Peripheral Library is defined in the "[plib_rtcc.h](#)" header file. Any C language source (.c) file that uses the RTCC Peripheral Library should include "[plib_rtcc.h](#)".

Library File: The RTCC Peripheral Library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for information on how the peripheral interacts with the framework.

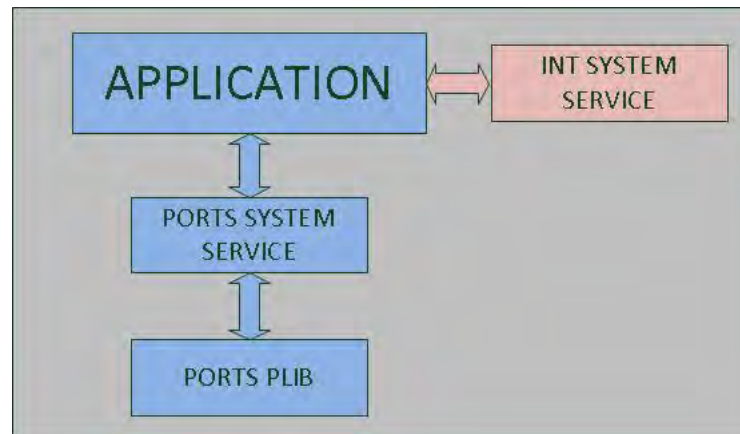
5.6.20.4.1 Hardware Abstraction Model

This library provides a low-level abstraction of the RTCC module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The RTCC peripheral library provides interface routines to interact with the blocks shown in the following diagram.

Hardware Abstraction Model



The **RTCC Control Logic** provides the capability to operate the RTCC module in the RTCC and Alarm modes. The control logic also handles the comparison and the counter increments, which in turn control the alarm generation. This module is also responsible for the generation of a square wave at the RTCC output pin. In addition, the RTCC module provides the RTCC drift calibration.

The **RTCC Interrupt Logic** is primarily used in alarm generation. The various configurations of the alarm and their repetition period may be defined.

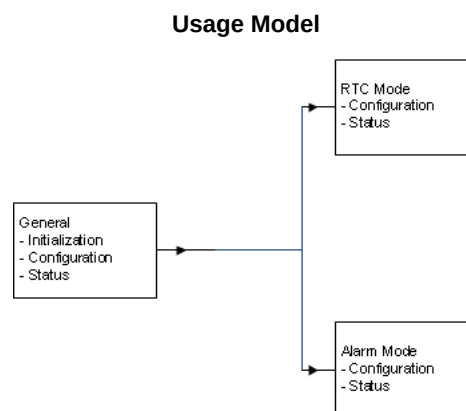
The **RTCC Registers** store the actual date and time. Separate registers are present for RTCC and alarm functionality.

5.6.20.4.2 Library Usage Model

This section provides an overview of the usage model for the RTCC Peripheral Library.

Description

The following diagram describes the major components of the usage model.



Each of the blocks in the diagram correspond to the library interface section.

Library Interface Section	Description
General Functions	Provides the setup and configuration of general functionalities in the RTCC module. This section also covers the drift calibration.
RTCC Mode Functions	Provides configuration of the RTCC-related registers. Updating the Date and Time registers along with reading them accurately is performed.

Alarm Mode Functions	Provides configuration of the Alarm-related registers. Updating the Date and Time registers along with reading them accurately is performed. In addition, the rate at which an alarm occurs and the frequency is also configured in this mode.

5.6.20.4.3 How the Library Works

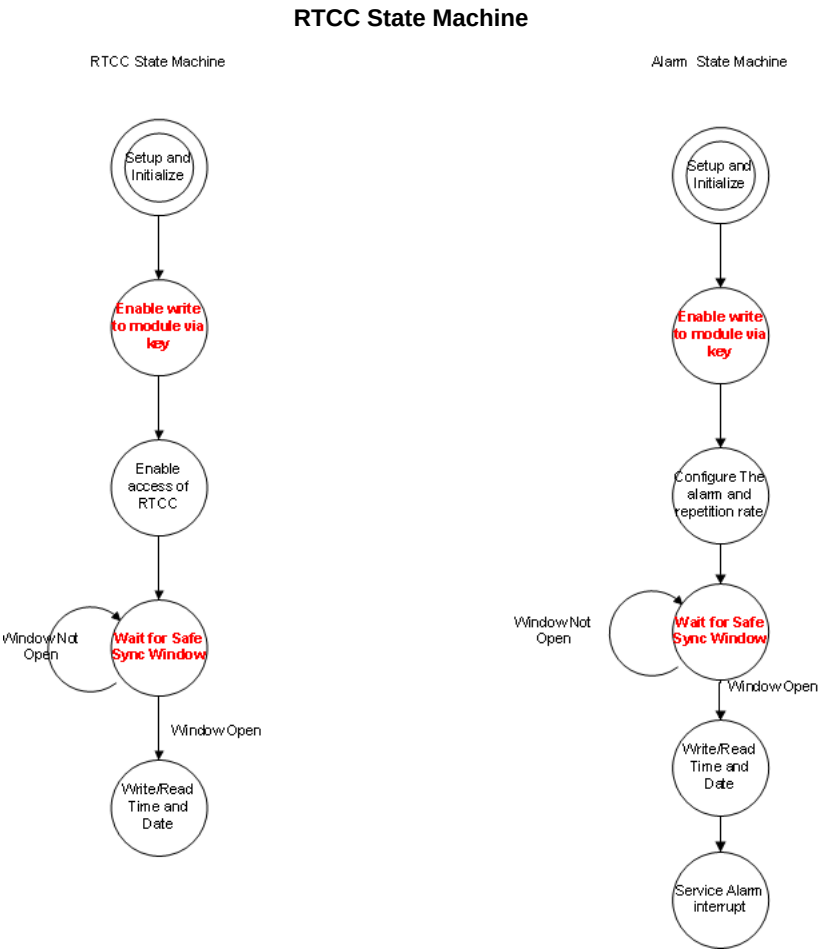
The usage model for this library is explained in the following sections.

5.6.20.4.3.1 State Machine

This RTCC state machine is provided to give a general idea of the usage model of the peripheral library. Refer to the usage model for more detailed steps for the scenario that is being used.

Description

The following state machine diagram is for RTCC mode and Alarm Mode during normal operation.



State	Associated Function
Setup and Initialization	Refer to mode of RTCC for detailed instructions of setup.
Enable Write via Key Sequence	For RTCC to perform any function, the module has to be enabled via a series of commands in a special sequence. This is abstracted and is achieved using PLIB_RTCC_WriteEnable .
Enable Access of RTC	These include the functions needed to set up the RTCC/Alarm, which will be explained in detail in the following sections.
Wait for Safe Sync Window	To update the RTCC on-the-fly, a safe window period is available, corresponding to 32 clock cycles called RTSYNC. Further sections will explain the functionality in greater detail.
Write/Read Date and Time	The application may read or write the date and time using PLIB_RTCC_WriteRTCCDateValueRegister , PLIB_RTCC_ReadRTCCDateValueRegister , among other. These functions are device-specific and may not exist for all devices.

5.6.20.4.3.2 RTCC Mode Operations

This section describes the RTCC mode of operation of the RTCC module.

Note: Please refer to the "RTCC" chapter in the specific device data sheet or the related family reference manual to determine which features are supported by your device.

Description

Enabling the RTCC Module for Write Operations

To perform a write to any of the RTCC timer registers the RTCWREN bit must be set. It is recommended that this bit be reset unless a write operation is specifically required.

A sequence of commands, each varying across processor families must be executed to set the RTCWREN.

Example: Enabling a Write

```
// Assume Interrupts are disabled.
// Assume the DMA controller is suspended

PLIB_RTCC_WriteEnable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

// This API performs the writing of the sequence
// 0xaa996655 and 0x556699aa to the SYSKET in MCU32
// and 0x55 and 0xaa in case of MCU 16 and MCU 8
// This API then sets the RTCWREN.
```

Updating RTCC Time and Date

1. Turn off the RTCC using the API [PLIB_RTCC_Disable](#).
2. Wait for the sync clock to turn off by reading the status of [PLIB_RTCC_ValueRegistersReadSyncBit](#) (This operation may not be required on all devices).
3. Update the Date and Time registers using [PLIB_RTCC_WriteRTCTimeValueRegister](#) and [PLIB_RTCC_WriteRTCCDateValueRegister](#).
4. Turn on the RTCC module using [PLIB_RTCC_Enable](#).

Example: Updating RTCC Time and Date

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006
```

```

    PLIB_RTCC_Disable(MY_RTCC_INSTANCE);

    // where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.
    while(PLIB_RTCC_ValueRegistersReadSyncBit(MY_RTCC_INSTANCE)); // Wait for clock to turn
off
    PLIB_RTCC_WriteRTCTimeValueRegister(MY_RTCC_INSTANCE, time); // Update the Time
    PLIB_RTCC_WriteRTCCDateValueRegister(MY_RTCC_INSTANCE, date); // Update the Date
    PLIB_RTCC_Enable(MY_RTCC_INSTANCE);

```

Updating RTCC Time and Date using RTSYNC Window

1. Wait for the sync clock to turn off by reading the status of `PLIB_RTCC_ValueRegistersReadSyncBit`.
2. Update the RTC Date and Time registers using `PLIB_RTCC_WriteRTCCDateValueRegister` and `PLIB_RTCC_WriteRTCTimeValueRegister`.

Example: Updating RTCC Time and Date Using RTSYNC Window

```

    // Assume Write Enable has been performed correctly
    unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
    unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

    // where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.
    //Disable Critical Interrupts here ....
    while(PLIB_RTCC_ValueRegistersReadSyncBit(MY_RTCC_INSTANCE)); // Wait for clock to turn
off
    PLIB_RTCC_WriteRTCTimeValueRegister(MY_RTCC_INSTANCE, time); // Update the Time
    PLIB_RTCC_WriteRTCCDateValueRegister(MY_RTCC_INSTANCE, date); // Update the Date
    //Enable Critical Interrupts here ....

```

Updating RTCC Time and Date Using Register Pointers

1. Turn off the RTCC using `PLIB_RTCC_Disable`.
2. Update the register pointer to point to the accurate registers using `PLIB_RTCC_ValueRegisterPointer`.
3. Update the RTC Date and Time registers using `PLIB_RTCC_ValueWrite`.
4. Turn on the RTCC module using `PLIB_RTCC_Enable`.

Example: Updating RTCC Time and Date Using Register Pointers

```

    // Assume Write Enable has been performed correctly
    PLIB_RTCC_Disable(MY_RTCC_INSTANCE);

    // where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.
    PLIB_RTCC_ValueRegisterPointer(MY_RTCC_INSTANCE, RTCC_REG_YEAR);
    PLIB_RTCC_ValueWrite(MY_RTCC_INSTANCE, 0x0007); // Update the Year
    PLIB_RTCC_ValueWrite(MY_RTCC_INSTANCE, 0x1028); // Update the Month and Day
    PLIB_RTCC_ValueWrite(MY_RTCC_INSTANCE, 0x0110); // Update the Weekday and Hour
    PLIB_RTCC_ValueWrite(MY_RTCC_INSTANCE, 0x0000); // Set the minute and second

```

```
PLIB_RTCC_Enable(MY_RTCC_INSTANCE);
```

5.6.20.4.3.3 Alarm Mode Operations

This section describes the Alarm mode of operation of the RTCC module.

Note: Please refer to the "RTCC" chapter in the specific device data sheet or the related family reference manual to determine which features are supported by your device.

Description

Configuring the One-Time-Per-Day Alarm

1. Turn off the Alarm and also the chime, alarm repeats, and masks using [PLIB_RTCC_AlarmDisable](#).
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_AlarmValueRegistersReadSyncBit (This operation may not be required on all devices).
3. Update the Alarm Date and Time registers using PLIB_RTCC_WriteAlarmTimeValueRegister and PLIB_RTCC_WriteAlarmDateValueRegister.
4. Set the alarm mask to every half-second and the repeat counter to zero using [PLIB_RTCC_AlarmMaskModeSelect](#) and [PLIB_RTCC_AlarmRepeatCountSet](#), respectively.
5. Turn on the RTCC Alarm module using [PLIB_RTCC_AlarmEnable](#).

Example: Configuring the One-Time-Per-Day Alarm

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

PLIB_RTCC_AlarmDisable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

while(PLIB_RTCC_AlarmValueRegistersReadSyncBit(MY_RTCC_INSTANCE)); // Wait for clock to
turn off

PLIB_RTCC_WriteAlarmTimeValueRegister(MY_RTCC_INSTANCE, time); // Update the Time
PLIB_RTCC_WriteAlarmDateValueRegister(MY_RTCC_INSTANCE, date); // Update the Date
PLIB_RTCC_AlarmMaskModeSelect(MY_RTCC_INSTANCE, RTCC_ALARM_EVERY_HALF_SECOND);
PLIB_RTCC_AlarmRepeatCountSet(MY_RTCC_INSTANCE, 0 /*No of repetitions*/ );
PLIB_RTCC_AlarmEnable(MY_RTCC_INSTANCE);
```

Configuring the Repeat Alarm

1. Turn off the Alarm and also the Chime, alarm repeats, and masks using [PLIB_RTCC_AlarmDisable](#).
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_AlarmValueRegistersReadSyncBit (This operation may not be required on all devices).
3. Update the Alarm Date and Time registers using PLIB_RTCC_WriteAlarmTimeValueRegister and PLIB_RTCC_WriteAlarmDateValueRegister.
4. Set the alarm mask to every half-second and the repeat counter to 10 using [PLIB_RTCC_AlarmMaskModeSelect](#) and [PLIB_RTCC_AlarmRepeatCountSet](#), respectively.
5. Turn on the RTCC Alarm module using [PLIB_RTCC_AlarmEnable](#).

Example: Configuring the Repeat Alarm

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006
```

```

    PLIB_RTCC_AlarmDisable(MY_RTCC_INSTANCE);

    // where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

    while(PLIB_RTCC_AlarmValueRegistersReadSyncBit(MY_RTCC_INSTANCE)); // Wait for clock to
    turn off

    PLIB_RTCC_WriteAlarmTimeValueRegister(MY_RTCC_INSTANCE, time); // Update the Time
    PLIB_RTCC_WriteAlarmDateValueRegister(MY_RTCC_INSTANCE, date); // Update the Date
    PLIB_RTCC_AlarmMaskModeSelect(MY_RTCC_INSTANCE, RTCC_ALARM_EVERY_HALF_SECOND);
    PLIB_RTCC_AlarmRepeatCountSet(MY_RTCC_INSTANCE, 10 /*No of repetitions*/ );
    PLIB_RTCC_AlarmEnable(MY_RTCC_INSTANCE);

```

Configuring the Indefinite One-Per-Day Alarm

1. Turn off the Alarm and also the Chime, alarm repeats and masks using [PLIB_RTCC_AlarmDisable](#).
2. Wait for the sync clock to turn off by reading the status of `PLIB_RTCC_AlarmValueRegistersReadSyncBit` (This operation may not be required on all devices).
3. Update the Alarm Date and Time registers using `PLIB_RTCC_WriteAlarmTimeValueRegister` and `PLIB_RTCC_WriteAlarmDateValueRegister`.
4. Set the alarm mask to every half-second and the repeat counter to 0 using [PLIB_RTCC_AlarmMaskModeSelect](#) and [PLIB_RTCC_AlarmRepeatCountSet](#), respectively.
5. Turn on the RTCC Alarm module using [PLIB_RTCC_AlarmEnable](#) and enable chime using [PLIB_RTCC_AlarmChimeEnable](#).

Example: Configuring the Indefinite One-Per-Day Alarm

```

    // Assume Write Enable has been performed correctly
    unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
    unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

    PLIB_RTCC_AlarmDisable(MY_RTCC_INSTANCE);

    // where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

    while(PLIB_RTCC_AlarmValueRegistersReadSyncBit(MY_RTCC_INSTANCE)); // Wait for clock to
    turn off

    PLIB_RTCC_WriteAlarmTimeValueRegister(MY_RTCC_INSTANCE, time); // Update the Time
    PLIB_RTCC_WriteAlarmDateValueRegister(MY_RTCC_INSTANCE, date); // Update the Date
    PLIB_RTCC_AlarmMaskModeSelect(MY_RTCC_INSTANCE, RTCC_ALARM_EVERY_HALF_SECOND);
    PLIB_RTCC_AlarmRepeatCountSet(MY_RTCC_INSTANCE, 0 /*No of repetitions*/ );
    PLIB_RTCC_AlarmEnable(MY_RTCC_INSTANCE);
    PLIB_RTCC_AlarmChimeEnable(MY_RTCC_INSTANCE);

```

5.6.20.4.3.4 Other Functionality

This section describes the Drift Calibration mode of operation of the RTCC module.

Note: Please refer to the "RTCC" chapter in the specific device data sheet or the related family reference manual to determine which features are supported by your device.

Description

The real-time crystal input can be calibrated using the periodic auto-adjust feature. When properly calibrated, the RTCC module

can provide an error of less than 0.66 seconds per month. Calibration has the ability to eliminate an error of up to 260 ppm.

The calibration is accomplished by finding the number of error clock pulses and writing this value into the calibration bit fields using `PLIB_RTCC_DriftCalibrate`.

Drift Calibration

1. Using another timer resource, the user must find the error of the 32.768 kHz crystal.
2. Once the error is known, it must be converted to the number of error clock pulses per minute, as shown by ***(Ideal Frequency(32,758) - Measured Frequency) * 60 = Error Clocks Per Minute***.
3. If the error is negative, the input to `PLIB_RTCC_DriftCalibrate` is negative. Likewise, if the error is positive, the input is positive. The negative and positive values are the actual number of clock pulses to be added or subtracted from the timer counter per minute.

Example: Configuring the One-Time-Per-Day Alarm

```
// Assume Write Enable has been performed correctly
// Assume steps 1 and 2 are performed and the error is determined.

int driftValue = 0x3FD    // 10 bits Adjustment, -3 in value
int T0,T1;

do
{
    T0 = PLIB_RTCC_ReadRTCTimeValueRegister(MY_RTCC_INSTANCE);
    T1 = PLIB_RTCC_ReadRTCTimeValueRegister(MY_RTCC_INSTANCE);

    // where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.
}while(T0!=T1)    //Read Valid Time Value

if((T0 & 0xFF) == 0) // we are at seconds exactly 00, wait for auto adjust
{
    while(!PLIB_RTCC_HalfSecondStatusBit(MY_RTCC_INSTANCE));; Wait for the second Half
}

PLIB_RTCC_DriftCalibrate(MY_RTCC_INSTANCE, 0x00); //Clear the calibration
PLIB_RTCC_DriftCalibrate(MY_RTCC_INSTANCE, 0xdriftValue);
```

5.6.20.4.4 Configuring the Library

The library is configured for the supported RTCC modules when the processor is chosen in the MPLAB IDE.

5.6.20.5 Library Interface

Data Types and Constants

	Name	Description
	<u>RTCC_ALARM_MASK_CONFIGURATION</u>	Data type defining the different configurations for the alarm mask bits.
	<u>RTCC_MODULE_ID</u>	Enumeration: <code>RTCC_MODULE_ID</code> This enumeration defines number of modules which are available on the microcontroller. This is the superset of all of the possible instances that may be available on the Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	<u>RTCC_VALUE_REGISTER_POINTER</u>	Data type defining the different configurations by which the RTCC Date and Time Registers can be accessed.

Alarm Mode Functions

	Name	Description
≡	PLIB_RTCC_AlarmChimeDisable	Disables the specific RTCC module's chime.
≡	PLIB_RTCC_AlarmChimeEnable	Enables the specific RTCC module's chime.
≡	PLIB_RTCC_AlarmDateGet	Returns the contents of the specific RTCC module's Alarm Date register.
≡	PLIB_RTCC_AlarmDateSet	Writes to the specific RTCC module's Alarm Date register.
≡	PLIB_RTCC_AlarmDayGet	Returns the contents of the Day bits in the specific RTCC module's Alarm Date register.
≡	PLIB_RTCC_AlarmDaySet	Writes to the specific RTCC module's Alarm Date value register.
≡	PLIB_RTCC_AlarmDisable	Disables the specific RTCC module's alarm.
≡	PLIB_RTCC_AlarmEnable	Enables the specific RTCC module's alarm.
≡	PLIB_RTCC_AlarmHourGet	Returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
≡	PLIB_RTCC_AlarmHourSet	The function returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
≡	PLIB_RTCC_AlarmMaskModeSelect	Sets the specific RTCC module's alarm mask Configuration bits.
≡	PLIB_RTCC_AlarmMinuteGet	Returns the contents of Minutes bits in the specific RTCC module's Alarm Time register.
≡	PLIB_RTCC_AlarmMinuteSet	Returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register.
≡	PLIB_RTCC_AlarmMonthGet	Returns the contents of the Month bits in the specific RTCC module's Alarm Date register.
≡	PLIB_RTCC_AlarmMonthSet	Writes to the specific RTCC module's Alarm Date register.
≡	PLIB_RTCC_AlarmPulseInitialGet	Returns the state of the initial alarm pulse.
≡	PLIB_RTCC_AlarmPulseInitialSet	Enables the determination of the initial alarm pulse.
≡	PLIB_RTCC_AlarmRepeatCountGet	Reads the specific RTCC module's alarm repeat counter.
≡	PLIB_RTCC_AlarmRepeatCountSet	Sets the specific RTCC module's alarm repeat counter.
≡	PLIB_RTCC_AlarmSecondGet	Returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register.
≡	PLIB_RTCC_AlarmSecondSet	Returns the contents of Seconds bits in the specific RTCC module's Alarm Time register.
≡	PLIB_RTCC_AlarmTimeGet	Returns the contents of the specific RTCC module's Alarm Time register.
≡	PLIB_RTCC_AlarmTimeSet	Writes to the specific RTCC module's Alarm Time register.
≡	PLIB_RTCC_AlarmValueRegisterPointer	Sets the specific RTCC module's Alarm register pointer.
≡	PLIB_RTCC_AlarmWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Alarm Date register.
≡	PLIB_RTCC_AlarmWeekDaySet	Writes to the specific RTCC module's Alarm Date register.

Feature Existence Functions

	Name	Description
≡	PLIB_RTCC_ExistsAlarmChimeControl	Identifies whether the AlarmChimeControl feature exists on the RTCC module.
≡	PLIB_RTCC_ExistsAlarmControl	Identifies whether the AlarmControl feature exists on the RTCC module.
≡	PLIB_RTCC_ExistsAlarmDate	Identifies whether the AlarmDate feature exists on the RTCC module.
≡	PLIB_RTCC_ExistsAlarmMaskControl	Identifies whether the AlarmMaskControl feature exists on the RTCC module.
≡	PLIB_RTCC_ExistsAlarmPulseInitial	Identifies whether the AlarmPulseInitial feature exists on the RTCC module.

◆	PLIB_RTCC_ExistsAlarmRepeatControl	Identifies whether the AlarmRepeatControl feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsAlarmSynchronization	Identifies whether the AlarmSynchronization feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsAlarmTime	Identifies whether the AlarmTime feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsCalibration	Identifies whether the Calibration feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsClockRunning	Identifies whether the ClockRunning feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsClockSelect	Identifies whether the ClockSelect feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsHalfSecond	Identifies whether the HalfSecond feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsOutputControl	Identifies whether the OutputControl feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsOutputSelect	Identifies whether the OutputSelect feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsRTCDate	Identifies whether the RTCDateValue feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsRTCTime	Identifies whether the RTCTimeValue feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsSynchronization	Identifies whether the Synchronization feature exists on the RTCC module.
◆	PLIB_RTCC_ExistsWriteEnable	Identifies whether the WriteEnable feature exists on the RTCC module.

General Functions

	Name	Description
◆	PLIB_RTCC_ClockOutputDisable	Disables the specific RTCC module's output pin.
◆	PLIB_RTCC_ClockOutputEnable	Enables the specific RTCC module's output pin.
◆	PLIB_RTCC_ClockRunningStatus	Provides the status of the RTCC clock.

Other Functions

	Name	Description
◆	PLIB_RTCC_ClockSourceSelect	Selects the clock source for the RTCC module.
◆	PLIB_RTCC_DriftCalibrateGet	Reads the specific RTCC module's drift calibration bits.
◆	PLIB_RTCC_DriftCalibrateSet	Sets the specific RTCC module's drift calibration bits.
◆	PLIB_RTCC_HalfSecondStatusGet	The function returns the half second status bit.
◆	PLIB_RTCC_OutputSelect	Selects which signal will be presented on the RTCC pin
◆	PLIB_RTCC_StopInIdleDisable	Continues normal RTCC operation when the device enters Idle mode.
◆	PLIB_RTCC_StopInIdleEnable	Disables access to the RTCC module by the Peripheral Bus Clock (PBCLK) when the CPU enters Idle mode.

RTCC Mode Functions

	Name	Description
◆	PLIB_RTCC_Disable	Disables the specific RTCC module on the device.
◆	PLIB_RTCC_Enable	Enables the specific RTCC module on the device.
◆	PLIB_RTCC_WriteDisable	Disables writing to the specific RTCC module's value registers.
◆	PLIB_RTCC_WriteEnable	Enables writing to the specific RTCC module's value registers.
◆	PLIB_RTCC_RTCDateGet	Returns the contents of the specific RTCC module's Date register.
◆	PLIB_RTCC_RTCDateSet	Writes to the specific RTCC module's Date register.
◆	PLIB_RTCC_RTCTimeGet	Returns the contents of the Days bits in the specific RTCC module's Date register.
◆	PLIB_RTCC_RTCTimeSet	Writes to the specific RTCC module's Date register.
◆	PLIB_RTCC_RTCHourGet	Returns the contents of the Hours bits in the specific RTCC module's Time register.

	PLIB_RTCC_RTCHourSet	Writes the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteGet	Returns the contents of the Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteSet	Writes the contents of Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMonthGet	Returns the contents of the Months bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCMonthSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCSecondGet	The function returns the contents of the Seconds bits in the specific RTCC device's Time register.
	PLIB_RTCC_RTCSecondSet	Writes the contents of Seconds bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeGet	Returns the contents of the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeSet	Writes to the specific RTCC module's Time register.
	PLIB_RTCC_RTCWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCWeekDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearGet	Returns the contents of the Year bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearSet	Writes to the specific RTCC module's Date register.

Description

This section lists and describes the functions, data types, and constants included in the RTCC Peripheral Library.

5.6.20.5.1 General Functions**5.6.20.5.1.1 PLIB_RTCC_ClockOutputDisable Function****C**

```
void PLIB_RTCC_ClockOutputDisable(
    RTCC_MODULE_ID index
);
```

Description

This function disables the specific RTCC module's output pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the OutputControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsOutputControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_ClockOutputDisable(RTCC_ID_0);
```

5.6.20.5.1.2 PLIB_RTCC_ClockOutputEnable Function

C

```
void PLIB_RTCC_ClockOutputEnable(  
    RTCC_MODULE_ID index  
);
```

Description

This function enables the specific RTCC module's output and generates a square wave using either the alarm or the 1 Hz clock output on the RTCC pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the OutputControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsOutputControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_ClockOutputEnable(RTCC_ID_0);
```

5.6.20.5.1.3 PLIB_RTCC_ClockRunningStatus Function

C

```
bool PLIB_RTCC_ClockRunningStatus(  
    RTCC_MODULE_ID index  
);
```

Description

This function provides the status of the RTCC clock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

The status of the RTCC clock.

Remarks

This function implements an operation of the ClockRunning feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsClockRunning](#) in your application to automatically determine whether this feature is available.

Example

```
bool status;  
status = PLIB_RTCC_ClockRunningStatus(RTCC_ID_0);
```

5.6.20.5.2 RTCC Mode Functions

5.6.20.5.2.1 PLIB_RTCC_Disable Function

C

```
void PLIB_RTCC_Disable(  
    RTCC_MODULE_ID index  
);
```

Description

This function disables the specific RTCC module on the device.

Preconditions

The RTCC module should be unlocked for writing using the function [PLIB_RTCC_WriteEnable](#) before this function is called.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsEnableControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_Disable(RTCC_ID_0);
```

5.6.20.5.2.2 PLIB_RTCC_Enable Function

C

```
void PLIB_RTCC_Enable(  
    RTCC_MODULE_ID index  
);
```

Description

This function enables the specific RTCC module on the device.

Preconditions

The RTCC module should be unlocked for writing using the function [PLIB_RTCC_WriteEnable](#) before this function is called.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

By calling this function, the RTCC pins are controlled by the RTCC module. The RTCC module will continue to function when the device is held in reset.

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsEnableControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_Enable(RTCC_ID_0);
```

5.6.20.5.2.3 PLIB_RTCC_WriteDisable Function**C**

```
void PLIB_RTCC_WriteDisable(  
    RTCC_MODULE_ID index  
);
```

Description

This function disables writing to the specific RTCC module's value registers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the WriteEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsWriteEnable](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_WriteDisable(RTCC_ID_0);
```

5.6.20.5.2.4 PLIB_RTCC_WriteEnable Function**C**

```
void PLIB_RTCC_WriteEnable(  
    RTCC_MODULE_ID index  
);
```

Description

This function enables writing to the specific RTCC module's value registers.

Preconditions

The SYSLOCK unlock sequence must be executed prior to calling this function by calling the PLIB_CORE_SysUnlock function.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the WriteEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsWriteEnable](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_WriteEnable(RTCC_ID_0);
```

5.6.20.5.2.5 PLIB_RTCC_RTCCDateGet Function**C**

```
uint32_t PLIB_RTCC_RTCCDateGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the contents of the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Date register contents.

Remarks

This function implements an operation of the RTCCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCCDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Date;  
Date = PLIB_RTCC_RTCCDateGet(RTCC_ID_0);
```

5.6.20.5.2.6 PLIB_RTCC_RTCCDateSet Function**C**

```
void PLIB_RTCC_RTCCDateSet(  
    RTCC_MODULE_ID index,  
    uint32_t data  
);
```

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the RTCCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCCDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t data = 0x06102705;    //Date = 27 Oct 2006 Friday
PLIB_RTCC_RTCCDateSet(RTCC_ID_0, data);
```

5.6.20.5.2.7 PLIB_RTCC_RTCCDayGet Function

C

```
uint32_t PLIB_RTCC_RTCCDayGet(
    RTCC_MODULE_ID index
);
```

Description

The function returns the contents of the Days bits in the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Days bits in the Date register.

Remarks

This function implements an operation of the RTCCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCCDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Day;
Day = PLIB_RTCC_RTCCDayGet(RTCC_ID_0);
```

5.6.20.5.2.8 PLIB_RTCC_RTCCDaySet Function

C

```
void PLIB_RTCC_RTCCDaySet(
    RTCC_MODULE_ID index,
    uint32_t day
);
```


Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
day	The BCD value of the day to set in the Date register

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the RTCCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCCDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Day = 0x27;           //Day = 27th of the month
PLIB_RTCC_RTCCDaySet(RTCC_ID_0, Day);
```

5.6.20.5.2.9 PLIB_RTCC_RTCHourGet Function

C

```
uint32_t PLIB_RTCC_RTCHourGet(
    RTCC_MODULE_ID index
);
```

Description

The function returns the contents of the Hours bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

BCD value of the Hours bits in the Time register.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Hour;
Hour = PLIB_RTCC_RTCHourGet(RTCC_ID_0);
```

5.6.20.5.2.10 PLIB_RTCC_RTCHourSet Function

C

```
void PLIB_RTCC_RTCHourSet(  
    RTCC_MODULE_ID index,  
    uint32_t hour  
);
```

Description

The function writes the contents of the Hours bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
hour	BCD value to be written to the Hours bits in the Time register

Returns

None

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Hour = 0x04;  
PLIB_RTCC_RTCHourSet(RTCC_ID_0, Hour);
```

5.6.20.5.2.11 PLIB_RTCC_RTCMinuteGet Function

C

```
uint32_t PLIB_RTCC_RTCMinuteGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the contents of the Minutes bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

BCD value of the Minutes bits in the Time register.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Minute;
```

```
Minute = PLIB_RTCC_RTCMinuteGet(RTCC_ID_0);
```

5.6.20.5.2.12 PLIB_RTCC_RTCMinuteSet Function

C

```
void PLIB_RTCC_RTCMinuteSet(  
    RTCC_MODULE_ID index,  
    uint32_t minute  
);
```

Description

The function writes the contents of these bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minute	BCD value to be written to the Minutes bits in the Time register

Returns

None

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Minute = 0x15;  
PLIB_RTCC_RTCMinuteSet(RTCC_ID_0, Minute);
```

5.6.20.5.2.13 PLIB_RTCC_RTCMonthGet Function

C

```
uint32_t PLIB_RTCC_RTCMonthGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the contents of the Months bits in the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Months bits in the Date register.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Month;  
Month = PLIB_RTCC_RTCMonthGet(RTCC_ID_0);
```

5.6.20.5.2.14 PLIB_RTCC_RTCMonthSet Function**C**

```
void PLIB_RTCC_RTCMonthSet(  
    RTCC_MODULE_ID index,  
    uint32_t month  
);
```

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
month	The BCD value of the month to set in the Date register

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the RTCCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCCDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Month = 0x10;    //Month = October  
PLIB_RTCC_RTCMonthSet(RTCC_ID_0, Month);
```

5.6.20.5.2.15 PLIB_RTCC_RTCSecondGet Function**C**

```
uint32_t PLIB_RTCC_RTCSecondGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the contents of the Seconds bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

BCD value of the Seconds bits in the Time register.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Second;  
Second = PLIB_RTCC_RTCSecondGet(RTCC_ID_0);
```

5.6.20.5.2.16 PLIB_RTCC_RTCSecondSet Function**C**

```
void PLIB_RTCC_RTCSecondSet(  
    RTCC_MODULE_ID index,  
    uint32_t second  
);
```

Description

The function writes the contents of the Seconds bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
second	BCD value to be written to the Seconds bits in the Time register

Returns

None.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Second = 0x33;  
PLIB_RTCC_RTCSecondSet(RTCC_ID_0, Second);
```

5.6.20.5.2.17 PLIB_RTCC_RTCTimeGet Function**C**

```
uint32_t PLIB_RTCC_RTCTimeGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the contents of the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Time register contents.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t time;
time = PLIB_RTCC_RTCTimeGet(RTCC_ID_0);
```

5.6.20.5.2.18 PLIB_RTCC_RTCTimeSet Function**C**

```
void PLIB_RTCC_RTCTimeSet(
    RTCC_MODULE_ID index,
    uint32_t data
);
```

Description

The function writes to the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Time register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t data = 0x04153300;    //Time = 4 hrs 15 mins 33 sec
PLIB_RTCC_RTCTimeSet(RTCC_ID_0, data);
```

5.6.20.5.2.19 PLIB_RTCC_RTCWeekDayGet Function**C**

```
uint32_t PLIB_RTCC_RTCWeekDayGet(
    RTCC_MODULE_ID index
);
```

Description

The function returns the contents of the WeekDay bits in the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

WeekDay field in the Date register.

Remarks

This function implements an operation of the RTCCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCCDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t WeekDay;  
WeekDay = PLIB_RTCC_RTCWeekDayGet(RTCC_ID_0);
```

5.6.20.5.2.20 PLIB_RTCC_RTCWeekDaySet Function**C**

```
void PLIB_RTCC_RTCWeekDaySet(  
    RTCC_MODULE_ID index,  
    uint32_t weekday  
);
```

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
weekday	The BCD value of the weekday to set in the Date register

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the RTCCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCCDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t WeekDay = 0x05;           //WeekDay = Friday  
PLIB_RTCC_RTCWeekDaySet(RTCC_ID_0, WeekDay);
```

5.6.20.5.2.21 PLIB_RTCC_RTCYearGet Function**C**

```
uint32_t PLIB_RTCC_RTCYearGet(  
    RTCC_MODULE_ID index
```

```
);
```

Description

The function returns the contents of the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Year bits in the Date register.

Remarks

This function implements an operation of the RTCCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCCDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Year;  
Year = PLIB_RTCC_RTCYearGet(RTCC_ID_0);
```

5.6.20.5.2.22 PLIB_RTCC_RTCYearSet Function**C**

```
void PLIB_RTCC_RTCYearSet(  
    RTCC_MODULE_ID index,  
    uint32_t year  
);
```

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
year	The BCD value of the year to set in the Date register

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the RTCCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsRTCCDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Year = 0x06;    //Year = 2006  
PLIB_RTCC_RTCYearSet(RTCC_ID_0, Year);
```


5.6.20.5.3 Alarm Mode Functions

5.6.20.5.3.1 PLIB_RTCC_AlarmChimeDisable Function

C

```
void PLIB_RTCC_AlarmChimeDisable(  
    RTCC_MODULE_ID index  
);
```

Description

This function disables the specific RTCC module's chime. The alarm repeat count value bits stop once they reach zero.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the AlarmChimeControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmChimeControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_AlarmChimeDisable(RTCC_ID_0);
```

5.6.20.5.3.2 PLIB_RTCC_AlarmChimeEnable Function

C

```
void PLIB_RTCC_AlarmChimeEnable(  
    RTCC_MODULE_ID index  
);
```

Description

This function enables the specific RTCC module's chime. The alarm repeat count bits are allowed to rollover.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the AlarmChimeControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmChimeControl](#) in your

application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_AlarmChimeEnable(RTCC_ID_0);
```

5.6.20.5.3.3 PLIB_RTCC_AlarmDateGet Function

C

```
uint32_t PLIB_RTCC_AlarmDateGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the contents of the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Value register.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Date;  
Date = PLIB_RTCC_AlarmDateGet(RTCC_ID_0);
```

5.6.20.5.3.4 PLIB_RTCC_AlarmDateSet Function

C

```
void PLIB_RTCC_AlarmDateSet(  
    RTCC_MODULE_ID index,  
    uint32_t data  
);
```

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	The value to set the Alarm Date register to, in BCD format

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t data = 0x06102705;    //Date = 27 Oct 2006 Friday
PLIB_RTCC_AlarmDateSet(RTCC_ID_0, data);
```

5.6.20.5.3.5 PLIB_RTCC_AlarmDayGet Function**C**

```
uint32_t PLIB_RTCC_AlarmDayGet(
    RTCC_MODULE_ID index
);
```

Description

The function returns the contents of Day bits in the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Days bits in the Alarm Date register.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Day;
Day = PLIB_RTCC_AlarmDayGet(RTCC_ID_0);
```

5.6.20.5.3.6 PLIB_RTCC_AlarmDaySet Function**C**

```
void PLIB_RTCC_AlarmDaySet(
    RTCC_MODULE_ID index,
    uint32_t day
);
```

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
day	The BCD value of the day to set in the Alarm Date register

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Day = 0x27;           //Day = 27th of the month
PLIB_RTCC_AlarmDaySet(RTCC_ID_0, Day);
```

5.6.20.5.3.7 PLIB_RTCC_AlarmDisable Function**C**

```
void PLIB_RTCC_AlarmDisable(
    RTCC_MODULE_ID index
);
```

Description

This function disables the specific RTCC module's alarm.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the AlarmControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_AlarmDisable(RTCC_ID_0);
```

5.6.20.5.3.8 PLIB_RTCC_AlarmEnable Function**C**

```
void PLIB_RTCC_AlarmEnable(
    RTCC_MODULE_ID index
);
```

Description

This function enables the specific RTCC module's alarm.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

The alarm enable bit is cleared automatically after an alarm event whenever the alarm is not set up to repeat, and chime is disabled.

This function implements an operation of the AlarmControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmControl](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_AlarmEnable(RTCC_ID_0);
```

5.6.20.5.3.9 PLIB_RTCC_AlarmHourGet Function**C**

```
uint32_t PLIB_RTCC_AlarmHourGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the contents of the Hours bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

BCD value of the Hours bits in the Alarm Time register.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Hour;  
Hour = PLIB_RTCC_AlarmHourGet(RTCC_ID_0);
```

5.6.20.5.3.10 PLIB_RTCC_AlarmHourSet Function

C

```
void PLIB_RTCC_AlarmHourSet(  
    RTCC_MODULE_ID index,  
    uint32_t hour  
);
```

Description

Returns the contents of the Hours bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
hour	BCD value to be written to the Hours bits in the Alarm Time register

Returns

None

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Hour = 0x04;  
PLIB_RTCC_AlarmHourSet(RTCC_ID_0, Hour);
```

5.6.20.5.3.11 PLIB_RTCC_AlarmMaskModeSelect Function

C

```
void PLIB_RTCC_AlarmMaskModeSelect(  
    RTCC_MODULE_ID index,  
    RTCC_ALARM_MASK_CONFIGURATION data  
);
```

Description

This function sets the specific RTCC module's alarm mask Configuration bits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Alarm mask Configuration bits

Returns

None.

Remarks

The actual definition of this enumeration is device-specific.

This function implements an operation of the AlarmMaskControl feature. This feature may not be available on all devices. Please

refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmMaskControl](#) in your application to automatically determine whether this feature is available.

Example

```
uint8_t data = 0;
PLIB_RTCC_AlarmMaskModeSelect(RTCC_ID_0, RTCC_ALARM_EVERY_HALF_SECOND);
```

5.6.20.5.3.12 PLIB_RTCC_AlarmMinuteGet Function**C**

```
uint32_t PLIB_RTCC_AlarmMinuteGet(
    RTCC_MODULE_ID index
);
```

Description

The function returns the contents of the field in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

BCD value of the Minutes bits in the Alarm Time register.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Minute;
Minute = PLIB_RTCC_AlarmMinuteGet(RTCC_ID_0);
```

5.6.20.5.3.13 PLIB_RTCC_AlarmMinuteSet Function**C**

```
void PLIB_RTCC_AlarmMinuteSet(
    RTCC_MODULE_ID index,
    uint32_t minute
);
```

Description

The function returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minute	BCD value to be written to the Minutes bits in the Alarm Time register

Returns

None

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Minute = 0x15;
PLIB_RTCC_AlarmMinuteSet(RTCC_ID_0, Minute);
```

5.6.20.5.3.14 PLIB_RTCC_AlarmMonthGet Function**C**

```
uint32_t PLIB_RTCC_AlarmMonthGet(
    RTCC_MODULE_ID index
);
```

Description

The function returns the contents of the Months bits in the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Months bits in the Date register.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Month;
Month = PLIB_RTCC_AlarmMonthGet(RTCC_ID_0);
```

5.6.20.5.3.15 PLIB_RTCC_AlarmMonthSet Function**C**

```
void PLIB_RTCC_AlarmMonthSet(
    RTCC_MODULE_ID index,
    uint32_t month
);
```

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

month	The BCD value of the month to set in the Alarm Date register

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Month = 0x10;           //Month = October
PLIB_RTCC_AlarmMonthSet(RTCC_ID_0, Month);
```

5.6.20.5.3.16 PLIB_RTCC_AlarmPulseInitialGet Function**C**

```
bool PLIB_RTCC_AlarmPulseInitialGet(
    RTCC_MODULE_ID index
);
```

Description

This function returns the state of the initial alarm pulse.

Preconditions

The ALRMEN bit should be '1' indicating the [PLIB_RTCC_AlarmEnable](#) function was called.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

1 - Logical High 0 - Logical Low

Remarks

This function implements an operation of the AlarmPulseInitial feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmPulseInitial](#) in your application to automatically determine whether this feature is available.

Example

```
bool PulseValue;
PulseValue = PLIB_RTCC_AlarmPulseInitialGet(RTCC_ID_0);
```

5.6.20.5.3.17 PLIB_RTCC_AlarmPulseInitialSet Function**C**

```
void PLIB_RTCC_AlarmPulseInitialSet(
    RTCC_MODULE_ID index,
    bool data
);
```

Description

This function enables the determination of initial alarm pulse.

Preconditions

The ALRMEN bit should be '0' indicating the [PLIB_RTCC_AlarmDisable](#) was called. This function must not be called when the RTCC is ON and the Alarm Sync is 1.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the AlarmPulseInitial feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmPulseInitial](#) in your application to automatically determine whether this feature is available.

Example

```
bool data = 0;
PLIB_RTCC_AlarmPulseInitialSet(RTCC_ID_0, data);
```

5.6.20.5.3.18 PLIB_RTCC_AlarmRepeatCountGet Function**C**

```
uint8_t PLIB_RTCC_AlarmRepeatCountGet(
    RTCC_MODULE_ID index
);
```

Description

This function reads the specific RTCC module's alarm repeat counter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

uint8_t - The current value of the alarm repeat counter

Remarks

The counter decrements on any alarm event. The counter is prevented from rolling over unless chime is enabled.

This function implements an operation of the AlarmRepeatControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmRepeatControl](#) in your application to automatically determine whether this feature is available.

Example

```
uint8_t currentCount;
currentCount = PLIB_RTCC_AlarmRepeatCountGet(RTCC_ID_0);
```

5.6.20.5.3.19 PLIB_RTCC_AlarmRepeatCountSet Function

C

```
void PLIB_RTCC_AlarmRepeatCountSet(  
    RTCC_MODULE_ID index,  
    uint8_t data  
);
```

Description

This function sets the specific RTCC module's alarm repeat counter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Alarm repeat counter bits

Returns

None.

Remarks

The counter decrements on any alarm event. The counter is prevented from rolling over unless chime is enabled.

This function implements an operation of the AlarmRepeatControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmRepeatControl](#) in your application to automatically determine whether this feature is available.

Example

```
uint8_t data = 0xFF;  
PLIB_RTCC_AlarmRepeatCountSet(RTCC_ID_0, data);
```

5.6.20.5.3.20 PLIB_RTCC_AlarmSecondGet Function

C

```
uint32_t PLIB_RTCC_AlarmSecondGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

BCD value of the Seconds bits in the Alarm Time register.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmTime](#) in your application to automatically

determine whether this feature is available.

Example

```
uint32_t Second;  
Second = PLIB_RTCC_AlarmSecondGet(RTCC_ID_0);
```

5.6.20.5.3.21 PLIB_RTCC_AlarmSecondSet Function**C**

```
void PLIB_RTCC_AlarmSecondSet(  
    RTCC_MODULE_ID index,  
    uint32_t second  
);
```

Description

The function returns the contents of the field in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
second	BCD value to be written to the Seconds bits in the Alarm Time register

Returns

None

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t Second = 0x33;  
PLIB_RTCC_AlarmSecondSet(RTCC_ID_0, Second);
```

5.6.20.5.3.22 PLIB_RTCC_AlarmTimeGet Function**C**

```
uint32_t PLIB_RTCC_AlarmTimeGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the contents of the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Value register.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to

the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t time;  
time = PLIB_RTCC_AlarmTimeGet(RTCC_ID_0);
```

5.6.20.5.3.23 PLIB_RTCC_AlarmTimeSet Function**C**

```
void PLIB_RTCC_AlarmTimeSet(  
    RTCC_MODULE_ID index,  
    uint32_t data  
);
```

Description

The function writes to the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Alarm Time register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmTime](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t data = 0x04153300;    //Time = 4 hrs 15 mins 33 sec  
PLIB_RTCC_AlarmTimeSet(RTCC_ID_0, data);
```

5.6.20.5.3.24 PLIB_RTCC_AlarmValueRegisterPointer Function**C**

```
void PLIB_RTCC_AlarmValueRegisterPointer(  
    RTCC_MODULE_ID index,  
    uint8_t data  
);
```

Description

This function sets the specific RTCC module's Alarm register pointer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Alarm register pointer

Returns

None.

Remarks

None.

Example

```
uint8_t data = 2;
PLIB_RTCC_AlarmValueRegisterPointer(RTCC_ID_0, data);
```

5.6.20.5.3.25 PLIB_RTCC_AlarmWeekDayGet Function**C**

```
uint32_t PLIB_RTCC_AlarmWeekDayGet(
    RTCC_MODULE_ID index
);
```

Description

The function returns the contents of Weekday bits in the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

WeekDay bits in the Alarm Date register.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t WeekDay;
WeekDay = PLIB_RTCC_AlarmWeekDayGet(RTCC_ID_0);
```

5.6.20.5.3.26 PLIB_RTCC_AlarmWeekDaySet Function**C**

```
void PLIB_RTCC_AlarmWeekDaySet(
    RTCC_MODULE_ID index,
    uint32_t weekday
);
```

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
weekday	The BCD value of the weekday to set in the field of the Alarm Date register

Returns

None.

Remarks

A write to this register is only allowed when access is allowed by using the [PLIB_RTCC_WriteEnable](#) function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsAlarmDate](#) in your application to automatically determine whether this feature is available.

Example

```
uint32_t WeekDay = 0x05;           //WeekDay = Friday
PLIB_RTCC_AlarmWeekDaySet(RTCC_ID_0, WeekDay);
```

5.6.20.5.4 Other Functions

5.6.20.5.4.1 PLIB_RTCC_ClockSourceSelect Function

C

```
void PLIB_RTCC_ClockSourceSelect(
    RTCC_MODULE_ID index,
    RTCC_CLOCK_SOURCE_SELECT source
);
```

Description

This function determines which clock source the RTCC module will use depending on the features of the device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Which clock source will be used

Returns

None.

Remarks

This function implements an operation of the ClockSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsClockSelect](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_ClockSourceSelect(RTCC_ID_0, RTCC_CLOCK_SOURCE_SELECT_NONE);
```

5.6.20.5.4.2 PLIB_RTCC_DriftCalibrateGet Function

C

```
uint16_t PLIB_RTCC_DriftCalibrateGet(  
    RTCC_MODULE_ID index  
);
```

Description

This function reads the specific RTCC module's drift calibration bits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

uint16_t - The current drift calibration value

Remarks

This function implements an operation of the Calibration feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsCalibration](#) in your application to automatically determine whether this feature is available.

Example

```
uint16_t calibrationbits;  
calibrationbits = PLIB_RTCC_DriftCalibrateGet(RTCC_ID_0);
```

5.6.20.5.4.3 PLIB_RTCC_DriftCalibrateSet Function

C

```
void PLIB_RTCC_DriftCalibrateSet(  
    RTCC_MODULE_ID index,  
    uint16_t calibrationbits  
);
```

Description

This function sets the specific RTCC module's drift calibration bits. The error between the system clock and the external clock has to be computed and calibration input must be provided to this function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
calibrationbits	Drift calibration bits

Returns

None.

Remarks

This function implements an operation of the Calibration feature. This feature may not be available on all devices. Please refer to

the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsCalibration](#) in your application to automatically determine whether this feature is available.

Example

```
uint16_t calibrationbits = 3; //Positive 3 adjustment derived from the formula  
                                // Error = (Ideal Freq(32758) - Measured)*60;  
PLIB_RTCC_DriftCalibrateSet(RTCC_ID_0, calibrationbits);
```

5.6.20.5.4.4 PLIB_RTCC_HalfSecondStatusGet Function**C**

```
bool PLIB_RTCC_HalfSecondStatusGet(  
    RTCC_MODULE_ID index  
);
```

Description

The function returns the half second status bit, which is used in the calibration procedure. When the seconds byte is zero, the calibration value must be updated when the half second bit becomes '1'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Second half period of a second
- false - First half period of a second

Remarks

This bit is read-only. It is cleared to '0' on a write to the seconds value. This function implements an operation of the HalfSecond feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsHalfSecond](#) in your application to automatically determine whether this feature is available.

Example

```
// Wait for the half second status bit to be '1'.  
while(PLIB_RTCC_HalfSecondStatusGet(RTCC_ID_0));
```

5.6.20.5.4.5 PLIB_RTCC_OutputSelect Function**C**

```
void PLIB_RTCC_OutputSelect(  
    RTCC_MODULE_ID index,  
    RTCC_OUTPUT_SELECT data  
);
```

Description

This function selects which signal will be presented on the RTCC pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Enumerated value of which signal to present

Returns

None.

Remarks

The RTCC module's output pin should be enabled using the function `PLIB_RTCC_OutputEnable`.

This function implements an operation of the OutputSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_RTCC_ExistsOutputSelect](#) in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_OutputSelect(RTCC_ID_0, RTCC_OUTPUT_SECONDS_CLOCK);
```

5.6.20.5.4.6 PLIB_RTCC_StopInIdleDisable Function**C**

```
void PLIB_RTCC_StopInIdleDisable(  
    RTCC_MODULE_ID index  
);
```

Description

This function continues normal RTCC operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsStopInIdle` in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_StopInIdleDisable(RTCC_ID_0);
```

5.6.20.5.4.7 PLIB_RTCC_StopInIdleEnable Function**C**

```
void PLIB_RTCC_StopInIdleEnable(  
    RTCC_MODULE_ID index  
);
```

Description

This function disables access to the RTCC module by the PBCLK when the CPU is in Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_RTCC_ExistsStopInIdle` in your application to automatically determine whether this feature is available.

Example

```
PLIB_RTCC_StopInIdleEnable(RTCC_ID_0);
```

5.6.20.5.5 Data Types and Constants

5.6.20.5.5.1 RTCC_ALARM_MASK_CONFIGURATION Enumeration

C

```
typedef enum {  
    RTCC_ALARM_EVERY_HALF_SECOND,  
    RTCC_ALARM_EVERY_SECOND,  
    RTCC_ALARM_EVERY_10_SECONDS,  
    RTCC_ALARM_EVERY_MINUTE,  
    RTCC_ALARM_EVERY_10_MINUTES,  
    RTCC_ALARM_EVERY_HOUR,  
    RTCC_ALARM_ONCE_A_DAY,  
    RTCC_ALARM_ONCE_A_WEEK,  
    RTCC_ALARM_ONCE_A_MONTH,  
    RTCC_ALARM_ONCE_A_YEAR  
} RTCC_ALARM_MASK_CONFIGURATION;
```

Description

Alarm Mask Configuration

This data type defines the different configurations for accessing the alarm mask Configuration bits.

Members

Members	Description
RTCC_ALARM_EVERY_HALF_SECOND	Alarm every half second
RTCC_ALARM_EVERY_SECOND	Alarm every second
RTCC_ALARM_EVERY_10_SECONDS	Alarm every 10 seconds
RTCC_ALARM_EVERY_MINUTE	Alarm every minute
RTCC_ALARM_EVERY_10_MINUTES	Alarm every 10 minutes
RTCC_ALARM_EVERY_HOUR	Alarm every hour
RTCC_ALARM_ONCE_A_DAY	Alarm every day

RTCC_ALARM_ONCE_A_WEEK	Alarm every week
RTCC_ALARM_ONCE_A_MONTH	Alarm every month
RTCC_ALARM_ONCE_A_YEAR	Alarm every year

Remarks

The actual definition of this enumeration is device-specific.

5.6.20.5.5.2 RTCC_MODULE_ID Enumeration**C**

```
typedef enum {
    RTCC_ID_1,
    RTCC_NUMBER_OF_MODULES
} RTCC_MODULE_ID;
```

Description

Enumeration: RTCC_MODULE_ID

This enumeration defines number of modules which are available on the microcontroller. This is the superset of all of the possible instances that may be available on the Microchip microcontrollers.

Refer to the data sheet to get the correct number of modules defined for desired microcontroller.

Members

Members	Description
RTCC_NUMBER_OF_MODULES	The total number of modules available.

5.6.20.5.5.3 RTCC_VALUE_REGISTER_POINTER Enumeration**C**

```
typedef enum {
    RTCC_REG_YEAR,
    RTCC_REG_DAY_MONTH,
    RTCC_REG_HOURS_WEEKDAY,
    RTCC_REG_MINUTES_SECONDS
} RTCC_VALUE_REGISTER_POINTER;
```

Description

Value Register Pointer

This data type defines the different configurations by which the RTCC Date and Time Registers can be accessed.

Members

Members	Description
RTCC_REG_YEAR	The year register is being pointed
RTCC_REG_DAY_MONTH	The day and month register is being pointed
RTCC_REG_HOURS_WEEKDAY	The hours and weekday register is being pointed
RTCC_REG_MINUTES_SECONDS	The minutes and seconds register is being pointed

Remarks

The actual definition of this enumeration is device-specific.

5.6.20.5.6 Feature Existence Functions

5.6.20.5.6.1 PLIB_RTCC_ExistsAlarmChimeControl Function

C

```
bool PLIB_RTCC_ExistsAlarmChimeControl(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the AlarmChimeControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_AlarmChimeEnable](#)
- [PLIB_RTCC_AlarmChimeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlarmChimeControl feature is supported on the device
- false - The AlarmChimeControl feature is not supported on the device

Remarks

None.

5.6.20.5.6.2 PLIB_RTCC_ExistsAlarmControl Function

C

```
bool PLIB_RTCC_ExistsAlarmControl(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the AlarmControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_AlarmEnable](#)
- [PLIB_RTCC_AlarmDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlarmControl feature is supported on the device
- false - The AlarmControl feature is not supported on the device

Remarks

None.

5.6.20.5.6.3 PLIB_RTCC_ExistsAlarmDate Function**C**

```
bool PLIB_RTCC_ExistsAlarmDate(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the AlarmDate feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_AlarmDateGet](#)
- [PLIB_RTCC_AlarmDateSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlarmDate feature is supported on the device
- false - The AlarmDate feature is not supported on the device

Remarks

None.

5.6.20.5.6.4 PLIB_RTCC_ExistsAlarmMaskControl Function**C**

```
bool PLIB_RTCC_ExistsAlarmMaskControl(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the AlarmMaskControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_AlarmMaskModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlarmMaskControl feature is supported on the device

- false - The AlarmMaskControl feature is not supported on the device

Remarks

None.

5.6.20.5.6.5 PLIB_RTCC_ExistsAlarmPulseInitial Function**C**

```
bool PLIB_RTCC_ExistsAlarmPulseInitial(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the AlarmPulseInitialValue feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_AlarmPulseInitialSet](#)
- [PLIB_RTCC_AlarmPulseInitialGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlarmPulseInitial feature is supported on the device
- false - The AlarmPulseInitial feature is not supported on the device

Remarks

None.

5.6.20.5.6.6 PLIB_RTCC_ExistsAlarmRepeatControl Function**C**

```
bool PLIB_RTCC_ExistsAlarmRepeatControl(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the AlarmRepeatControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_AlarmRepeatCountSet](#)
- [PLIB_RTCC_AlarmRepeatCountRead](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlarmRepeatControl feature is supported on the device
- false - The AlarmRepeatControl feature is not supported on the device

Remarks

None.

5.6.20.5.6.7 PLIB_RTCC_ExistsAlarmSynchronization Function**C**

```
bool PLIB_RTCC_ExistsAlarmSynchronization(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the AlarmSynchronization feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmReadSyncBit

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlarmSynchronization feature is supported on the device
- false - The AlarmSynchronization feature is not supported on the device

Remarks

None.

5.6.20.5.6.8 PLIB_RTCC_ExistsAlarmTime Function**C**

```
bool PLIB_RTCC_ExistsAlarmTime(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the AlarmTime feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_AlarmTimeGet](#)
- [PLIB_RTCC_AlarmTimeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlarmTime feature is supported on the device
- false - The AlarmTime feature is not supported on the device

Remarks

None.

5.6.20.5.6.9 PLIB_RTCC_ExistsCalibration Function**C**

```
bool PLIB_RTCC_ExistsCalibration(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the Calibration feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_DriftCalibrate

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Calibration feature is supported on the device
- false - The Calibration feature is not supported on the device

Remarks

None.

5.6.20.5.6.10 PLIB_RTCC_ExistsClockRunning Function**C**

```
bool PLIB_RTCC_ExistsClockRunning(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the ClockRunning feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- [PLIB_RTCC_ClockRunningStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ClockRunning feature is supported on the device
- false - The ClockRunning feature is not supported on the device

Remarks

None.

5.6.20.5.6.11 PLIB_RTCC_ExistsClockSelect Function**C**

```
bool PLIB_RTCC_ExistsClockSelect(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the ClockSelect feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_SourceClockSelect

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ClockSelect feature is supported on the device
- false - The ClockSelect feature is not supported on the device

Remarks

None.

5.6.20.5.6.12 PLIB_RTCC_ExistsEnableControl Function**C**

```
bool PLIB_RTCC_ExistsEnableControl(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_Enable](#)
- [PLIB_RTCC_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Remarks

None.

5.6.20.5.6.13 PLIB_RTCC_ExistsHalfSecond Function**C**

```
bool PLIB_RTCC_ExistsHalfSecond(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the HalfSecond feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_HalfSecondStatusBit

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The HalfSecond feature is supported on the device
- false - The HalfSecond feature is not supported on the device

Remarks

None.

5.6.20.5.6.14 PLIB_RTCC_ExistsOutputControl Function**C**

```
bool PLIB_RTCC_ExistsOutputControl(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the OutputControl feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- [PLIB_RTCC_ClockOutputEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OutputControl feature is supported on the device
- false - The OutputControl feature is not supported on the device

Remarks

None.

5.6.20.5.6.15 PLIB_RTCC_ExistsOutputSelect Function**C**

```
bool PLIB_RTCC_ExistsOutputSelect(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the OutputSelect feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_SecondsClockOutputSelect
- PLIB_RTCC_AlarmPulseOutputSelect

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OutputSelect feature is supported on the device
- false - The OutputSelect feature is not supported on the device

Remarks

None.

5.6.20.5.6.16 PLIB_RTCC_ExistsRTCDate Function**C**

```
bool PLIB_RTCC_ExistsRTCDate(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the RTCDateValue feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_RTCDateGet](#)
- [PLIB_RTCC_RTCDateSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RTCDate feature is supported on the device
- false - The RTCDate feature is not supported on the device

Remarks

None.

5.6.20.5.6.17 PLIB_RTCC_ExistsRTCTime Function

C

```
bool PLIB_RTCC_ExistsRTCTime(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the RTCTimeValue feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_RTCTimeGet](#)
- [PLIB_RTCC_RTCTimeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RTCTime feature is supported on the device
- false - The RTCTime feature is not supported on the device

Remarks

None.

5.6.20.5.6.18 PLIB_RTCC_ExistsStopInIdleControl Function

C

```
bool PLIB_RTCC_ExistsStopInIdleControl(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_StopInIdleEnable](#)
- [PLIB_RTCC_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Remarks

None.

5.6.20.5.6.19 PLIB_RTCC_ExistsSynchronization Function**C**

```
bool PLIB_RTCC_ExistsSynchronization(  
    RTCC_MODULE_ID index  
);
```

Description

This function identifies whether the Synchronization feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_ValueRegistersReadSyncBit

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Synchronization feature is supported on the device
- false - The Synchronization feature is not supported on the device

Remarks

None.

5.6.20.5.6.20 PLIB_RTCC_ExistsWriteEnable Function**C**

```
bool PLIB_RTCC_ExistsWriteEnable(  
    RTCC_MODULE_ID index
```

```
);
```

Description

This function identifies whether the WriteEnable feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- [PLIB_RTCC_WriteEnable](#)
- [PLIB_RTCC_WriteDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WriteEnable feature is supported on the device
- false - The WriteEnable feature is not supported on the device

Remarks

None.

5.6.20.6 Files

Files

Name	Description
plib_rtcc.h	Real-Time Clock and Calendar (RTCC) Peripheral Library interface header for RTCC common definitions.

Description

5.6.20.6.1 plib_rtcc.h

RTCC Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the RTCC Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the RTCC peripheral.

Functions

	Name	Description
≡	PLIB_RTCC_AlarmChimeDisable	Disables the specific RTCC module's chime.
≡	PLIB_RTCC_AlarmChimeEnable	Enables the specific RTCC module's chime.
≡	PLIB_RTCC_AlarmDateGet	Returns the contents of the specific RTCC module's Alarm Date register.
≡	PLIB_RTCC_AlarmDateSet	Writes to the specific RTCC module's Alarm Date register.
≡	PLIB_RTCC_AlarmDayGet	Returns the contents of the Day bits in the specific RTCC module's Alarm Date register.
≡	PLIB_RTCC_AlarmDaySet	Writes to the specific RTCC module's Alarm Date value register.

	PLIB_RTCC_AlarmDisable	Disables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmEnable	Enables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmHourGet	Returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmHourSet	The function returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMaskModeSelect	Sets the specific RTCC module's alarm mask Configuration bits.
	PLIB_RTCC_AlarmMinuteGet	Returns the contents of Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMinuteSet	Returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMonthGet	Returns the contents of the Month bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmMonthSet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmPulseInitialGet	Returns the state of the initial alarm pulse.
	PLIB_RTCC_AlarmPulseInitialSet	Enables the determination of the initial alarm pulse.
	PLIB_RTCC_AlarmRepeatCountGet	Reads the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmRepeatCountSet	Sets the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmSecondGet	Returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmSecondSet	Returns the contents of Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmTimeGet	Returns the contents of the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmTimeSet	Writes to the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmValueRegisterPointer	Sets the specific RTCC module's Alarm register pointer.
	PLIB_RTCC_AlarmWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmWeekDaySet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_ClockOutputDisable	Disables the specific RTCC module's output pin.
	PLIB_RTCC_ClockOutputEnable	Enables the specific RTCC module's output pin.
	PLIB_RTCC_ClockRunningStatus	Provides the status of the RTCC clock.
	PLIB_RTCC_ClockSourceSelect	Selects the clock source for the RTCC module.
	PLIB_RTCC_Disable	Disables the specific RTCC module on the device.
	PLIB_RTCC_DriftCalibrateGet	Reads the specific RTCC module's drift calibration bits.
	PLIB_RTCC_DriftCalibrateSet	Sets the specific RTCC module's drift calibration bits.
	PLIB_RTCC_Enable	Enables the specific RTCC module on the device.
	PLIB_RTCC_ExistsAlarmChimeControl	Identifies whether the AlarmChimeControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmControl	Identifies whether the AlarmControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmDate	Identifies whether the AlarmDate feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmMaskControl	Identifies whether the AlarmMaskControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmPulseInitial	Identifies whether the AlarmPulseInitial feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmRepeatControl	Identifies whether the AlarmRepeatControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmSynchronization	Identifies whether the AlarmSynchronization feature exists on the RTCC module.

	PLIB_RTCC_ExistsAlarmTime	Identifies whether the AlarmTime feature exists on the RTCC module.
	PLIB_RTCC_ExistsCalibration	Identifies whether the Calibration feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockRunning	Identifies whether the ClockRunning feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockSelect	Identifies whether the ClockSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsHalfSecond	Identifies whether the HalfSecond feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputControl	Identifies whether the OutputControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputSelect	Identifies whether the OutputSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCDate	Identifies whether the RTCDateValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCTime	Identifies whether the RTCTimeValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the RTCC module.
	PLIB_RTCC_ExistsSynchronization	Identifies whether the Synchronization feature exists on the RTCC module.
	PLIB_RTCC_ExistsWriteEnable	Identifies whether the WriteEnable feature exists on the RTCC module.
	PLIB_RTCC_HalfSecondStatusGet	The function returns the half second status bit.
	PLIB_RTCC_OutputSelect	Selects which signal will be presented on the RTCC pin
	PLIB_RTCC_RTCDateGet	Returns the contents of the specific RTCC module's Date register.
	PLIB_RTCC_RTCDateSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCDayGet	Returns the contents of the Days bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCHourGet	Returns the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCHourSet	Writes the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteGet	Returns the contents of the Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteSet	Writes the contents of Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMonthGet	Returns the contents of the Months bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCMonthSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCSecondGet	The function returns the contents of the Seconds bits in the specific RTCC device's Time register.
	PLIB_RTCC_RTCSecondSet	Writes the contents of Seconds bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeGet	Returns the contents of the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeSet	Writes to the specific RTCC module's Time register.
	PLIB_RTCC_RTCWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCWeekDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearGet	Returns the contents of the Year bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_StopInIdleDisable	Continues normal RTCC operation when the device enters Idle mode.
	PLIB_RTCC_StopInIdleEnable	Disables access to the RTCC module by the Peripheral Bus Clock (PBCLK) when the CPU enters Idle mode.

	PLIB_RTCC_WriteDisable	Disables writing to the specific RTCC module's value registers.
	PLIB_RTCC_WriteEnable	Enables writing to the specific RTCC module's value registers.

Company

Microchip Technology Inc.

5.6.21 System Bus Peripheral Library

5.6.21.1 Introduction

System Bus Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the System Bus on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The System Bus is a multi-layer crossbar switching matrix that connects initiators to targets. An initiator is a device or peripheral that initiates a data transfer to or from a target. Examples of initiators include the CPU, DMA and Ethernet controller. A target is a memory, bus or peripheral that sends or receives data to/from an initiator.

The System Bus allows the system programmer to control which initiators can access each target by defining permission groups for each initiator. Each target has one or more regions (physical address spaces) that can be individually programmed to allow access to different permission groups. By setting initiator permission groups and target region read/write permissions, the system programmer can allow or deny task or program access to system resources.

5.6.21.2 Release Notes

MPLAB Harmony Version: v0.70b **System Bus Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.21.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.21.4 Using the Library

This topic describes the basic architecture of the System Bus Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_sb.h](#)

The interface to the System Bus Peripheral library is defined in the "[plib_sb.h](#)" header file. This file is included by the "peripheral.h" file. Any C language source (.c) file that uses the System Bus Peripheral library should include "peripheral.h".

Library File:

The System Bus Peripheral library archive (.a) file installed with MPLAB Harmony.

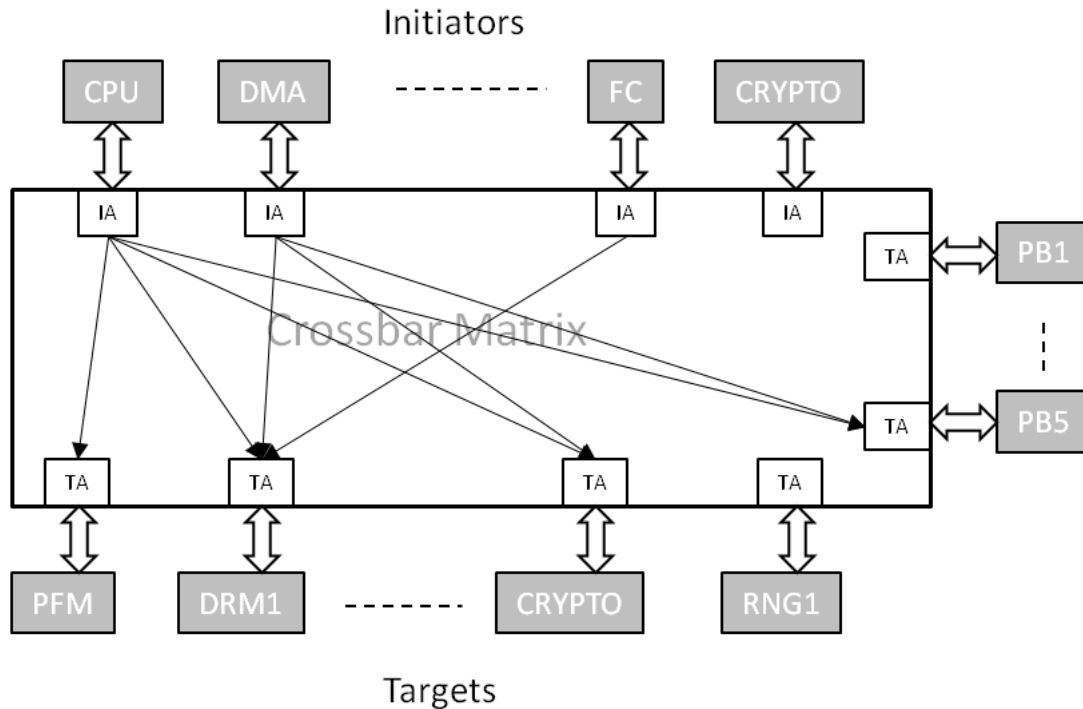
Please refer to the [MPLAB Harmony Overview](#) for how the peripheral interacts with the framework.

5.6.21.4.1 Hardware Abstraction Model

This library provides the low-level abstraction of the System Bus on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

System Bus Software Abstraction Block Diagram



Hardware Abstraction Model

The System Bus Peripheral Library provides interface routines to initialize the bus matrix such that:

- Initiator permission groups are assigned
- Target regions are defined and given read/write access permissions
- Errors are identified, logged and cleared when a permission group violation occurs
- CPU and DMA priorities are set

5.6.21.4.2 How the Library Works

The System Bus is a crossbar interconnect matrix that provides for simultaneous data transfer between the CPU, memory, and various peripherals on the PIC32 device. In addition, it provides a protection mechanism that allows trusted code to define and restrict access to memory regions and peripherals. A typical application would be a secure bootloader that prevents a third party or untrusted application from accessing regions of Flash memory that contain privileged code or information. The protection mechanism works by dividing the target physical address space into regions, each of which may be accessed by one or more initiators. Individual targets may have one or more regions. Each region can be individually programmed to allow read or write access to one or more permission groups. Each initiator is assigned to a permission group.

This library provides interface routines to initialize the System Bus and initiator permission groups. On reset or NMI, the CPU is assigned to permission group 0 (full access).

Note: Please refer to the PIC32 Family Reference Manual and specific device data sheet for information on target region mapping and initiator permission groups for a specific device.

5.6.21.4.2.1 Initiator Initialization

Initializing the initiators for the System Bus is a two-step process:

- Set CPU and DMA priorities (if needed)
- Assign permission groups to each initiator

Priority assignments for CPU and DMA

By default, the System Bus uses a least-recently-served (LRS) priority arbitration scheme for most initiators, including the CPU and DMA. For applications that require low latency, the user may set the CPU and/or DMA arbitration to fixed high priority. For the CPU, fixed high priority means that the CPU gets arbitration preference while servicing an interrupt over all initiators using LRS arbitration.

Priority is set using the `PLIB_SB_CPUPrioritySet()` and `PLIB_SB_DMAPrioritySet()` library functions. These functions write to a register that must be unlocked first. The `PLIB_SB_PriorityUnlock()` is provided for this purpose.

Initiator permission groups

At reset, all initiators default to permission group 0. Each target region can be individually programmed to allow read or write access to any or all permission groups. By assigning permission groups to initiators and restricting target region read/write access based on permission group, it is possible to restrict initiator access to target regions during normal operations.

Permission groups are assigned using the library function `PLIB_SB_InitPermGrpSet()`.

5.6.21.4.2.2 Target Initialization

Each System Bus target resides in a physical address space that contains one or more regions. Regions are used to provide separate access permissions for different areas within a target address space. Region 0 for any target is defined as the entire address space for that target. At reset, region 0 for all targets is accessible by any initiator with group 0 permissions.

Target regions are defined by base (physical) address and size. Read and write permissions are programmed individually for each region. The number of regions for each target varies, depending on the target. Some regions have fixed addresses, sizes and/or permissions. For example, region 0 for all targets encompasses the entire address space for the target, so the address and size are pre-programmed. Similarly, write permission for PFM is disabled for all initiators except the Flash controller.

Target region initialization is a four step process:

1. Set the base address of a region using `PLIB_SB_PGRegionAddrSet()`. This function takes an enumerated value for the target region. Only valid target regions should be enumerated for any specific device.
2. Set the size of the region using `PLIB_SB_PGRegionSizeSet()`.
3. Set the read permissions using `PLIB_SB_PGRegionReadPermSet()`.
4. Set the write permissions using `PLIB_SB_PGRegionWritePermSet()`.

Note: Please refer to the Memory Organization Family Reference Manual section and the specific device data sheet for information on target region mapping and initiator permission groups.

5.6.21.4.2.3 PGV Error Handling

A permission group violation (PGV) occurs when an initiator attempts to access a target without the proper permissions. Any permission group violation will trigger an interrupt to the CPU. If enabled, the System Bus will report a PGV to the error log registers in the System Bus. The PGV interrupt handler can then use this library to retrieve information about the violation and act accordingly.

By default, error reporting is disabled in the System Bus and must be enabled during the startup or boot code. Errors are divided into two categories: primary and secondary. Primary errors are errors arising from secure accesses, such as trying to access a target region without the proper permissions. Secondary errors are errors arising from non-secure accesses, such as trying to access a memory location outside of a target region.

PIC32 devices support logging of primary errors only. Primary error logging is enabled using the library function `PLIB_SB_PGVErrorReportPrimaryEnable()`. Primary error reporting can be disabled using the library function

[PLIB_SB_PGVErrorReportPrimaryDisable\(\)](#). Error logging must be enabled or disabled separately for each target.

Once a permission group violation error is logged, the user can use this library to retrieve information from the System Bus error log registers. Each target has its own set of error log registers. The error information available and the library API used to retrieve it are shown in the table below.

Error Log Data	System Bus Peripheral Library Function
Has the specified target reported an error?	PLIB_SB_PGVErrorStatus()
Has the specified target reported more than one error since last cleared?	PLIB_SB_PGVErrorMulti()
Type of error.	PLIB_SB_PGVErrorCode()
Initiator ID.	PLIB_SB_PGVErrorInitiatorID()
OCP command code of the transaction.	PLIB_SB_PGVErrorCommandCode()
The region number of the specified target accessed during the violation.	PLIB_SB_PGVErrorRegion()
Permission group of initiator that caused the violation.	PLIB_SB_PGVErrorPermissionGroup()

Once an error is logged, it needs to be cleared by software before any data for a subsequent violation can be logged (for the same target). This is typically done in the exception handler that reads the data. Which function is used depends on whether a single or multiple error violation occurred. Single violations are cleared with [PLIB_SB_PGVErrorClearSingle\(\)](#), and multiple violations are cleared with [PLIB_SB_PGVErrorClearMulti\(\)](#).

5.6.21.5 Configuring the Library

The library is configured for the supported System Bus module when the processor is chosen in the MPLAB IDE.

5.6.21.6 Library Interface

Data Types and Constants

	Name	Description
	PLIB_SB_ARB_POLICY	This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.
	PLIB_SB_ERROR	Lists the possible SB Transaction Error Codes
	PLIB_SB_INIT_ID	Lists the possible SB Initiator IDs.
	PLIB_SB_INIT_PG	Lists the possible Initiator permission groups
	PLIB_SB_OCP_CMD_CODE	Lists the possible OCP Command codes.
	PLIB_SB_PG_INITIATOR	Lists the possible permission group Initiators
	PLIB_SB_REGION_PG	This enumeration lists the possible permission groups assigned to a region for read and/or write access.
	PLIB_SB_TGT_ID	Lists the possible SB Target IDs
	PLIB_SB_TGT_REGION	Lists the programmable target regions
	SB_MODULE_ID	Lists the possible Module IDs for the system bus.

Feature Existence Functions

	Name	Description
	PLIB_SB_ExistsCPUPriority	Identifies whether the CPUPriority feature exists on the SB module

	PLIB_SB_ExistsDMAPriority	Identifies whether the DMAPriority feature exists on the SB module
	PLIB_SB_ExistsInitPermGrp	Identifies whether the InitPermGrp feature exists on the SB module
	PLIB_SB_ExistsPGRegAddr	Identifies whether the PGRegAddr feature exists on the SB module
	PLIB_SB_ExistsPGRegRdPerm	Identifies whether the PGRegRdPerm feature exists on the SB module
	PLIB_SB_ExistsPGRegSize	Identifies whether the PGRegSize feature exists on the SB module
	PLIB_SB_ExistsPGRegWrPerm	Identifies whether the PGRegWrPerm feature exists on the SB module
	PLIB_SB_ExistsPGVErrClear	Identifies whether the PGVErrClear feature exists on the SB module
	PLIB_SB_ExistsPGVErrClrMulti	Identifies whether the PGVErrClrMulti feature exists on the SB module
	PLIB_SB_ExistsPGVErrClrSingle	Identifies whether the PGVErrClrSingle feature exists on the SB module
	PLIB_SB_ExistsPGVErrCmdCode	Identifies whether the PGVErrCmdCode feature exists on the SB module
	PLIB_SB_ExistsPGVErrInitID	Identifies whether the PGVErrInitID feature exists on the SB module
	PLIB_SB_ExistsPGVErrPG	Identifies whether the PGVErrPG feature exists on the SB module
	PLIB_SB_ExistsPGVErrRegion	Identifies whether the PGVErrRegion feature exists on the SB module
	PLIB_SB_ExistsPGVErrRptPri	Identifies whether the PGVErrRptPri feature exists on the SB module
	PLIB_SB_ExistsPGVErrStatus	Identifies whether the PGVErrStatus feature exists on the SB module

Other Functions

	Name	Description
	PLIB_SB_CPUPrioritySet	Sets the CPU arbitration policy to SRAM when servicing an interrupt
	PLIB_SB_DMAPrioritySet	Sets the DMA arbitration policy
	PLIB_SB_InitPermGrpSet	Sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.

PGV Error Functions

	Name	Description
	PLIB_SB_PGVErrorClearMulti	Clears multiple permission group errors for the specified target.
	PLIB_SB_PGVErrorClearSingle	Clears a single permission group error for the specified target.
	PLIB_SB_PGVErrorCode	Returns a value corresponding to the type of error logged for the specified target.
	PLIB_SB_PGVErrorCommandCode	Returns the OCP command code of the transaction that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorInitiatorID	Returns the ID of the Initiator that caused the protection violation
	PLIB_SB_PGVErrorLogClearMulti	Clears a multiple protection group violations error from the specified target error log register.
	PLIB_SB_PGVErrorLogClearSingle	Clears a single protection group violation error from the specified target error log register.
	PLIB_SB_PGVErrorMulti	Indicates if more than one permission group violation has occurred since last cleared.
	PLIB_SB_PGVErrorPermissionGroup	Returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorRegion	Returns the number of the protection region in the specified target address space that caused the protection violation.
	PLIB_SB_PGVErrorReportPrimaryDisable	Disables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorReportPrimaryEnable	Enables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorStatus	Identifies whether a permission group violation has been reported for the specified target.

Target Initialization Functions

	Name	Description
	PLIB_SB_PGRegionAddrGet	Returns the base address for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionAddrSet	Sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermClear	Clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermSet	Sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionSizeGet	Returns the size for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionSizeSet	Sets the size for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermClear	Clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermSet	Sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Description

This section describes the Application Programming Interface (API) functions of the System Bus Peripheral Library.

Refer to each section for a detailed description.

5.6.21.6.1 Target Initialization Functions**5.6.21.6.1.1 PLIB_SB_PGRegionAddrGet Function****C**

```
uint32_t PLIB_SB_PGRegionAddrGet(
    SB_MODULE_ID index,
    PLIB_SB_TGT_REGION region
);
```

Description

This function returns the base address for a permission group region within a target's physical address space.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the address is returned.

Returns

uint32_t - The base address of the region.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
uint32_t T1R6BaseAddress;
T1R6BaseAddress = PLIB_SB_PGRegionAddrGet(SB_ID_1, PLIB_T1_REGION_6);
```

5.6.21.6.1.2 PLIB_SB_PGRegionAddrSet Function**C**

```
void PLIB_SB_PGRegionAddrSet(
    SB_MODULE_ID index,
    PLIB_SB_TGT_REGION region,
    uint32_t phys_addr
);
```

Description

This function sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the base address is set.
phys_addr	The base address of the region. Must be aligned to the intended size of the region.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Region 0 of all targets have a defined start address which may not be changed. Some regions (such as those containing peripheral SFRs) are fixed and may not be changed. Please refer to the specific device data sheet for details.

Example

```
// Set up two regions in PFM with full read and write permission
#define REGION_5_BASE_ADDR 0x1D000000
#define REGION_6_BASE_ADDR (REGION5_BASE_ADDR + (32*1024))
#define REGION_5_SIZE      0x06 // 32KB
#define REGION_6_SIZE      0x05 // 16KB
#define FULL_PERM          (REGION_PG_0 | REGION_PG_1 | REGION_PG_2 | REGION_PG_3)

PLIB_SB_PGRegionAddrSet(SB_ID_1, PLIB_SB_T1_REGION_5, REGION_5_BASE_ADDR);
PLIB_SB_PGRegionAddrSet(SB_ID_1, PLIB_SB_T1_REGION_6, REGION_6_BASE_ADDR);

PLIB_SB_PGRegionSizeSet(SB_ID_1, PLIB_SB_T1_REGION_5, REGION_5_SIZE);
PLIB_SB_PGRegionSizeSet(SB_ID_1, PLIB_SB_T1_REGION_6, REGION_6_SIZE);

PLIB_SB_PGRegionReadPermSet(SB_ID_1, PLIB_T1_REGION_5, FULL_PERM);
PLIB_SB_PGRegionReadPermSet(SB_ID_1, PLIB_T1_REGION_6, FULL_PERM);

PLIB_SB_PGRegionWritePermSet(SB_ID_1, PLIB_T1_REGION_5, FULL_PERM);
PLIB_SB_PGRegionWritePermSet(SB_ID_1, PLIB_T1_REGION_6, FULL_PERM);
```

5.6.21.6.1.3 PLIB_SB_PGRegionReadPermClear Function

C

```
void PLIB_SB_PGRegionReadPermClear(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_REGION region,  
    PLIB_SB_REGION_PG readPerm  
);
```

Description

This function clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
readPerm	Bitwise OR of the groups given read permission for the region.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is read permission for all groups.

Example

See code example for [PLIB_SB_PGRegionAddrSet\(\)](#).

5.6.21.6.1.4 PLIB_SB_PGRegionReadPermSet Function

C

```
void PLIB_SB_PGRegionReadPermSet(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_REGION region,  
    PLIB_SB_REGION_PG readPerm  
);
```

Description

This function sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

region	The region number within the target's address space to be given permissions. Not all regions are programmable.
readPerm	Bitwise OR of the groups given read permission for the region.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is read permission for all groups.

Example

See code example for [PLIB_SB_PGRegionAddrSet\(\)](#).

5.6.21.6.1.5 PLIB_SB_PGRegionSizeGet Function

C

```
uint32_t PLIB_SB_PGRegionSizeGet(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_REGION region  
);
```

Description

This function returns the size for a permission group region within a target's physical address space. Not all regions are programmable.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the size is to be set. Not all regions are programmable.

Returns

The size of the region.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Region 0 of all targets encompasses the entire address range of the target, and may not be changed. Some regions (such as those containing peripheral SFRs) may not be changed. Please refer to the specific device data sheet for details.

Example

```
uint32_t T1R6Size;  
T1R6Size = PLIB_SB_PGRegionSizeGet(SB_ID_1, PLIB_T1_REGION_6);
```

5.6.21.6.1.6 PLIB_SB_PGRegionSizeSet Function

C

```
void PLIB_SB_PGRegionSizeSet(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_REGION region,  
    uint32_t size  
);
```

Description

This function sets the size for a permission group region within a target's physical address space. Not all regions are programmable.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the size is to be set. Not all regions are programmable.
size	The actual size of the region being programmed is calculated as
follows	$2^{(size-1)} \times 1024$ bytes

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Region 0 of all targets encompasses the entire address range of the target, and may not be changed. Some regions (such as those containing peripheral SFRs) may not be changed. Please refer to the specific device data sheet for details.

Example

See code example for [PLIB_SB_PGRegionAddrSet\(\)](#).

5.6.21.6.1.7 PLIB_SB_PGRegionWritePermClear Function**C**

```
void PLIB_SB_PGRegionWritePermClear(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_REGION region,  
    PLIB_SB_REGION_PG writePerm  
);
```

Description

This function clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
writePerm	Bitwise OR of the groups given read permission for the region.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is write permission for all groups. Some regions may not be programmable by all initiators (flash, for example). Please refer to the specific device data sheet for details.

Example

See code example for [PLIB_SB_PGRegionAddrSet\(\)](#).

5.6.21.6.1.8 PLIB_SB_PGRegionWritePermSet Function**C**

```
void PLIB_SB_PGRegionWritePermSet(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_REGION region,  
    PLIB_SB_REGION_PG writePerm  
);
```

Description

This function sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
writePerm	Bitwise OR of the groups given read permission for the region.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is write permission for all groups. Some regions may not be programmable by all initiators (flash, for example). Please refer to the specific device data sheet for details.

Example

See code example for [PLIB_SB_PGRegionAddrSet\(\)](#).

5.6.21.6.2 PGV Error Functions**5.6.21.6.2.1 PLIB_SB_PGVErrorClearMulti Function****C**

```
bool PLIB_SB_PGVErrorClearMulti(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function clears multiple permission group errors for the specified target.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Returns

mult - Returns the value of the CLEAR bit in the SBTxECLRM register for the specified target. The act of reading this bit clears the error.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
bool mult;  
mult = PLIB_SB_PGVErrClearMulti(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.2 PLIB_SB_PGVErrClearSingle Function**C**

```
bool PLIB_SB_PGVErrClearSingle(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function clears a single permission group error for the specified target.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Returns

sing - Returns the value of the CLEAR bit in the SBTxECLRM register for the specified target. The act of reading this bit clears the error.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
bool sing;  
sing = PLIB_SB_PGVErrClearSingle(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.3 PLIB_SB_PGVErrrorCode Function

C

```
PLIB_SB_ERROR PLIB_SB_PGVErrrorCode(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function returns a value corresponding to the type of error logged for the specified target.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be read.

Returns

[PLIB_SB_ERROR](#) enumeration representing the type of SB error logged states.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
PLIB_SB_ERROR error;  
error = PLIB_SB_ERROR PLIB_SB_PGVErrrorCode(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.4 PLIB_SB_PGVErrrorCommandCode Function

C

```
PLIB_SB_OCP_CMD_CODE PLIB_SB_PGVErrrorCommandCode(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function returns the OCP command code of the transaction that caused the protection violation for the specified target.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Returns

OCP command code

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
PLIB_SB_OCP_CMD_CODE commandCode;  
commandCode = PLIB_SB_PGVErrorCommandCode(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.5 PLIB_SB_PGVErrorInitiatorID Function**C**

```
PLIB_SB_INIT_ID PLIB_SB_PGVErrorInitiatorID(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function returns the ID of the Initiator that caused the protection violation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Returns

An enumerated value representing the ID of the initiator that caused the protection violation.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
PLIB_SB_INIT_ID id;  
id = PLIB_SB_PGVErrorInitiatorID(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.6 PLIB_SB_PGVErrorLogClearMulti Function**C**

```
void PLIB_SB_PGVErrorLogClearMulti(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function clears a multiple protection group violations error from the specified target error log register. Multiple errors are cleared by writing a '1' to both the MULTI and CODE fields of the SBTxELOG1 register for the specified target.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be cleared.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
PLIB_SB_PGVErrorLogClearMulti(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.7 PLIB_SB_PGVErrorLogClearSingle Function**C**

```
void PLIB_SB_PGVErrorLogClearSingle(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function clears a single protection group violation error from the specified target error log register. Single errors are cleared by writing a '1' to the CODE field of the SBTxELOG1 register for the specified target.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be cleared.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
PLIB_SB_PGVErrorLogClearSingle(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.8 PLIB_SB_PGVErrorMulti Function**C**

```
bool PLIB_SB_PGVErrorMulti(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function indicates if more than one permission group violation has occurred since last cleared.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be read.

Returns

true - Multiple permission group violations. false - Single or no permission group violations.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
bool multiPGV;  
multiPGV = PLIB_SB_PGVErrorMulti(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.9 PLIB_SB_PGVErrorPermissionGroup Function**C**

```
int PLIB_SB_PGVErrorPermissionGroup(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
int pg;  
pg = PLIB_SB_PGVErrorPermissionGroup(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.10 PLIB_SB_PGVErrorRegion Function**C**

```
int PLIB_SB_PGVErrorRegion(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function returns the number of the protection region in the specified target address space that caused the protection violation. Note that if there are no other region matches, region 0 (the default region that spans the entire target address space) will always match, and this function will return 0.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Returns

Region number in target address space or -1 on unrecognized target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
int region;  
region = PLIB_SB_PGVErrorRegion(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.11 PLIB_SB_PGVErrorReportPrimaryDisable Function**C**

```
void PLIB_SB_PGVErrorReportPrimaryDisable(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function disables primary permission group error reporting for the specified target to the SB flag register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Reporting of primary errors is disabled at reset.

Example

```
PLIB_SB_PGVErrorReportPrimaryDisable(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.12 PLIB_SB_PGVErrorReportPrimaryEnable Function**C**

```
void PLIB_SB_PGVErrorReportPrimaryEnable(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function enables primary permission group error reporting for the specified target to the SB flag register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Reporting of primary errors is disabled at reset.

Example

```
PLIB_SB_PGVErrorReportPrimaryEnable(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.2.13 PLIB_SB_PGVErrorStatus Function**C**

```
bool PLIB_SB_PGVErrorStatus(  
    SB_MODULE_ID index,  
    PLIB_SB_TGT_ID target  
);
```

Description

This function identifies whether a permission group violation has been reported for the specified target.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.
target	Target to be checked.

Returns

true - Target is reporting a permission group violation. false - Target is not reporting a permission group violation.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
bool sbPgv;  
sbPgv = PLIB_SB_PGVErrorStatus(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

5.6.21.6.3 Data Types and Constants

5.6.21.6.3.1 PLIB_SB_ARB_POLICY Enumeration

C

```
typedef enum {
    PRIORITY_LRS,
    PRIORITY_HI
} PLIB_SB_ARB_POLICY;
```

Description

SB Arbitration Policy

This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.

Members

Members	Description
PRIORITY_LRS	Least Recently Serviced Arbitration
PRIORITY_HI	High Priority

5.6.21.6.3.2 PLIB_SB_ERROR Enumeration

C

```
typedef enum {
    PLIB_SB_ERROR_NONE,
    PLIB_SB_ERROR_PGV
} PLIB_SB_ERROR;
```

Description

PG Error Code

This enumeration lists the possible transaction error codes for the SB.

Members

Members	Description
PLIB_SB_ERROR_NONE	No Error
PLIB_SB_ERROR_PGV	Permission Group Violation

5.6.21.6.3.3 PLIB_SB_INIT_ID Enumeration

C

```
typedef enum {
    PLIB_SB_INIT_ID_NONE,
    PLIB_SB_INIT_ID_CPU_LRS,
    PLIB_SB_INIT_ID_CPU_HI,
    PLIB_SB_INIT_ID_DMA1_RD_LRS,
    PLIB_SB_INIT_ID_DMA1_RD_HI,
    PLIB_SB_INIT_ID_DMA1_WR_LRS,
    PLIB_SB_INIT_ID_DMA1_WR_HI,
    PLIB_SB_INIT_ID_USB1,
    PLIB_SB_INIT_ID_ETH1_RD,
    PLIB_SB_INIT_ID_ETH1_WR,
    PLIB_SB_INIT_ID_CAN1,
    PLIB_SB_INIT_ID_CAN2,
    PLIB_SB_INIT_ID_SQI1,
    PLIB_SB_INIT_ID_FLASH_CTL,
    PLIB_SB_INIT_ID_CRYPT0
} PLIB_SB_INIT_ID;
```

Description

SB Initiator ID

This enumeration lists the possible SB Initiator IDs. This ID is used for self-reporting and error logging. LRS and HI are SB arbitration schemes set by the soft configuration register CFGCON.

5.6.21.6.3.4 PLIB_SB_INIT_PG Enumeration

C

```
typedef enum {
    PLIB_SB_INIT_PG_0,
    PLIB_SB_INIT_PG_1,
    PLIB_SB_INIT_PG_2,
    PLIB_SB_INIT_PG_3
} PLIB_SB_INIT_PG;
```

Description

SB Initiator Permission Groups

This enumeration lists the possible initiator permission groups for the SB.

Members

Members	Description
PLIB_SB_INIT_PG_0	Privilege Group 0
PLIB_SB_INIT_PG_1	Privilege Group 1
PLIB_SB_INIT_PG_2	Privilege Group 2
PLIB_SB_INIT_PG_3	Privilege Group 3

Remarks

These values are used to program the CFGPG soft configuration register, which is not part of the SB. This should be done by the boot code to set up the desired initiator permissions prior to programming the SB.

5.6.21.6.3.5 PLIB_SB_OCP_CMD_CODE Enumeration

C

```
typedef enum {
    PLIB_SB_OCP_CMD_IDLE,
    PLIB_SB_OCP_CMD_WRITE,
    PLIB_SB_OCP_CMD_READ,
    PLIB_SB_OCP_CMD_READEX,
    PLIB_SB_OCP_CMD_WRITE_NON_POST,
    PLIB_SB_OCP_CMD_BROADCAST
} PLIB_SB_OCP_CMD_CODE;
```

Description

OCF Command Codes

This enumeration lists the possible OCP command codes. An OCP command code is logged when a transaction violation occurs. The command code of the offending command can then be read.

5.6.21.6.3.6 PLIB_SB_PG_INITIATOR Enumeration

C

```
typedef enum {
    PLIB_SB_PG_INITIATOR_CPU,
```

```

    PLIB_SB_PG_INITIATOR_DMA1,
    PLIB_SB_PG_INITIATOR_USB1,
    PLIB_SB_PG_INITIATOR_CAN1,
    PLIB_SB_PG_INITIATOR_CAN2,
    PLIB_SB_PG_INITIATOR_ETH1,
    PLIB_SB_PG_INITIATOR_SQI1,
    PLIB_SB_PG_INITIATOR_FLASH_CTL,
    PLIB_SB_PG_INITIATOR_CRYPTO
} PLIB_SB_PG_INITIATOR;

```

Description

SB Initators

This enumeration lists the possible permission group Initiators

5.6.21.6.3.7 PLIB_SB_REGION_PG Enumeration

C

```

typedef enum {
    REGION_PG_0,
    REGION_PG_1,
    REGION_PG_2,
    REGION_PG_3
} PLIB_SB_REGION_PG;

```

Description

SB Region Permission Groups

Lists the possible permission groups assigned to a region for read and/or write access.

Members

Members	Description
REGION_PG_0	Privilege Group 0 has read/write permission
REGION_PG_1	Privilege Group 1 has read/write permission
REGION_PG_2	Privilege Group 2 has read/write permission
REGION_PG_3	Privilege Group 3 has read/write permission

5.6.21.6.3.8 PLIB_SB_TGT_ID Enumeration

C

```

typedef enum {
    PLIB_SB_TGT_ID_T0,
    PLIB_SB_TGT_ID_T1,
    PLIB_SB_TGT_ID_T2,
    PLIB_SB_TGT_ID_T3,
    PLIB_SB_TGT_ID_T4,
    PLIB_SB_TGT_ID_T5,
    PLIB_SB_TGT_ID_T6,
    PLIB_SB_TGT_ID_T7,
    PLIB_SB_TGT_ID_T8,
    PLIB_SB_TGT_ID_T9,
    PLIB_SB_TGT_ID_T10,
    PLIB_SB_TGT_ID_T11,
    PLIB_SB_TGT_ID_T12,
    PLIB_SB_TGT_ID_T13
} PLIB_SB_TGT_ID;

```

Description

SB Target ID

This enumeration lists the possible SB Target IDs

Members

Members	Description
PLIB_SB_TGT_ID_T0	System Bus
PLIB_SB_TGT_ID_T1	Prefetch Module
PLIB_SB_TGT_ID_T2	Data RAM 1
PLIB_SB_TGT_ID_T3	Data RAM 2
PLIB_SB_TGT_ID_T4	External Bus Interface
PLIB_SB_TGT_ID_T5	Peripheral Bus 1
PLIB_SB_TGT_ID_T6	Peripheral Bus 2
PLIB_SB_TGT_ID_T7	Peripheral Bus 3
PLIB_SB_TGT_ID_T8	Peripheral Bus 4
PLIB_SB_TGT_ID_T9	Peripheral Bus 5
PLIB_SB_TGT_ID_T10	USB
PLIB_SB_TGT_ID_T11	Serial Quad Interface
PLIB_SB_TGT_ID_T12	Crypto
PLIB_SB_TGT_ID_T13	Random Number Generator

5.6.21.6.3.9 PLIB_SB_TGT_REGION Enumeration

C

```
typedef enum {
    PLIB_SB_T0_REGION_0,
    PLIB_SB_T0_REGION_1,
    PLIB_SB_T1_REGION_0,
    PLIB_SB_T1_REGION_1,
    PLIB_SB_T1_REGION_2,
    PLIB_SB_T1_REGION_3,
    PLIB_SB_T1_REGION_4,
    PLIB_SB_T1_REGION_5,
    PLIB_SB_T1_REGION_6,
    PLIB_SB_T1_REGION_7,
    PLIB_SB_T1_REGION_8,
    PLIB_SB_T2_REGION_0,
    PLIB_SB_T2_REGION_1,
    PLIB_SB_T2_REGION_2,
    PLIB_SB_T3_REGION_0,
    PLIB_SB_T3_REGION_1,
    PLIB_SB_T3_REGION_2,
    PLIB_SB_T4_REGION_0,
    PLIB_SB_T4_REGION_1,
    PLIB_SB_T4_REGION_2,
    PLIB_SB_T5_REGION_0,
    PLIB_SB_T5_REGION_1,
    PLIB_SB_T5_REGION_2,
    PLIB_SB_T6_REGION_0,
    PLIB_SB_T6_REGION_1,
    PLIB_SB_T7_REGION_0,
    PLIB_SB_T7_REGION_1,
    PLIB_SB_T8_REGION_0,
    PLIB_SB_T8_REGION_1,
    PLIB_SB_T9_REGION_0,
    PLIB_SB_T9_REGION_1,
    PLIB_SB_T10_REGION_0,
    PLIB_SB_T11_REGION_0,
    PLIB_SB_T11_REGION_1,
    PLIB_SB_T12_REGION_0,
```



```

PLIB_SB_T13_REGION_0
} PLIB_SB_TGT_REGION;

```

Description

Regions

This enumeration lists the programmable SB target regions.

Members

Members	Description
PLIB_SB_T0_REGION_0	System Bus Region 0
PLIB_SB_T0_REGION_1	System Bus Region 1
PLIB_SB_T1_REGION_0	Prefetch Module Region 0
PLIB_SB_T1_REGION_1	Prefetch Module Region 1
PLIB_SB_T1_REGION_2	Prefetch Module Region 2
PLIB_SB_T1_REGION_3	Prefetch Module Region 3
PLIB_SB_T1_REGION_4	Prefetch Module Region 4
PLIB_SB_T1_REGION_5	Prefetch Module Region 5
PLIB_SB_T1_REGION_6	Prefetch Module Region 6
PLIB_SB_T1_REGION_7	Prefetch Module Region 7
PLIB_SB_T1_REGION_8	Prefetch Module Region 8
PLIB_SB_T2_REGION_0	Data RAM 1 Region 0
PLIB_SB_T2_REGION_1	Data RAM 1 Region 1
PLIB_SB_T2_REGION_2	Data RAM 1 Region 2
PLIB_SB_T3_REGION_0	Data RAM 2 Region 0
PLIB_SB_T3_REGION_1	Data RAM 2 Region 1
PLIB_SB_T3_REGION_2	Data RAM 2 Region 2
PLIB_SB_T4_REGION_0	External Bus Interface Region 0
PLIB_SB_T4_REGION_1	External Bus Interface Region 1
PLIB_SB_T4_REGION_2	External Bus Interface Region 2
PLIB_SB_T5_REGION_0	Peripheral Bus 1 Region 0
PLIB_SB_T5_REGION_1	Peripheral Bus 1 Region 1
PLIB_SB_T5_REGION_2	Peripheral Bus 1 Region 2
PLIB_SB_T6_REGION_0	Peripheral Bus 2 Region 0
PLIB_SB_T6_REGION_1	Peripheral Bus 2 Region 1
PLIB_SB_T7_REGION_0	Peripheral Bus 3 Region 0
PLIB_SB_T7_REGION_1	Peripheral Bus 3 Region 1
PLIB_SB_T8_REGION_0	Peripheral Bus 4 Region 0
PLIB_SB_T8_REGION_1	Peripheral Bus 4 Region 1
PLIB_SB_T9_REGION_0	Peripheral Bus 5 Region 0
PLIB_SB_T9_REGION_1	Peripheral Bus 5 Region 1
PLIB_SB_T10_REGION_0	USB Region 0
PLIB_SB_T11_REGION_0	Serial Quad Interface Region 0
PLIB_SB_T11_REGION_1	Serial Quad Interface Region 1
PLIB_SB_T12_REGION_0	Crypto Region 9
PLIB_SB_T13_REGION_0	Random Number Generator 0

5.6.21.6.3.10 SB_MODULE_ID Enumeration

C

```
typedef enum {
    SB_ID_1,
    SB_NUMBER_OF_MODULES
} SB_MODULE_ID;
```

Description

Module ID

This enumeration lists the possible Module IDs for the system bus.

Remarks

Refer to the data sheet to get the correct number of modules defined for the desired device.

5.6.21.6.4 Other Functions

5.6.21.6.4.1 PLIB_SB_CPUPrioritySet Function

C

```
void PLIB_SB_CPUPrioritySet(
    SB_MODULE_ID index,
    PLIB_SB_ARB_POLICY priority
);
```

Description

This function sets the CPU arbitration policy to SRAM when servicing an interrupt.

Preconditions

System must be unlocked before the priority can be set.

Parameters

Parameters	Description
priority	Use either high priority or least-recently-served algorithm.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

This function writes to the soft configuration register CFGCON, which is not part of the SB. This should be done by the boot code prior to programming the SB. Note that the system must be unlocked before the priority can be set.

Default at reset is PRIORITY_LRS.

Example

```
PLIB_SB_PriorityUnlock();
PLIB_SB_CPUPrioritySet(SB_ID_1, PRIORITY_HI);
```

5.6.21.6.4.2 PLIB_SB_DMAPrioritySet Function

C

```
void PLIB_SB_DMAPrioritySet(
```

```
    SB_MODULE_ID index,  
    PLIB_SB_ARB_POLICY priority  
);
```

Description

This function sets the DMA arbitration.

Preconditions

System must be unlocked before the priority can be set.

Parameters

Parameters	Description
priority	Use either high priority or least-recently-served algorithm.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

This function writes to the soft configuration register CFGCON, which is not part of the SB. This should be done by the boot code prior to programming the SB. Note that the system must be unlocked before the priority can be set.

Default at reset is PRIORITY_LRS.

Example

```
PLIB_SB_PriorityUnlock();  
void PLIB_SB_DMAPrioritySet(SB_ID_1, PRIORITY_HI);
```

5.6.21.6.4.3 PLIB_SB_InitPermGrpSet Function**C**

```
void PLIB_SB_InitPermGrpSet(  
    SB_MODULE_ID index,  
    PLIB_SB_PG_INITIATOR initiator,  
    PLIB_SB_INIT_PG pg  
);
```

Description

This function sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.

Preconditions

None.

Parameters

Parameters	Description
initiator	The initiator for which permission groups are assigned.
pg	The permission group(s) to which the initiator is assigned.

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

This function writes to the soft configuration register CFGPG, which is not part of the SB. Permission groups should be assigned

by the boot code prior to programming the SB.

Default permission group at reset for all initiators is 0.

After an NMI, the CPU permission group reverts to 0. All other initiator permission groups remain unchanged.

The effective CPU permission group value in debug mode is controlled by boot configuration value DBGPER[2:0]. If DBGPER denies access to the group CPU1PG selects, the effective value selects group 3.

Example

```
PLIB_SB_InitPermGrpSet(SB_ID_1, PLIB_SB_PG_INITIATOR_CPU, PLIB_SB_INIT_PG_1);
```

5.6.21.6.5 Feature Existence Functions

5.6.21.6.5.1 PLIB_SB_ExistsCPUPriority Function

```
C
bool PLIB_SB_ExistsCPUPriority(
    SB_MODULE_ID index
);
```

Description

This function identifies whether the CPUPriority feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_CPUPrioritySet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CPUPriority feature is supported on the device
- false - The CPUPriority feature is not supported on the device

Remarks

None.

5.6.21.6.5.2 PLIB_SB_ExistsDMAPriority Function

```
C
bool PLIB_SB_ExistsDMAPriority(
    SB_MODULE_ID index
);
```

Description

This function identifies whether the DMAPriority feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_DMAPrioritySet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DMAPriority feature is supported on the device
- false - The DMAPriority feature is not supported on the device

Remarks

None.

5.6.21.6.5.3 PLIB_SB_ExistsInitPermGrp Function**C**

```
bool PLIB_SB_ExistsInitPermGrp(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the InitPermGrp feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_InitPermGrpSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InitPermGrp feature is supported on the device
- false - The InitPermGrp feature is not supported on the device

Remarks

None.

5.6.21.6.5.4 PLIB_SB_ExistsPGRegAddr Function**C**

```
bool PLIB_SB_ExistsPGRegAddr(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGRegAddr feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGRegionAddrSet](#)
- [PLIB_SB_PGRegionAddrGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGRRegAddr feature is supported on the device
- false - The PGRRegAddr feature is not supported on the device

Remarks

None.

5.6.21.6.5.5 PLIB_SB_ExistsPGRRegRdPerm Function**C**

```
bool PLIB_SB_ExistsPGRRegRdPerm(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGRRegRdPerm feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGRegionReadPermSet](#)
- [PLIB_SB_PGRegionReadPermClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGRRegRdPerm feature is supported on the device
- false - The PGRRegRdPerm feature is not supported on the device

Remarks

None.

5.6.21.6.5.6 PLIB_SB_ExistsPGRRegSize Function**C**

```
bool PLIB_SB_ExistsPGRRegSize(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGRRegSize feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGRegionSizeSet](#)
- [PLIB_SB_PGRegionSizeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGRegSize feature is supported on the device
- false - The PGRegSize feature is not supported on the device

Remarks

None.

5.6.21.6.5.7 PLIB_SB_ExistsPGRegWrPerm Function

C

```
bool PLIB_SB_ExistsPGRegWrPerm(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGRegWrPerm feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGRegionWritePermSet](#)
- [PLIB_SB_PGRegionWritePermClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGRegWrPerm feature is supported on the device
- false - The PGRegWrPerm feature is not supported on the device

Remarks

None.

5.6.21.6.5.8 PLIB_SB_ExistsPGVErrClear Function

C

```
bool PLIB_SB_ExistsPGVErrClear(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGVErrClear feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGVErrorMulti](#)
- [PLIB_SB_PGVErrorCode](#)
- [PLIB_SB_PGVErrorLogClearSingle](#)
- [PLIB_SB_PGVErrorLogClearMulti](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGVErrClear feature is supported on the device
- false - The PGVErrClear feature is not supported on the device

Remarks

None.

5.6.21.6.5.9 PLIB_SB_ExistsPGVErrClrMulti Function**C**

```
bool PLIB_SB_ExistsPGVErrClrMulti(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGVErrClrMulti feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGVErrorClearMulti](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGVErrClrMulti feature is supported on the device
- false - The PGVErrClrMulti feature is not supported on the device

Remarks

None.

5.6.21.6.5.10 PLIB_SB_ExistsPGVErrClrSingle Function

C

```
bool PLIB_SB_ExistsPGVErrClrSingle(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGVErrClrSingle feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGVErrorClearSingle](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGVErrClrSingle feature is supported on the device
- false - The PGVErrClrSingle feature is not supported on the device

Remarks

None.

5.6.21.6.5.11 PLIB_SB_ExistsPGVErrCmdCode Function

C

```
bool PLIB_SB_ExistsPGVErrCmdCode(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGVErrCmdCode feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGVErrorCommandCode](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGVErrCmdCode feature is supported on the device
- false - The PGVErrCmdCode feature is not supported on the device

Remarks

None.

5.6.21.6.5.12 PLIB_SB_ExistsPGVErrInitID Function

C

```
bool PLIB_SB_ExistsPGVErrInitID(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGVErrInitID feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGVErrorInitiatorID](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGVErrInitID feature is supported on the device
- false - The PGVErrInitID feature is not supported on the device

Remarks

None.

5.6.21.6.5.13 PLIB_SB_ExistsPGVErrPG Function

C

```
bool PLIB_SB_ExistsPGVErrPG(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGVErrPG feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGVErrorPermissionGroup](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGVErrPG feature is supported on the device
- false - The PGVErrPG feature is not supported on the device

Remarks

None.

5.6.21.6.5.14 PLIB_SB_ExistsPGVErrRegion Function

C

```
bool PLIB_SB_ExistsPGVErrRegion(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGVErrRegion feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGVErrorRegion](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGVErrRegion feature is supported on the device
- false - The PGVErrRegion feature is not supported on the device

Remarks

None.

5.6.21.6.5.15 PLIB_SB_ExistsPGVErrRptPri Function

C

```
bool PLIB_SB_ExistsPGVErrRptPri(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGVErrRptPri feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGVErrorReportPrimaryEnable](#)
- [PLIB_SB_PGVErrorReportPrimaryDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGVErrRptPri feature is supported on the device
- false - The PGVErrRptPri feature is not supported on the device

Remarks

None.

5.6.21.6.5.16 PLIB_SB_ExistsPGVErrStatus Function

C

```
bool PLIB_SB_ExistsPGVErrStatus(  
    SB_MODULE_ID index  
);
```

Description

This function identifies whether the PGVErrStatus feature is available on the SB module. When this function returns true, these functions are supported on the device:

- [PLIB_SB_PGVErrorStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PGVErrStatus feature is supported on the device
- false - The PGVErrStatus feature is not supported on the device

Remarks

None.

5.6.21.7 Files

Files

Name	Description
plib_sb.h	Defines the System Bus (SB) Peripheral Library interface

Description

5.6.21.7.1 plib_sb.h

System Bus Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the system bus (SB) peripheral library (PLIB) for Microchip microcontrollers. The definitions in this file are for the SB controller module.

Functions

	Name	Description
≡	PLIB_SB_CPUPrioritySet	Sets the CPU arbitration policy to SRAM when servicing an interrupt
≡	PLIB_SB_DMAMPrioritySet	Sets the DMA arbitration policy
≡	PLIB_SB_ExistsCPUPriority	Identifies whether the CPUPriority feature exists on the SB module
≡	PLIB_SB_ExistsDMAMPriority	Identifies whether the DMAMPriority feature exists on the SB module

	PLIB_SB_ExistsInitPermGrp	Identifies whether the InitPermGrp feature exists on the SB module
	PLIB_SB_ExistsPGRegAddr	Identifies whether the PGRegAddr feature exists on the SB module
	PLIB_SB_ExistsPGRegRdPerm	Identifies whether the PGRegRdPerm feature exists on the SB module
	PLIB_SB_ExistsPGRegSize	Identifies whether the PGRegSize feature exists on the SB module
	PLIB_SB_ExistsPGRegWrPerm	Identifies whether the PGRegWrPerm feature exists on the SB module
	PLIB_SB_ExistsPGVErrClear	Identifies whether the PGVErrClear feature exists on the SB module
	PLIB_SB_ExistsPGVErrClrMulti	Identifies whether the PGVErrClrMulti feature exists on the SB module
	PLIB_SB_ExistsPGVErrClrSingle	Identifies whether the PGVErrClrSingle feature exists on the SB module
	PLIB_SB_ExistsPGVErrCmdCode	Identifies whether the PGVErrCmdCode feature exists on the SB module
	PLIB_SB_ExistsPGVErrInitID	Identifies whether the PGVErrInitID feature exists on the SB module
	PLIB_SB_ExistsPGVErrPG	Identifies whether the PGVErrPG feature exists on the SB module
	PLIB_SB_ExistsPGVErrRegion	Identifies whether the PGVErrRegion feature exists on the SB module
	PLIB_SB_ExistsPGVErrRptPri	Identifies whether the PGVErrRptPri feature exists on the SB module
	PLIB_SB_ExistsPGVErrStatus	Identifies whether the PGVErrStatus feature exists on the SB module
	PLIB_SB_InitPermGrpSet	Sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.
	PLIB_SB_PGRegionAddrGet	Returns the base address for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionAddrSet	Sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermClear	Clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermSet	Sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionSizeGet	Returns the size for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionSizeSet	Sets the size for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermClear	Clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermSet	Sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGVErrorClearMulti	Clears multiple permission group errors for the specified target.
	PLIB_SB_PGVErrorClearSingle	Clears a single permission group error for the specified target.
	PLIB_SB_PGVErrorCode	Returns a value corresponding to the type of error logged for the specified target.
	PLIB_SB_PGVErrorCommandCode	Returns the OCP command code of the transaction that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorInitiatorID	Returns the ID of the Initiator that caused the protection violation
	PLIB_SB_PGVErrorLogClearMulti	Clears a multiple protection group violations error from the specified target error log register.
	PLIB_SB_PGVErrorLogClearSingle	Clears a single protection group violation error from the specified target error log register.

	PLIB_SB_PGVErrorMulti	Indicates if more than one perimission group violation has occurred since last cleared.
	PLIB_SB_PGVErrorPermissionGroup	Returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorRegion	Returns the number of the protection region in the specified target address space that caused the protection violation.
	PLIB_SB_PGVErrorReportPrimaryDisable	Disables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorReportPrimaryEnable	Enables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorStatus	Identifies whether a permission group violation has been reported for the specified target.

File Name

plib_sb.h

Company

Microchip Technology Inc.

5.6.22 SPI Peripheral Library

5.6.22.1 Introduction

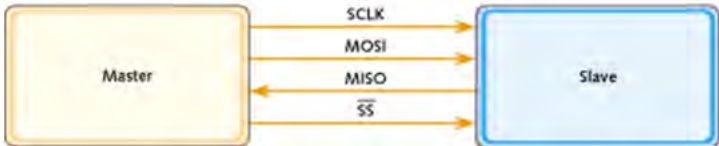
SPI Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Serial Peripheral Interface (SPI) module that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description



The Serial Peripheral Interface (SPI) module is a synchronous serial interface useful for communicating with other peripherals or microcontroller device. These peripheral devices may be serial EEPROMs, shift registers, display drivers, analog-to-digital converters, etc.

SPI is a synchronous serial data link operating at full duplex Master/slave relationship.

Two data lines:

- MOSI – Master Data Output, Slave Data Onput
- MISO – Master Data Input, Slave Data Output

Two control lines:

- SCLK – Clock
- /SS – Slave Select (no addressing)

Devices communicate in Master/Slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (Chip Select) lines. The SPI is sometimes referred to as a "four-wire" serial bus, contrasting with three-, two-, and one-wire serial buses.

5.6.22.2 Release Notes

MPLAB Harmony Version: v0.70b **SPI Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.22.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.22.4 Using the Library

This topic describes the basic architecture of the SPI Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_spi.h](#)

The interface to the SPI Peripheral Library is defined in the "[plib_spi.h](#)" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the SPI Peripheral library must include "peripheral.h".

Library File:

The SPI Peripheral Library is part of the processor-specific peripheral library archive (.a) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

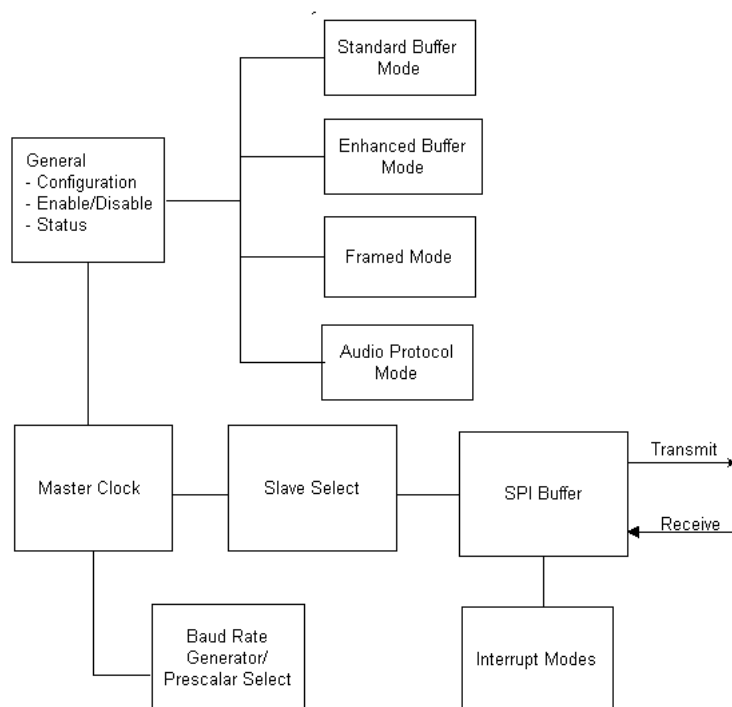
5.6.22.4.1 Library Architecture

This library provides the low level abstraction of the SPI module available on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all SPI functionality available on the device. Refer to the "**Serial Peripheral Interface (SPI)**" chapter in the specific device data sheet for availability.

Description

The following diagram shows the general architecture of the SPI Peripheral Library.



The library interface routines are divided into various subsections, each of the subsection addresses one of the blocks or the overall operation of the SPI module.

Library Interface Section	Description
General Configuration and Status Functions	This section provides a set of functions and data types for configuring the SPI and to read the status of the SPI.
Data Transfer Functions	This section provides a set of functions and data types for Reading and Writing the SPI buffer values.
Transmitter Functions	This section provides a set of functions for transmitter.
Receiver Functions	This section provides a set of functions for receiver.
Framed Mode Functions	Provides control, status, and data transfer routines for Framed SPI mode.
Audio Mode Functions	Provides control, status routines to support audio protocol functionality.

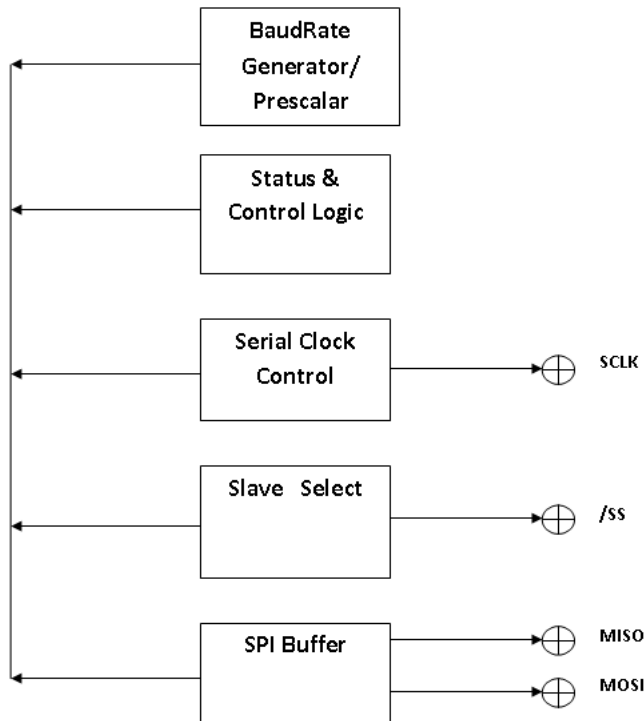
Note: Not all modes are available on all devices. Refer to the "**Serial Peripheral Interface (SPI)**" chapter in the specific device data sheet for availability.

5.6.22.4.2 Hardware Abstraction Model

This library provides a low-level abstraction of the SPI module on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The SPI Peripheral Library provides interface routines to interact with the blocks shown in the following diagram.



Hardware Abstraction Model

The Serial Peripheral Interface (SPI) module is a synchronous serial interface useful for communicating with external peripherals and other microcontroller devices.

The SPI bus specifies four logic signals:

- SCLK: serial clock (output from master)
- MOSI: master output, slave input (output from master)
- MISO: master input, slave output (output from slave)
- /SS: slave select (active low, output from master); on certain devices, this pin is implemented using general purpose I/O (GPIO)

The SPI bus can operate with a single master device and with one or more slave devices. To begin a communication, the master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data.

The master then transmits the appropriate chip select bit for the desired chip to a logic '0'. A logic '0' is transmitted because the Chip Select line (/SS) is active low, meaning its off state is a logic '1'; the on state is asserted with a logic 0. The master issues the clock cycles.

During each SPI clock cycle, a full duplex data transmission occurs:

- The master sends a bit on the MOSI line; the slave reads it from that same line
- The slave sends a bit on the MISO line; the master reads it from that same line

Transmissions may involve any number of clock cycles. When there are no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

The **Baud Rate Generator/Prescaler** controls the timing, the desired baud rate can be programmed in the baud rate controller.

In the Master mode, the **clock** becomes the serial clock and is provided to external devices via the SCK pin (the clock can be prescaled by the primary prescaler and the secondary prescaler if device supports).

The **Buffers** are for data transmitted or received by the SPI module over the MISO and MOSI line synchronized with the SCK line by the clock control logic.

The **Status and Control logic**, provide the capability to control different ways of enabling or disabling the master and slave. It also can provides status about the transmitter and receiver.

Active-Low **Slave Select** or Frame Synchronization I/O Pulse allows for a Synchronous Slave mode.

The SPI module supports the following four SPI modes.

- **Standard Mode:** In this mode of operation, data can be thought of as taking a direct path between the Most Significant bit (MSb) of one module's shift register and the Least Significant bit (LSb) of the other, and then into the appropriate Transmit or Receive Buffer. The master provides the serial clock and synchronization signals required to the slave device.
- **Enhanced Buffer Mode:** The operation of this mode is very similar to that of Standard mode. The difference is that data can be thought of as moving from the Shift register to a receive FIFO buffer and moving from the transmit FIFO buffer to the Shift register.
- **Framed Mode:** In this mode of operation, the Frame Master controls the generation of the frame synchronization pulse and provides this pulse to other devices at the Slave Select (/SS) pin. The SPI clock is generated by the SPI Master and is continuously running. The SPI slave module uses a frame synchronization pulse received at the SS pin.
- **Audio Protocol Mode:** The SPI module provides support to the audio protocol modes; with this mode, the SPI module can be interfaced to most codec devices available today to provide PIC microcontroller-based audio solutions

Note: Not all features are available on all devices. Refer to the "**Serial Peripheral Interface (SPI)**" chapter in the specific device data sheet for availability.

5.6.22.4.3 How the Library Works

The SPI module has the following operating modes:

- Standard SPI
- Enhanced Buffer SPI
- Framed SPI
- Audio Protocol Interface

Note: Not all modes are available on all devices. Refer to the "**Serial Peripheral Interface (SPI)**" chapter in the specific device data sheet for availability.

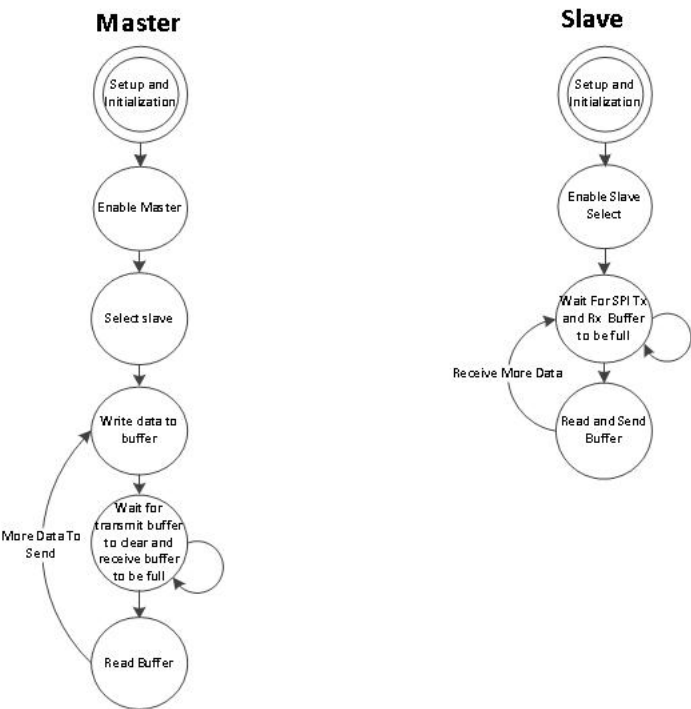
5.6.22.4.3.1 State Machine

This state machine is provided to give a general idea of the usage model of the peripheral library. Refer to the usage model for more detailed steps for the scenario that is being used.

Description

SPI State Machine

The following state machine diagrams shows the transmission and reception during normal operation.



SPI Routines

State	Associated Function
Setup and Initialization	Initialize the SPI by setting prescalar/ baud rate generator, interrupt modes.
Enable Master	Once the SPI has been appropriately set up and initialized, the state machine enables Master mode (PLIB_SPI_MasterEnable).
Select Slave	Select slave by pulling the /SSx pin low to transmit the data.
Write Data To Transfer	Write data to the buffer to transmit using PLIB_SPI_BufferWrite .
Wait for Transmit Buffer to Clear	Buffer Data will be transmitted to transmit buffer and transmit buffer flag will be full. The state machine waits for transmit buffer to clear. Check for status by calling PLIB_SPI_TransmitBufferIsFull .
Wait for Receive Buffer Full to Set	Slave sends received data back to master. Upon data received receive buffer flag will set. The state machine waits for Receive buffer to clear. Check for status by calling PLIB_SPI_ReceiveBufferIsFull .
Read Buffer	Read the buffer using PLIB_SPI_BufferRead .
Enable Slave Select	Enable slave select by calling PLIB_SPI_PinEnable .

Note: Not all features are available on all devices. Refer to the "Serial Peripheral Interface (SPI)" chapter in the specific device data sheet for availability.

5.6.22.4.3.2 Standard SPI Mode

In Standard (legacy) Master and Slave modes, data can be thought of as taking a direct path between the Most Significant bit (MSb) of one module's shift register and the Least Significant bit (LSb) of the other, and then into the appropriate Transmit or Receive buffer. The module configured as the master module provides the serial clock and synchronization signals (as required) to the slave device.

5.6.22.4.3.2.1 Standard Master Mode

In Standard Master mode, the input clock is used as the serial clock. The serial clock is output via the SCK pin to slave devices. Clock pulses are only generated when there is data to be transmitted.

Setup

1. Select on which edge of the clock data transmission occurs, using [PLIB_SPI_OutputDataPhaseSelect](#) and [PLIB_SPI_ClockPolaritySelect](#).
2. The clock is prescaled using [PLIB_SPI_PrescalePrimarySelect/PLIB_SPI_PrescaleSecondarySelect](#), if supported by the device.
3. If the module needs to operate as a Master, use [PLIB_SPI_MasterEnable](#).

Example: Standard Master Mode Communication Setup

```
#define MY_SPI_ID    SPI_ID_1

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_MasterEnable(MY_SPI_ID);
PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

1. Data to be transmitted is written to the SPI buffer([PLIB_SPI_BufferWrite](#)).
2. When contents of buffer are moved to the shift register, the SPI transmit buffer full flag is cleared (this can be verified using [PLIB_SPI_TransmitBufferIsFull](#)).
3. A series of 8/16/32 clock pulses shifts out 8/16/32 bits of transmit data from the shift register to the data out (SDO) pin and simultaneously shifts in the data at the data in (SDI) pin into the shift register.
4. When the transfer is complete, the following events occur:
 - The SPI interrupt flag is set. Interrupts will occur if SPI interrupts are enabled. The SPI interrupt flag is not cleared automatically by the hardware.

- Also, when the ongoing transmit and receive operation is completed, the contents of the shift register are moved to the SPI receive register.
 - The SPI receive buffer full flag (`PLIB_SPI_ReceiveBufferIsFull`) is set by the module, indicating that the receive buffer is full. Once the SPI buffer is read by the user application using `PLIB_SPI_BufferRead`, the hardware clears the SPI receive buffer full flag.
5. If the SPI receive buffer full flag is set when the SPI module needs to transfer data from SPI shift register to SPI receive buffer, the module will set the SPI receive overflow flag (`PLIB_SPI_ReceiverHasOverflowed`), indicating an overflow condition.
 6. Data to be transmitted can be written to SPI buffer (`PLIB_SPI_BufferWrite`) by the user software at any time as long as the SPI Transmit buffer full flag is clear (`PLIB_SPI_TransmitBufferIsFull`). The write can occur while SPI shift register is shifting out the previously written data, allowing continuous transmission.

Example: Standard Master Mode Communication Transfer

```
#define MY_SPI_ID    SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiveBufferIsFull or wait for the interrupt

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

1. Refer to the **"Interrupts"** section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.
2. `MY_SPI_ID` is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

5.6.22.4.3.2.2 Standard Slave Mode

Slave Select Synchronization

The Slave Select pin (/SS) allows for Synchronous Slave mode. If the slave select is enabled (`PLIB_SPI_PinEnable`), transmission and reception are enabled in Slave mode only if the /SS pin is driven to a low state. The port output or other peripheral outputs must not be driven in order to allow the /SS pin to function as an input. If the slave select is enabled and the /SS pin is driven high, the data output (SDO) pin is no longer driven and will tri-state even if the module is in the middle of a transmission. An aborted transmission will be retried the next time the /SS pin is driven low, using the data held in the SPI Transmit buffer. If slave select is not enabled, the /SS pin does not affect the module operation in Slave mode.

SPI Transmit Buffer Full Status Operation

The function of the SPI Transmit buffer full (`PLIB_SPI_TransmitBufferIsFull`) is different in the Slave mode of operation. If slave select is disabled (`PLIB_SPI_PinDisable`), the `PLIB_SPI_TransmitBufferIsFull` returns a '1' when the SPI buffer is loaded by the user application. It is cleared when the module transfers data from SPI transmit buffer to SPI shift register. This is similar to the SPI Transmit buffer full function in Master mode.

If slave select is enabled (`PLIB_SPI_PinEnable`), `PLIB_SPI_TransmitBufferIsFull` returns a '1' when the SPI buffer is loaded by the user application. However, `PLIB_SPI_TransmitBufferIsFull` returns zero only when the SPI module completes data transmission. A transmission will be aborted when the Slave Select pin goes high and may be retried at a later time. Each data word is held in SPI transmit buffer until all bits are transmitted to the receiver.

Setup

In Standard Slave mode, data is transmitted and received as the external clock pulses appear on the SCK pin. The clock polarity and clock edge ([PLIB_SPI_ClockPolaritySelect](#)) determine upon which edge of the clock data transmission occurs.

Example: Standard Slave Mode Communication Setup

```
#define MY_SPI_ID    SPI_ID_1

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and
edge is subject to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);//
... based on your communication mode.

PLIB_SPI_SlaveEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer ([PLIB_SPI_BufferWrite](#)/ [PLIB_SPI_BufferRead](#)). The remainder of the operation of the module is identical to that of Standard Master mode.

Example: Standard Slave Mode Communication Receive

```
#define MY_SPI_ID    SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflown(MY_SPI_ID))
{
    // error
    PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
    data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
    // error
}
else
{
    data = PLIB_SPI_BufferRead (MY_SPI_ID); // read data
    while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
    PLIB_SPI_BufferWrite(MY_SPI_ID, data); // send back
}
```

```
}
```

Notes:

1. Refer to the **"Interrupts"** section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

5.6.22.4.3.3 Enhanced Buffer SPI Mode

The operation of Enhanced Buffer Master and Slave modes is very similar to Standard Master and Slave modes. The difference is that data can be thought of as moving from the Shift register to a receive FIFO buffer and moving from the transmit FIFO buffer to the Shift register.

5.6.22.4.3.3.1 Enhanced Buffer Master Mode

In Enhanced Buffer Master mode (referred to as a FIFO), the input clock used as the serial clock. The serial clock is output via the SCK pin to slave devices. Clock pulses are only generated when there is data to be transmitted.

Setup

1. Select on which edge of the clock data transmission occurs, using [PLIB_SPI_OutputDataPhaseSelect](#) and [PLIB_SPI_ClockPolaritySelect](#).
2. The clock is prescaled using [PLIB_SPI_PrescalePrimarySelect/PLIB_SPI_PrescaleSecondarySelect](#), or the clock is generated using [PLIB_SPI_BaudRateClockSelect](#), if supported by the device.
3. If the module needs to operate as a Master, use [PLIB_SPI_MasterEnable](#).

Example: Enhanced Master Mode Communication Setup

```
#define MY_SPI_ID    SPI_ID_1

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_MasterEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_FIFOEnable(MY_SPI_ID);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

The CPU loads data to be transmitted into the transmit buffer by writing the SPI buffer ([PLIB_SPI_BufferWrite](#)). An SPI transmission begins after the first buffer write. Up to all pending transmissions can be loaded. The number of pending transfers is indicated by the Buffer Element Count bits through [PLIB_SPI_FIFOCountGet](#).

In Master mode, this count reflects the number of transfers pending in the transmit buffer. In Slave mode, it reflects the number of unread receptions in the receive buffer. If the Shift register is empty, the first write will immediately load the Shift register, leaving all transmit buffer locations available.

After an SPI transfer completes, the receive buffer location is updated with the received data. The CPU accesses the received data by reading the SPI buffer. After each CPU read, the SPI buffer points to the next buffer location. SPI transfers continue until all pending data transfers have completed.

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

1. Data to be transmitted is written to the SPI buffer([PLIB_SPI_BufferWrite](#)) and is loaded into the next available transmit buffer location.
2. The SPI transmit buffer full flag ([PLIB_SPI_TransmitBufferIsFull](#)) and SPI interrupt flag are set after pending transfers are loaded.
3. The current buffer location's contents are moved to the Shift register. The SPI transmit buffer is cleared by the module if a buffer location is available for a CPU write.
4. A series of 8/16/32 clock pulses shifts out 8/16/32 bits of transmit data from the shift register to the data out (SDO) pin and simultaneously shifts in the data at the data in (SDI) pin into the shift register.
5. When the transfer is complete, the following events occur:
 - The contents of the SPI shift register are moved into the next available location in the receive buffer.
 - If the last unread location is written by the SPI module, the SPI receive buffer full flag ([PLIB_SPI_ReceiveBufferIsFull](#)) is set, indicating that all buffer locations are full. Enable the SPI interrupts. The SPI interrupt flag is not cleared automatically by the hardware.
 - Once the SPI buffer is read ([PLIB_SPI_BufferRead](#)) by the user application, the hardware clears the SPI receive buffer full flag ([PLIB_SPI_ReceiveBufferIsFull](#)) and the SPI Buffer increments to the next unread receive buffer location. SPI buffer reads beyond the last unread location will not increment the buffer location.
6. When [PLIB_SPI_ReceiveBufferIsFull](#) is set, if the SPI module needs to transfer one more data from SPI shift register to the buffer, the module will enable the receive overflow flag ([PLIB_SPI_ReceiverHasOverflowed](#)) indicating an overflow condition.
7. Data to be transmitted can be written to the SPI buffer ([PLIB_SPI_BufferWrite](#)) by the user application at any time as long as the SPI transmit buffer full flag is clear ([PLIB_SPI_TransmitBufferIsFull](#)). Up to all pending transfers can be loaded into the buffer allowing continuous transmission.

Example: Enhanced Master Mode communication Transfer

```
#define MY_SPI_ID    SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiveBufferIsFull or wait for the interrupt

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

1. Refer to the **"Interrupts"** section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device

data sheet for availability.

5.6.22.4.3.3.2 Enhanced Buffer Slave Mode

Slave Select Synchronization

The Slave Select pin allows a Synchronous Slave mode. If the slave select enabled ([PLIB_SPI_PinEnable](#)), transmission and reception is enabled in Slave mode only if the /SS pin is driven to a low state. The port output or other peripheral outputs must not be driven in order to allow the /SS pin to function as an input. If the slave select is enabled and the /SS pin is driven high, the SDO pin is no longer driven and will tri-state even if the module is in the middle of a transmission. An aborted transmission will be retried the next time the /SS pin is driven low using the data held in the SPI transmit buffer. If the Slave select is disabled ([PLIB_SPI_PinDisable](#)), the /SS pin does not affect the module operation in Slave mode.

SPI Transmit Buffer Full Operation

The function of the SPI transmit buffer full flag ([PLIB_SPI_TransmitBufferIsFull](#)) is different in the Slave mode of operation. If Slave select is disabled ([PLIB_SPI_PinDisable](#)), the SPI Transmit Buffer Full flag ([PLIB_SPI_TransmitBufferIsFull](#)) is set when the last available buffer location is loaded by the user application. It is cleared when the module transfers data from the buffer to SPI status register and a buffer location is available for a CPU write. This is similar to the SPI transmit buffer in Master mode.

If Slave select is enabled ([PLIB_SPI_PinEnable](#)), the SPI transmit buffer is set when the last available buffer location is loaded by the user application. However, it is cleared only when the SPI module completes data transmission, leaving a buffer location available for a CPU write. A transmission will be aborted when the /SS pin goes high and may be retried at a later time. Each data word is held in the buffer until all bits are transmitted to the receiver.

Setup

In Slave mode, data is transmitted and received as the external clock pulses appear on the SCK pin. [PLIB_SPI_OutputDataPhaseSelect](#) and [PLIB_SPI_ClockPolaritySelect](#) determine upon which edge of the clock data transmission occurs. The rest of the operation of the module is identical to that of Enhanced Buffer Master mode.

Example: Enhanced Slave Mode communication Setup

```
#define MY_SPI_ID    SPI_ID_1

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_AT_END); //clock polarity and
edge is subject to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK); //
... based on your communication mode.

PLIB_SPI_SlaveEnable(MY_SPI_ID);
PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_FIFOEnable(MY_SPI_ID);

// Optional: Enable interrupts.
```

```
// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive:

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer ([PLIB_SPI_BufferWrite](#)/ [PLIB_SPI_BufferRead](#)). The remainder of the operation of the module is identical to that of Enhanced Buffer Master mode.

Example: Enhanced Slave Mode communication Receive

```
#define MY_SPI_ID    SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
    // error
    PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID); // clear overflow
    data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
    // error
}
else
{
    data = PLIB_SPI_BufferRead (MY_SPI_ID); // read data
    while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
    PLIB_SPI_BufferWrite(MY_SPI_ID, data); // send back
}
```

Notes:

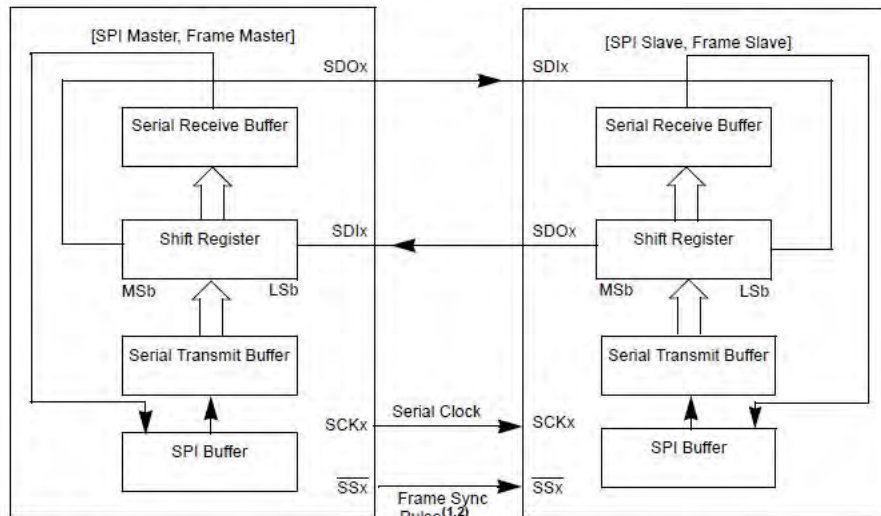
1. Refer to the "**Interrupts**" section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the "**Serial Peripheral Interface (SPI)**" chapter in the specific device data sheet for availability.

5.6.22.4.3.4 Framed SPI Modes

The SPI module supports a basic framed SPI protocol while operating in either Master or Slave modes.

Description

The SPI module supports two framed modes of operation. In Frame Master mode, the SPI module generates the frame synchronization pulse and provides this pulse to other devices at the /SS pin. In Frame Slave mode, the SPI module uses a frame synchronization pulse received at the /SS pin.



Note 1: In Framed SPI modes, the $\overline{SSx}$ pin is used to transmit/receive the frame synchronization pulse.
 2: Framed SPI modes require the use of all four pins (i.e., using the $\overline{SSx}$ pin is not optional).

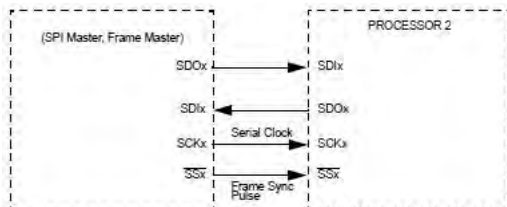
The framed SPI modes are supported in conjunction with the unframed Master and Slave modes. This makes four framed SPI configurations:

- SPI Master mode and Frame Master mode
- SPI Master mode and Frame Slave mode
- SPI Slave mode and Frame Master mode
- SPI Slave mode and Frame Slave mode

These modes determine whether or not the SPI module generates the serial clock and the frame synchronization pulse.

5.6.22.4.3.4.1 SPI Master Mode and Frame Master Mode

In Master/Frame Master mode, the SPI module generates both the clock and frame synchronization signals, as shown in the following figure.



In this mode, the serial clock is output continuously at the SCK pin, regardless of whether the module is transmitting. When the SPI buffer is written ([PLIB_SPI_BufferWrite](#)), the $\overline{SS}$ pin will be driven to its active state on the appropriate transmit edge of the SCK clock, and remain active for one data frame. If the [PLIB_SPI_FrameSyncPulseEdgeSelect](#) function decides whether sync pulse precedes the data transmission or coincides with the beginning of the data transmission. The module starts transmitting data on the next transmit edge of the SCK.

Setup

The mode is enabled by calling [PLIB_SPI_MasterEnable](#), [PLIB_SPI_FramedCommunicationEnable](#) and [PLIB_SPI_FrameSyncPulseDirectionSelect](#) (as output).

Example: SPI Master Mode and Frame Master Mode Communication Setup

```

#define MY_SPI_ID    SPI_ID_1

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_MasterEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID, SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);

```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Master Mode and Frame Master Mode Communication Transfer

```

#define MY_SPI_ID    SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiveBufferIsFull or wait for the interrupt

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);

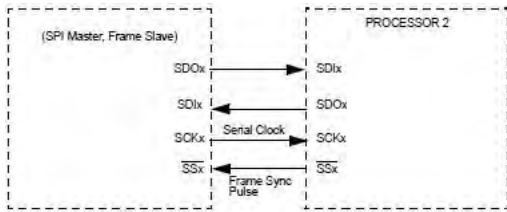
```

Notes:

1. Refer to the **"Interrupts"** section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

5.6.22.4.3.4.2 SPI Master Mode and Frame Slave Mode

In Master/Frame Slave mode, the module generates the clock signal but uses the slave module's frame synchronization signal for data transmission, as shown in the following figure.



In this mode, the /SS pin is an input and it is sampled on the sample edge of the SPI clock. When it is sampled in its active state, data will be transmitted on the subsequent transmit edge of the SPI clock. The SPI interrupt flag is set when the transmission is complete. The user application must make sure that the correct data is loaded into the SPI buffer for transmission before the signal is received at the /SS pin.

Setup

The mode is enabled by calling [PLIB_SPI_MasterEnable](#), [PLIB_SPI_FramedCommunicationEnable](#), and [PLIB_SPI_FrameSyncPulseDirectionSelect](#).

Example: SPI Master Mode and Frame Slave Mode Communication Setup

```
#define MY_SPI_ID    SPI_ID_1

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_ID, SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER);
PLIB_SPI_MasterEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID, SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Master Mode and Frame Slave Mode Communication Transfer

```
#define MY_SPI_ID    SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID, data);

// Either Poll for Receiver buffer to be full using
```

```
// PLIB_SPI_ReceiveBufferIsFull or wait for the interrupt
```

```
// Read the received value
```

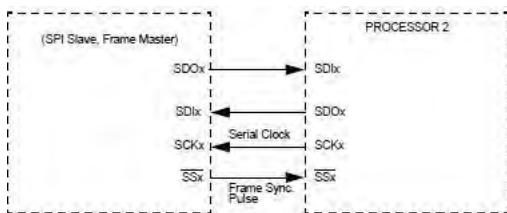
```
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

1. Refer to the **"Interrupts"** section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

5.6.22.4.3.4.3 SPI Slave Mode and Frame Master Mode

In Slave/Frame Master mode, the module acts as the SPI slave and takes its clock from the other SPI module; however, it produces frame synchronization signals to control data transmission, as shown in the following figure.



The input SPI clock will be continuous in Slave mode. The /SS pin will be an output when the Frame sync pulse is output. Therefore, when the SPI buffer is written, the module drives the /SS pin to the active state on the appropriate transmit edge of the SPI clock for one SPI clock cycle. Data will start transmitting on the appropriate SPI clock transmit edge.

Setup

The mode is enabled by calling [PLIB_SPI_SlaveEnable](#), [PLIB_SPI_FramedCommunicationEnable](#), and [PLIB_SPI_FrameSyncPulseDirectionSelect](#).

Example: SPI Slave Mode and Frame Master Mode Communication Setup

```
#define MY_SPI_ID    SPI_ID_1

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH); //clock polarity and edge is subject to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK); // ... based on your communication mode.
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_ID, SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER);

PLIB_SPI_SlaveEnable(MY_SPI_ID);
PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
```

```

PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID, SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);

```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Slave Mode and Frame Master Mode Communication Receive

```

#define MY_SPI_ID    SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
    // error
    PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID); // clear overflow
    data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
    // error
}
else
{
    data = PLIB_SPI_BufferRead (MY_SPI_ID); // read data
    while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
    PLIB_SPI_BufferWrite(MY_SPI_ID, data); // send back
}

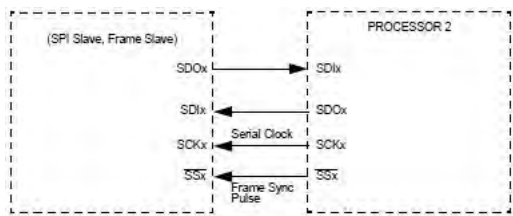
```

Notes:

1. Refer to the "Interrupts" section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the "Serial Peripheral Interface (SPI)" chapter in the specific device data sheet for availability.

5.6.22.4.3.4.4 SPI Slave Mode and Frame Master Mode

In Slave/Frame Slave mode, the module obtains both its clock and frame synchronization signal from the master module, as shown in the following figure.



In this mode, both the clock and Slave Select pins will be inputs. The /SS pin is sampled on the sample edge of the SPI clock. When /SS is sampled at its active state, data will be transmitted on the appropriate transmit edge of SCK.

Setup

The mode is enabled by calling [PLIB_SPI_SlaveEnable](#), [PLIB_SPI_FramedCommunicationEnable](#) and [PLIB_SPI_FrameSyncPulseDirectionSelect](#) (as input).

Example: SPI Slave Mode and Frame Slave Mode Communication Setup

```

#define MY_SPI_ID    SPI_ID_1

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and
edge is subject to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);//
... based on your communication mode.
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_ID, SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER);

PLIB_SPI_SlaveEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID, SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);

```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Slave Mode and Frame Slave Mode Communication Receive

```

#define MY_SPI_ID    SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
    // error
    PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
    data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
    // error
}
else
{
    data = PLIB_SPI_BufferRead (MY_SPI_ID);           // read data
    while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
    PLIB_SPI_BufferWrite(MY_SPI_ID, data);           // send back
}

```

Notes:

1. Refer to the **"Interrupts"** section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.

2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

5.6.22.4.3.5 Audio Protocol Interface Mode

The SPI module can be interfaced to most codec devices available today to provide PIC microcontroller-based audio solutions.

Description

The SPI module provides support to the audio protocol functionality via four standard I/O pins. The four pins that make up the audio protocol interface modes are:

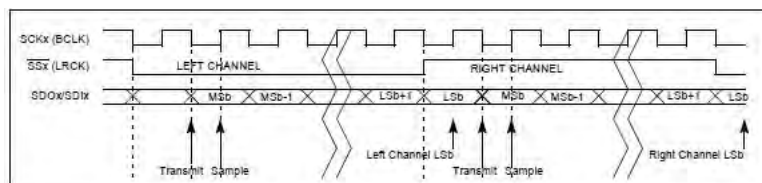
- SDIx: Serial Data Input for receiving sample digital audio data
- SDOx: Serial Data Output for transmitting digital audio data
- SCKx: Serial Clock (also known as bit clock)
- /SSx: Left/Right Channel Clock

The SPI module supports four audio protocol modes and can be operated in any one of these modes:

Note: Each of the modes can additionally support some or all of the features listed below. Please refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for more information.

I²S Mode

The Inter-IC Sound (I²S) protocol enables transmission of two channels of digital audio data over a single serial interface. The I²S specification defines a half-duplex interface that supports transmit or receive, but not both at the same time. With both SDO and SDI available, full-duplex operation is supported by this peripheral, as shown in the following figure.



Data Transmit and Clocking:

- The transmitter shifts the audio sample data's Most Significant bit (MSb) on the first falling edge of SCK after an LRCK transition
- The receiver samples the MSB on the second rising edge of SCK
- The left channel data shifts out while LRCK is low and right channel data is shifted out while LRCK is high
- The data in the left and right channel consists of a single frame

To set the module to I²S mode, the following bits must be set:

- Set SPI to I²S Mode by calling [PLIB_SPI_AudioProtocolModeSelect](#)
- Select [PLIB_SPI_FrameSyncPulsePolaritySelect](#) (as active low)
- Select [PLIB_SPI_ClockPolaritySelect](#) (as active low)

Setting these bits enables the SDO and LRCK (/SSx) transitions to occur on the falling edge of SCK (BCLK) and sampling of SDI to occur on the rising edge of SCK.

Left-Justified Mode

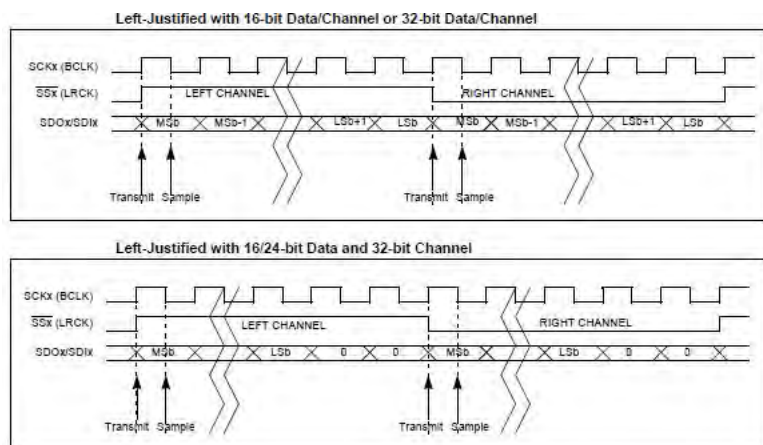
The Left-Justified mode is similar to I²S mode; however, in this mode, the SPI shifts the audio data's MSb on the first SCK edge that is coincident with an LRCK transition. On the receiver side, the SPI module samples the MSb on the next SCK edge.

In general, a codec using justified protocols defaults to transmitting data on the rising edge of SCK and receiving data on the falling edge of SCK.

To set the module to Left-Justified mode, the following bits must be set

- Set SPI to Left Justified Mode by calling [PLIB_SPI_AudioProtocolModeSelect](#)
- Select [PLIB_SPI_FrameSyncPulsePolaritySelect](#) (as active high)
- Select [PLIB_SPI_ClockPolaritySelect](#) (as active high)

This enables the SDO and LRCK transitions to occur on the rising edge of SCK. Refer to the following sample waveform diagrams for 16-, 24-, and 32-bit audio data transfers.



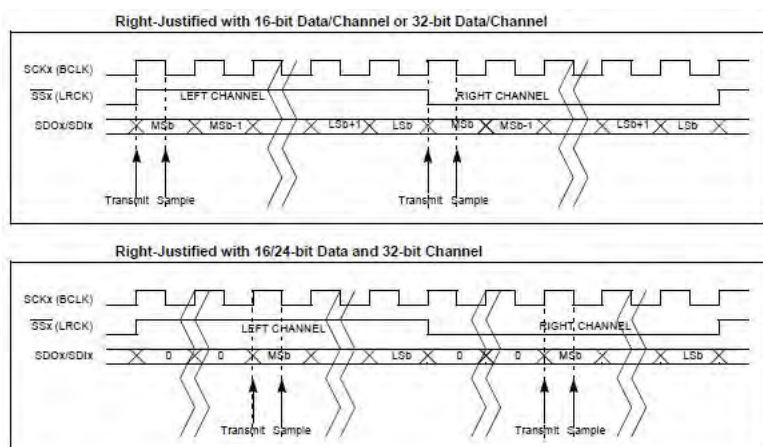
Right-Justified Mode

In Right-Justified mode, the SPI module shifts the audio sample data's MSb after aligning the data to the last clock cycle. The bits preceding the audio sample data can be driven to logic level 0.

To set the module to Right-Justified mode, the following bits must be set:

- Set SPI to Right Justified Mode by calling [PLIB_SPI_AudioProtocolModeSelect](#)
- Select [PLIB_SPI_FrameSyncPulsePolaritySelect](#) (as active high)
- Select [PLIB_SPI_ClockPolaritySelect](#) (as active high)

This enables the SDO and LRCK transitions to occur on the rising edge of SCK after the Least Significant bit (LSb) being aligned to the last clock cycle. Refer to the following sample waveform diagrams for 16-, 24-, and 32-bit audio data transfers.



PCM/DSP Mode

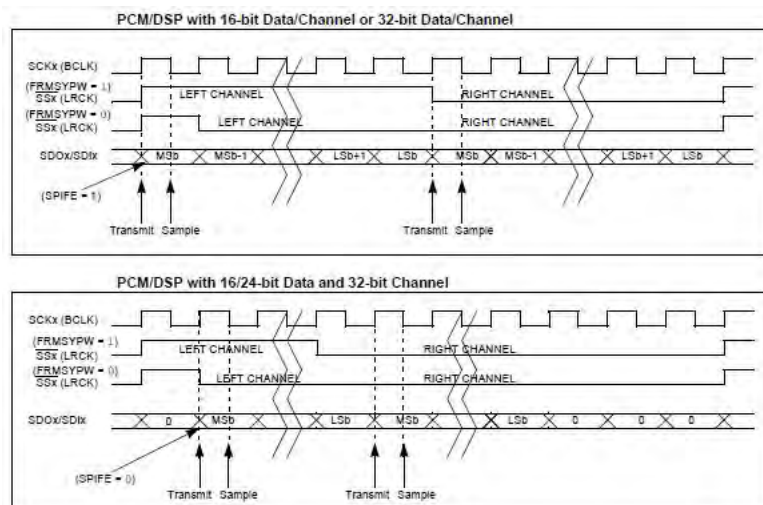
This mode modifies the behavior of LRCK and audio data spacing. In PCM/DSP mode, the LRCK can be a single bit wide (i.e., 1 SCK) or as wide as the audio data (16-, 24-, and 32-bits). The audio data is packed in the frame with the left channel data immediately followed by the right channel data. The frame length is still either 32 or 64 clocks when this device is the master.

In PCM/DSP mode, the transmitter drives the audio data's (left channel) MSb on the first or second transmit edge of SCK (after an LRCK transition). Immediately after the (left channel) LSB, the transmitter drives the (right channel) MSb.

To set the module to Left-Justified mode, the following bit must to be set:

- Set the SPI module to Right-Justified Mode by calling [PLIB_SPI_AudioProtocolModeSelect](#)

Refer to the following sample waveform for 16-, 24-, and 32-bit audio data transfers.



Mono Mode Versus Stereo Mode

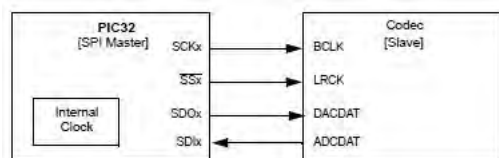
The SPI module enables the audio data transmission in Mono or Stereo mode by setting [PLIB_SPI_AudioTransmitModeSelect](#). In Stereo mode, the shift register uses each FIFO location once, which gives each channel a unique stream of data for stereo data. In Mono mode, the shift register uses each FIFO location twice, to give each channel the same mono stream of audio data.

5.6.22.4.3.5.1 Master Mode

A few characteristics of Master mode are:

- This mode enables the device to generate SCK and LRCK pulses as long as the master mode enabled
- The SPI module generates LRCK and SCK continuously in all the cases, regardless of the transmit data while in Master mode
- The SPI module drives the leading edge of LRCK and SCK within 1 SCK period and the serial data shifts in and out continuously even when the Transmit FIFO is empty

The following figure shows a typical interface between the master and slave while in Master mode.



Setup

To configure the device in Audio Protocol Master, enable master mode through [PLIB_SPI_MasterEnable](#) and enable audio mode through [PLIB_SPI_AudioProtocolEnable](#).

Example: Audio Mode Communication Setup

```
#define MY_SPI_ID    SPI_ID_1

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_DATA_IN|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_MasterEnable(MY_SPI_ID);
PLIB_SPI_AudioProtocolModeSelect(MY_SPI_ID, SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED );

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_AudioProtocolEnable(MY_SPI_ID);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: Audio Mode Communication Transfer

```
#define MY_SPI_ID    SPI_ID_1

uint16_t data;
data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID, data);

// wait for transfer to complete
while(!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID));

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

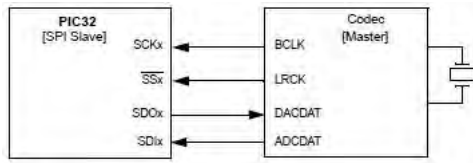
1. Refer to the **"Interrupts"** section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

5.6.22.4.3.5.2 Slave Mode

A few characteristics of Slave mode are:

- This mode enables the device to receive SCK and LRCK pulses as long as slave mode is enabled
- The SPI module drives zeros out of SDO, but does not shift data out or in (SDI) until the module receives the LRCK (i.e., the edge that precedes the left channel)
- Once the module receives the leading edge of LRCK, it starts receiving data if `PLIB_SPI_PinEnable` (enables data in) is selected and the serial data shifts out continuously even when the TX FIFO is empty

The following figure shows the interface between a SPI module in Audio Slave Interface mode to a codec master device.



Setup

The SPI module can be configured in Audio Protocol Slave mode by setting `PLIB_SPI_SlaveEnable` and enabling the audio protocol through `PLIB_SPI_AudioProtocolEnable`.

Example: Slave Mode communication Setup

```
#define MY_SPI_ID    SPI_ID_1

//Diabile SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear(MY_SPI_ID);

// Configure General SPI Options
// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH); //clock polarity and
edge is subject to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK); //
... based on your communication mode.

PLIB_SPI_SlaveEnable(MY_SPI_ID);
PLIB_SPI_AudioProtocolModeSelect(MY_SPI_ID, SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED );

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_AudioProtocolEnable(MY_SPI_ID);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer

([PLIB_SPI_BufferWrite](#)/ [PLIB_SPI_BufferRead](#)).

Example: Slave Mode communication Receive

```
#define MY_SPI_ID    SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflown(MY_SPI_ID))
{
    // error
    PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID); // clear overflow
    data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
    // error
}
else
{
    data = PLIB_SPI_BufferRead (MY_SPI_ID); // read data
    while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
    PLIB_SPI_BufferWrite(MY_SPI_ID, data); // send back
}
```

Notes:

1. Refer to the **"Interrupts"** section in the related family reference manual or device data sheet for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

5.6.22.4.3.6 Other Features

This topic describes the additional features available in the SPI Peripheral Library.

5.6.22.4.3.6.1 Communication Mode

The SPI module allows three types of data widths when transmitting and receiving data over a SPI bus. The selection of data width determines the minimum length of the SPI data. The user application should select the appropriate data width to maximize its data throughput. To change the mode of operation on the fly, the SPI module must be idle. If the SPI module is switched off, the new mode will be available when the module is switched on again.

The macro, `PLIB_SPI_CommunicationWidthSet`, allows the module to communicate in either 8-/16-/32-bit modes. The functionality will be the same for each mode, except for the number of bits that are received and transmitted.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

5.6.22.4.3.6.2 Power-Saving Modes

Note: Each of the following modes can additionally support some or all of the features listed below. Please refer to the **"Power-Saving Features"** chapter in the specific data sheet or the **"Power-Saving Modes"** family reference manual section for more information.

Sleep Mode

When the device enters Sleep mode, the device clock source and the entire device is shut down. The consequences of entering Sleep depend upon which mode (Master or Slave) the module is configured in at the time that Sleep mode is invoked.

Idle Mode

When the device enters Idle mode, the device clock is operational, but the CPU and selected peripherals are shut down.

The SPI module can continue to operate in Idle mode by calling [PLIB_SPI_StopInIdleDisable](#), or can stop operation in Idle mode by calling [PLIB_SPI_StopInIdleEnable](#).

Note: Not all features are available on all devices. Refer to the "Serial Peripheral Interface (SPI)" chapter in the specific device data sheet for availability.

5.6.22.4.3.6.3 SPI Receive-only Operation

Calling [PLIB_SPI_PinDisable](#) (data in pin) disables the transmission at the SDO pin. This allows the SPI module to be configured for a receive-only mode of operation. This function is applicable to all SPI operating modes.

Note: Not all features are available on all devices. Refer to the "Serial Peripheral Interface (SPI)" chapter in the specific device data sheet for availability.

5.6.22.4.3.6.4 SPI Error Handling

If a new data word has been shifted and the previous buffer contents have not been read, the status can be read by calling [PLIB_SPI_ReceiverHasOverflowed](#). Any received data is not transferred, and further data reception is disabled until the buffer is cleared by calling [PLIB_SPI_ReceiverOverflowClear](#).

Note: Not all features are available on all devices. Refer to the "Serial Peripheral Interface (SPI)" chapter in the specific device data sheet for availability.

5.6.22.4.3.6.5 SPI Master Mode Clock Frequency

In the SPI Master mode, the clock provided to the SPI module is the system cycle (TCY). This clock will then be prescaled by the primary prescaler ([PLIB_SPI_PrescaleSelectPrimary](#)), and the secondary prescaler ([PLIB_SPI_PrescaleSelectSecondary](#)). The prescaled instruction clock becomes the serial clock and is provided to external devices via the SCKx pin.

Note: Not all features are available on all devices. Refer to the "Serial Peripheral Interface (SPI)" chapter in the specific device data sheet for availability.

5.6.22.4.3.6.6 Interrupts

The SPI module has the ability to generate interrupts reflecting the events that occur during the data communication. Interrupts can be enabled or disabled using [PLIB_SPI_ErrorInterruptEnable/PLIB_SPI_ErrorInterruptDisable](#).

The following types of interrupts can be generated:

- Receive data available interrupts are signalled by SPI receive and transmit flag. This event occurs when there is new data assembled in the SPI receive buffer.
- Transmit buffer empty interrupts are signalled by SPI receive and transmit flag. This event occurs when there is space available in the SPI transmit buffer and new data can be written.
- Error interrupts are signalled by SPI receive and transmit flag. This event occurs when there is an overflow condition for the receive buffer when there is an underrun of the transmit buffer, or when a frame error event occurs.

Note: Not all features are available on all devices. Refer to the "Serial Peripheral Interface (SPI)" chapter in the specific device data sheet for availability.

5.6.22.4.4 Configuring the Library

The library is configured for the supported SPI modules when the processor is chosen in the MPLAB IDE.

5.6.22.5 Library Interface

Data Types and Constants

	Name	Description
	<u>SPI_AUDIO_COMMUNICATION_WIDTH</u>	Defines the list of SPI audio communication width.
	<u>SPI_AUDIO_ERROR</u>	Defines the list of SPI audio error.
	<u>SPI_AUDIO_PROTOCOL</u>	Data type defining the audio protocol mode.
	<u>SPI_AUDIO_TRANSMIT_MODE</u>	Defines the list of SPI transmit audio mode format.
	<u>SPI_BAUD_RATE_CLOCK</u>	Defines the list of SPI Baud Rate Generator (BRG).
	<u>SPI_CLOCK_POLARITY</u>	Defines the list of SPI clock polarity.
	<u>SPI_COMMUNICATION_WIDTH</u>	Defines the list of SPI communication width.
	<u>SPI_DATA_TYPE</u>	Data type defining the SPI data size.
	<u>SPI_ERROR_INTERRUPT</u>	Defines the list of SPI error interrupts.
	<u>SPI_FIFO_INTERRUPT</u>	Defines the list of SPI Buffer Interrupt mode.
	<u>SPI_FIFO_TYPE</u>	Defines the list of SPI buffer mode.
	<u>SPI_FRAME_PULSE_DIRECTION</u>	Defines the list of SPI frame sync pulse direction.
	<u>SPI_FRAME_PULSE_EDGE</u>	Defines the list of SPI frame sync pulse edge.
	<u>SPI_FRAME_PULSE_POLARITY</u>	Defines the list of SPI frame sync pulse polarity.
	<u>SPI_FRAME_PULSE_WIDTH</u>	Defines the list of SPI frame sync pulse width.
	<u>SPI_FRAME_SYNC_PULSE</u>	Data type defining the frame sync pulse counter values.
	<u>SPI_INPUT_SAMPLING_PHASE</u>	Defines the list of SPI data input sample phase.
	<u>SPI_MODULE_ID</u>	Identifies the supported SPI modules.
	<u>SPI_OUTPUT_DATA_PHASE</u>	Defines the list of SPI serial output data changes.
	<u>SPI_PIN</u>	Data type defining the SPI pin.
	<u>SPI_PRESCALE_PRIMARY</u>	Defines the list of possible primary prescaler values.
	<u>SPI_PRESCALE_SECONDARY</u>	Defines the list of possible secondary prescaler values.

Audio Mode Functions

	Name	Description
	<u>PLIB_SPI_AudioCommunicationWidthSelect</u>	Selects the data width for the SPI audio communication.
	<u>PLIB_SPI_AudioErrorDisable</u>	Disables the SPI error.
	<u>PLIB_SPI_AudioErrorEnable</u>	Enables the SPI error.
	<u>PLIB_SPI_AudioProtocolDisable</u>	Audio protocol is disabled.
	<u>PLIB_SPI_AudioProtocolEnable</u>	Audio protocol is enabled.
	<u>PLIB_SPI_AudioProtocolModeSelect</u>	Selects the Audio Protocol mode.
	<u>PLIB_SPI_AudioTransmitModeSelect</u>	Selects the transmit audio data format.

Data Transfer Functions

	Name	Description
	<u>PLIB_SPI_BufferClear</u>	Clears the SPI buffer.
	<u>PLIB_SPI_BufferRead</u>	Returns the SPI buffer value.
	<u>PLIB_SPI_BufferWrite</u>	Write the data to the SPI buffer.

Feature Existence Functions

	Name	Description
	PLIB_SPI_ExistsAudioCommunicationWidth	Identifies whether the AudioCommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsAudioErrorControl	Identifies whether the AudioErrorControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolControl	Identifies whether the AudioProtocolControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolMode	Identifies whether the AudioProtocolMode feature exists on the SPI module.
	PLIB_SPI_ExistsAudioTransmitMode	Identifies whether the AudioTransmitMode feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRate	Identifies whether the BaudRate feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRateClock	Identifies whether the BaudRateClock feature exists on the SPI module.
	PLIB_SPI_ExistsBuffer	Identifies whether the Buffer feature exists on the SPI module.
	PLIB_SPI_ExistsBusStatus	Identifies whether the BusStatus feature exists on the SPI module.
	PLIB_SPI_ExistsClockPolarity	Identifies whether the ClockPolarity feature exists on the SPI module.
	PLIB_SPI_ExistsCommunicationWidth	Identifies whether the CommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SPI module.
	PLIB_SPI_ExistsErrorInterruptControl	Identifies whether the ErrorInterruptControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOControl	Identifies whether the FIFOControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOCount	Identifies whether the FIFOCount feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOInterruptMode	Identifies whether the FIFOInterruptMode feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus	Identifies whether the FIFOShiftRegisterEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFramedCommunication	Identifies whether the FramedCommunication feature exists on the SPI module.
	PLIB_SPI_ExistsFrameErrorStatus	Identifies whether the FrameErrorStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseCounter	Identifies whether the FrameSyncPulseCounter feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseDirection	Identifies whether the FrameSyncPulseDirection feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseEdge	Identifies whether the FrameSyncPulseEdge feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulsePolarity	Identifies whether the FrameSyncPulsePolarity feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseWidth	Identifies whether the FrameSyncPulseWidth feature exists on the SPI module.
	PLIB_SPI_ExistsInputSamplePhase	Identifies whether the InputSamplePhase feature exists on the SPI module.

	PLIB_SPI_ExistsMasterControl	Identifies whether the MasterControl feature exists on the SPI module.
	PLIB_SPI_ExistsOutputDataPhase	Identifies whether the OutputDataPhase feature exists on the SPI module.
	PLIB_SPI_ExistsPinControl	Identifies whether the PinControl feature exists on the SPI module.
	PLIB_SPI_ExistsPrimaryPrescale	Identifies whether the PrimaryPrescale feature exists on the SPI module.
	PLIB_SPI_ExistsReadDataSignStatus	Identifies whether the ReadDataSignStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveBufferStatus	Identifies whether the ReceiveBufferStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveFIFOStatus	Identifies whether the ReceiveFIFOStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the SPI module.
	PLIB_SPI_ExistsSecondaryPrescale	Identifies whether the SecondaryPrescale feature exists on the SPI module.
	PLIB_SPI_ExistsSlaveSelectControl	Identifies whether the SlaveSelectControl feature exists on the SPI module.
	PLIB_SPI_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferEmptyStatus	Identifies whether the TransmitBufferEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferFullStatus	Identifies whether the TransmitBufferFullStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitUnderRunStatus	Identifies whether the TransmitUnderRunStatus feature exists on the SPI module.

Framed Mode Functions

	Name	Description
	PLIB_SPI_FramedCommunicationDisable	Disables framed SPI support.
	PLIB_SPI_FramedCommunicationEnable	Enables framed SPI support.
	PLIB_SPI_FrameErrorStatusGet	Returns the current status of the SPI frame error.
	PLIB_SPI_FrameSyncPulseCounterSelect	Selects at which character the SPI frame sync pulse is generated.
	PLIB_SPI_FrameSyncPulseDirectionSelect	Selects the frame sync pulse direction.
	PLIB_SPI_FrameSyncPulseEdgeSelect	Selects the frame sync pulse edge.
	PLIB_SPI_FrameSyncPulsePolaritySelect	Selects the frame sync pulse polarity.
	PLIB_SPI_FrameSyncPulseWidthSelect	Sets the frame sync pulse width.

General Configuration and Status Functions

	Name	Description
	PLIB_SPI_BaudRateClockSelect	Selects the type of clock is used by the Baud Rate Generator.
	PLIB_SPI_BaudRateSet	Sets the baud rate to the desired value.
	PLIB_SPI_ClockPolaritySelect	Enables clock polarity.
	PLIB_SPI_CommunicationWidthSelect	Selects the data width for the SPI communication.
	PLIB_SPI_Disable	Disables the SPI module.
	PLIB_SPI_Enable	Enables the SPI module.
	PLIB_SPI_ErrorInterruptDisable	Enables SPI error interrupts.
	PLIB_SPI_ErrorInterruptEnable	Enables SPI error interrupts
	PLIB_SPI_FIFOCountGet	Reads the SPI Buffer Element Count bits for either receive or transmit.

	PLIB_SPI_FIFODisable	Disables the SPI enhanced buffer.
	PLIB_SPI_FIFOEnable	Enables the SPI enhanced buffer.
	PLIB_SPI_FIFOInterruptModeSelect	Selects the SPI buffer interrupt mode.
	PLIB_SPI_FIFOShiftRegisterIsEmpty	Returns the current status of the SPI shift register.
	PLIB_SPI_InputSamplePhaseSelect	Selects the SPI data input sample phase.
	PLIB_SPI_IsBusy	Returns the current SPI module activity status.
	PLIB_SPI_MasterEnable	Enables the SPI in Master mode.
	PLIB_SPI_OutputDataPhaseSelect	Selects serial output data change.
	PLIB_SPI_PinDisable	Enables the selected SPI pins.
	PLIB_SPI_PinEnable	Enables the selected SPI pins.
	PLIB_SPI_PrescalePrimarySelect	Selects the primary prescale for SPI Master mode.
	PLIB_SPI_PrescaleSecondarySelect	Selects the secondary prescale for SPI Master mode.
	PLIB_SPI_ReadDataIsSignExtended	Returns the current status of the receive (RX) FIFO sign-extended data.
	PLIB_SPI_SlaveEnable	Enables the SPI in Slave mode.
	PLIB_SPI_SlaveSelectDisable	Disables Master mode slave select.
	PLIB_SPI_SlaveSelectEnable	Enables Master mode slave select.
	PLIB_SPI_StopInIdleDisable	Continues module operation when the device enters Idle mode.
	PLIB_SPI_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.

Receiver Functions

	Name	Description
	PLIB_SPI_ReceiverBufferIsFull	Returns the current status of the SPI receive buffer.
	PLIB_SPI_ReceiverFIFOIsEmpty	Returns the current status of the SPI receive FIFO.
	PLIB_SPI_ReceiverHasOverflowed	Returns the current status of the SPI receiver overflow.
	PLIB_SPI_ReceiverOverflowClear	Clears the SPI receive overflow flag.

Transmitter Functions

	Name	Description
	PLIB_SPI_TransmitBufferIsEmpty	Returns the current status of the transmit buffer.
	PLIB_SPI_TransmitBufferIsFull	Returns the current transmit buffer status of the SPI module.
	PLIB_SPI_TransmitUnderRunStatusGet	Returns the current status of the transmit underrun.

Description

This section lists and describes the functions, data types, and constants provided in the SPI Peripheral Library.

5.6.22.5.1 General Configuration and Status Functions

5.6.22.5.1.1 PLIB_SPI_BaudRateClockSelect Function

C

```
void PLIB_SPI_BaudRateClockSelect(
    SPI_MODULE_ID index,
    SPI_BAUD_RATE_CLOCK type
);
```

Description

This function selects the type of clock is used by the Baud Rate Generator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
type	One of the SPI_BAUD_RATE_CLOCK enumeration values as the SPI baud clock

Returns

None.

Remarks

This function implements an operation of the baud rate clock control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsBaudRateClock](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_BaudRateClockSelect (MY_SPI_INSTANCE, SPI_BAUD_RATE_MCLK_CLOCK);
```

5.6.22.5.1.2 PLIB_SPI_BaudRateSet Function**C**

```
void PLIB_SPI_BaudRateSet(
    SPI_MODULE_ID index,
    uint32_t clockFrequency,
    uint32_t baudRate
);
```

Description

This function sets the baud rate to the desired value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clockFrequency	Clock frequency
baudrate	Baud rate value

Returns

None.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receive error or data loss may result. To avoid this issue, verify that no receptions are in progress before changing the system clock.

This function implements an operation of the baud rate set feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsBaudRate](#)" in your application to automatically

determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_BaudRateSet(MY_SPI_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
```

5.6.22.5.1.3 PLIB_SPI_ClockPolaritySelect Function**C**

```
void PLIB_SPI_ClockPolaritySelect(  
    SPI_MODULE_ID index,  
    SPI_CLOCK_POLARITY polarity  
);
```

Description

This function enables clock polarity.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
polarity	One of the SPI_CLOCK_POLARITY enumeration values as the SPI clock polarity

Returns

None.

Remarks

This function implements an operation of the clock polarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsClockPolarity](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_ClockPolaritySelect(MY_SPI_INSTANCE, SPI_CLOCK_POLARITY_IDLE_HIGH );
```

5.6.22.5.1.4 PLIB_SPI_CommunicationWidthSelect Function**C**

```
void PLIB_SPI_CommunicationWidthSelect(  
    SPI_MODULE_ID index,  
    SPI_COMMUNICATION_WIDTH width  
);
```

Description

This function selects the data width for the SPI communication.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

width	One of the SPI_COMMUNICATION_WIDTH enumeration values as the SPI buffer width

Returns

None.

Remarks

This function implements an operation of the communication width feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsCommunicationWidth](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_CommunicationWidthSelect(MY_SPI_INSTANCE, SPI_COMMUNICATION_WIDTH_8BITS);
```

5.6.22.5.1.5 PLIB_SPI_Disable Function

C

```
void PLIB_SPI_Disable(  
    SPI_MODULE_ID index  
);
```

Description

This function disables the SPI module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsEnableControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_Disable(MY_SPI_INSTANCE);
```

5.6.22.5.1.6 PLIB_SPI_Enable Function

C

```
void PLIB_SPI_Enable(  
    SPI_MODULE_ID index  
);
```

Description

This function enables the SPI module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

The SCKx, SDOx, SDIx and SSx pins must be assigned to available RPn pins before use.

This function implements an operation of the enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsEnableControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_Enable(MY_SPI_INSTANCE);
```

5.6.22.5.1.7 PLIB_SPI_ErrorInterruptDisable Function**C**

```
void PLIB_SPI_ErrorInterruptDisable(
    SPI_MODULE_ID index,
    SPI_ERROR_INTERRUPT error
);
```

Description

This function enables SPI error interrupts.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
error	One of the SPI_ERROR_INTERRUPT enumeration values as the SPI interrupt error

Returns

None.

Remarks

This function implements an operation of the error interrupt control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsErrorInterruptControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ErrorInterruptDisable (MY_SPI_INSTANCE, SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW);
```

5.6.22.5.1.8 PLIB_SPI_ErrorInterruptEnable Function

C

```
void PLIB_SPI_ErrorInterruptEnable(  
    SPI_MODULE_ID index,  
    SPI_ERROR_INTERRUPT error  
);
```

Description

This function enables SPI error interrupts.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
type	One of the SPI_ERROR_INTERRUPT enumeration values as the SPI interrupt error

Returns

None.

Remarks

This function implements an operation of the error interrupt control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsErrorInterruptControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_ErrorInterruptEnable (MY_SPI_INSTANCE, SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW);
```

5.6.22.5.1.9 PLIB_SPI_FIFOCountGet Function

C

```
uint8_t PLIB_SPI_FIFOCountGet(  
    SPI_MODULE_ID index,  
    SPI_FIFO_TYPE type  
);
```

Description

This function reads the number of SPI transfers pending for Master mode and the number of unread SPI transfers for Slave mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
type	One of the SPI_FIFO_TYPE enumeration values

Returns

CountValue - Buffer element count bits

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFIFOCount](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
uint8_t count = PLIB_SPI_FIFOCountGet(MY_SPI_INSTANCE, SPI_FIFO_TYPE_TRANSMIT);
```

5.6.22.5.1.10 PLIB_SPI_FIFODisable Function**C**

```
void PLIB_SPI_FIFODisable(
    SPI_MODULE_ID index
);
```

Description

This function disables the SPI enhanced buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Enables the legacy standard single buffer mode.

This function implements an operation of the FIFO control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFIFOControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FIFODisable(MY_SPI_INSTANCE);
```

5.6.22.5.1.11 PLIB_SPI_FIFOEnable Function**C**

```
void PLIB_SPI_FIFOEnable(
    SPI_MODULE_ID index
);
```

Description

This function enables the SPI enhanced buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This enables the enhanced buffer mode.

This function implements an operation of the FIFO control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFIFOControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FIFOEnable(MY_SPI_INSTANCE);
```

5.6.22.5.1.12 PLIB_SPI_FIFOInterruptModeSelect Function**C**

```
void PLIB_SPI_FIFOInterruptModeSelect(
    SPI_MODULE_ID index,
    SPI_FIFO_INTERRUPT mode
);
```

Description

This function selects the SPI buffer interrupt mode from [SPI_FIFO_INTERRUPT](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the SPI_FIFO_INTERRUPT enumeration values as the SPI buffer interrupt mode

Returns

None.

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO interrupt feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFIFOInterruptMode](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
```

```
PLIB_SPI_FIFOInterruptModeSelect(MY_SPI_INSTANCE,  
    SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_NOT_FULL);
```

5.6.22.5.1.13 PLIB_SPI_FIFOShiftRegisterIsEmpty Function

C

```
bool PLIB_SPI_FIFOShiftRegisterIsEmpty(  
    SPI_MODULE_ID index  
);
```

Description

This function returns the current status of the SPI shift register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - SPI shift register is empty and ready to [send](#) or receive
- false - SPI shift register is not empty

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
bool Status = PLIB_SPI_FIFOShiftRegisterIsEmpty(MY_SPI_INSTANCE);
```

5.6.22.5.1.14 PLIB_SPI_InputSamplePhaseSelect Function

C

```
void PLIB_SPI_InputSamplePhaseSelect(  
    SPI_MODULE_ID index,  
    SPI_INPUT_SAMPLING_PHASE phase  
);
```

Description

This function selects the input sampling phase in Master mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
phase	One of the SPI_INPUT_SAMPLING_PHASE as the SPI input sampling phase

Returns

None.

Remarks

This function implements an operation of the input sample phase feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsInputSamplePhase](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_INSTANCE, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
```

5.6.22.5.1.15 PLIB_SPI_IsBusy Function**C**

```
bool PLIB_SPI_IsBusy(  
    SPI_MODULE_ID index  
);
```

Description

This function returns the current SPI module activity status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - SPI module is currently busy with some transactions
- false - SPI module is currently idle

Remarks

This function implements an operation of the bus status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsBusStatus](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
bool status = PLIB_SPI_IsBusy(MY_SPI_INSTANCE);
```

5.6.22.5.1.16 PLIB_SPI_MasterEnable Function**C**

```
void PLIB_SPI_MasterEnable(  
    SPI_MODULE_ID index  
);
```

Description

This function enables the SPI in Master mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the master enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsMasterControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_MasterEnable(MY_SPI_INSTANCE);
```

5.6.22.5.1.17 PLIB_SPI_OutputDataPhaseSelect Function**C**

```
void PLIB_SPI_OutputDataPhaseSelect(
    SPI_MODULE_ID index,
    SPI_OUTPUT_DATA_PHASE phase
);
```

Description

This function selects serial output data change.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	One of the SPI_OUTPUT_DATA_PHASE enumeration values as the SPI serial output data change

Returns

None.

Remarks

This function implements an operation of the output data phase feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsOutputDataPhase](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_INSTANCE, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
```

5.6.22.5.1.18 PLIB_SPI_PinDisable Function

C

```
void PLIB_SPI_PinDisable(  
    SPI_MODULE_ID index,  
    SPI_PIN pin  
);
```

Description

This function enables the selected SPI pins.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	One of the SPI_PIN enumeration values as the SPI pin

Returns

None.

Remarks

This function implements an operation of the pin control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsPinControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_PinDisable(MY_SPI_INSTANCE, SPI_PIN_SLAVE_SELECT);
```

5.6.22.5.1.19 PLIB_SPI_PinEnable Function

C

```
void PLIB_SPI_PinEnable(  
    SPI_MODULE_ID index,  
    SPI_PIN pin  
);
```

Description

This function enables the selected SPI pins.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	One of the SPI_PIN enumeration values as the SPI pin

Returns

None.

Remarks

This function implements an operation of the pin control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsPinControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_PinEnable(MY_SPI_INSTANCE, SPI_PIN_SLAVE_SELECT);
```

5.6.22.5.1.20 PLIB_SPI_PrescalePrimarySelect Function**C**

```
void PLIB_SPI_PrescalePrimarySelect(
    SPI_MODULE_ID index,
    SPI_PRESCALE_PRIMARY prescale
);
```

Description

This function selects the primary prescale for SPI Master mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
prescale	One of the SPI_PRIMARY_PRESCALE enumeration values as the SPI master primary prescale

Returns

None.

Remarks

This function implements an operation of the primary prescale select feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsPrimaryPrescale](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_PrescalePrimarySelect(MY_SPI_INSTANCE,
                                SPI_PRIMARY_PRESCALE_4);
```

5.6.22.5.1.21 PLIB_SPI_PrescaleSecondarySelect Function**C**

```
void PLIB_SPI_PrescaleSecondarySelect(
    SPI_MODULE_ID index,
    SPI_PRESCALE_SECONDARY prescale
);
```

Description

This function selects the secondary prescale for SPI Master mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
prescale	One of the SPI_SECONDARY_PRESCALE enumeration values as the SPI master secondary prescale

Returns

None.

Remarks

This function implements an operation of the secondary prescale select feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsSecondaryPrescale](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_PrescaleSecondarySelect( MY_SPI_INSTANCE,
                                  SPI_SECONDARY_PRESCALE_3);
```

5.6.22.5.1.22 PLIB_SPI_ReadDataIsSignExtended Function**C**

```
bool PLIB_SPI_ReadDataIsSignExtended(
    SPI_MODULE_ID index
);
```

Description

This function returns the current status of the receive (RX) FIFO sign-extended data.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Data from RX FIFO is sign-extended.
- false - Data from RX FIFO is not sign-extended

Remarks

This function implements an operation of the data sign feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsReadDataSignStatus](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_ReadDataIsSignExtended(MY_SPI_INSTANCE);
```


5.6.22.5.1.23 PLIB_SPI_SlaveEnable Function

C

```
void PLIB_SPI_SlaveEnable(  
    SPI_MODULE_ID index  
);
```

Description

This function enables the SPI in Slave mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the master enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsMasterControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_SlaveEnable(MY_SPI_INSTANCE);
```

5.6.22.5.1.24 PLIB_SPI_SlaveSelectDisable Function

C

```
void PLIB_SPI_SlaveSelectDisable(  
    SPI_MODULE_ID index  
);
```

Description

This function disables Master mode slave select.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature does not support Framed SPI mode.

This function implements an operation of the slave select feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsSlaveSelectControl](#)" in your application to

automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_SlaveSelectDisable(MY_SPI_INSTANCE);
```

5.6.22.5.1.25 PLIB_SPI_SlaveSelectEnable Function**C**

```
void PLIB_SPI_SlaveSelectEnable(  
    SPI_MODULE_ID index  
);
```

Description

This function enables Master mode slave select.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature does not support Framed SPI mode.

This function implements an operation of the slave select feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsSlaveSelectControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_SlaveSelectEnable(MY_SPI_INSTANCE);
```

5.6.22.5.1.26 PLIB_SPI_StopInIdleDisable Function**C**

```
void PLIB_SPI_StopInIdleDisable(  
    SPI_MODULE_ID index  
);
```

Description

This function sets up the SPI module such that module operation is continued when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the stop in idle control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsStopInIdleControl](#)" in your application to determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_StopInIdleDisable(MY_SPI_INSTANCE);
```

5.6.22.5.1.27 PLIB_SPI_StopInIdleEnable Function**C**

```
void PLIB_SPI_StopInIdleEnable(
    SPI_MODULE_ID index
);
```

Description

This function sets up the SPI module such that module operation is disabled when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the stop in idle control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsStopInIdleControl](#)" in your application to automatically determine if this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_StopInIdleEnable(MY_SPI_INSTANCE);
```

5.6.22.5.2 Data Transfer Functions**5.6.22.5.2.1 PLIB_SPI_BufferClear Function****C**

```
void PLIB_SPI_BufferClear(
    SPI_MODULE_ID index
);
```

Description

This function clears the SPI buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsBuffer](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_BufferClear(MY_SPI_INSTANCE);
```

5.6.22.5.2.2 PLIB_SPI_BufferRead Function**C**

```
SPI_DATA_TYPE PLIB_SPI_BufferRead(  
    SPI_MODULE_ID index  
);
```

Description

This function returns the SPI buffer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Returns the [SPI_DATA_TYPE](#) enumeration value.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsBuffer](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
uint8_t bufferValue = PLIB_SPI_BufferRead(MY_SPI_INSTANCE);
```

5.6.22.5.2.3 PLIB_SPI_BufferWrite Function**C**

```
void PLIB_SPI_BufferWrite(  
    SPI_MODULE_ID index,
```

```
    SPI_DATA_TYPE data  
);
```

Description

This function writes data to the SPI buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data of the type SPI_DATA_TYPE to written to the SPI buffer

Returns

None.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsBuffer](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_BufferWrite (MY_SPI_INSTANCE, 0xFF);
```

5.6.22.5.3 Framed Mode Functions

5.6.22.5.3.1 PLIB_SPI_FramedCommunicationDisable Function

C

```
void PLIB_SPI_FramedCommunicationDisable(  
    SPI_MODULE_ID index  
);
```

Description

This function disables framed SPI support.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFramedCommunication](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_FramedCommunicationDisable(MY_SPI_INSTANCE);
```

5.6.22.5.3.2 PLIB_SPI_FramedCommunicationEnable Function**C**

```
void PLIB_SPI_FramedCommunicationEnable(  
    SPI_MODULE_ID index  
);
```

Description

This function enables framed SPI support.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFramedCommunication](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_FramedCommunicationEnable(MY_SPI_INSTANCE);
```

5.6.22.5.3.3 PLIB_SPI_FrameErrorStatusGet Function**C**

```
bool PLIB_SPI_FrameErrorStatusGet(  
    SPI_MODULE_ID index  
);
```

Description

This function returns the current status of the SPI frame error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Frame error detected
- false - No frame error detected

Remarks

Valid only if Frame mode is enabled.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFrameErrorStatus](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
bool status = PLIB_SPI_FrameErrorStatusGet(MY_SPI_INSTANCE);
```

5.6.22.5.3.4 PLIB_SPI_FrameSyncPulseCounterSelect Function**C**

```
void PLIB_SPI_FrameSyncPulseCounterSelect(  
    SPI_MODULE_ID index,  
    SPI_FRAME_SYNC_PULSE pulse  
);
```

Description

This function selects at which character the SPI frame sync pulse is generated.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pulse	One of the SPI_FRAME_SYNC_PULSE enumeration values as the SPI frame sync pulse count

Returns

None.

Remarks

This is valid only when [PLIB_SPI_FramedCommunicationEnable](#) is enabled.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFrameSyncPulseCounter](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_INSTANCE,  
    SPI_FRAME_SYNC_PULSE_ON_EVERY_32_DATA_CHARACTER );
```

5.6.22.5.3.5 PLIB_SPI_FrameSyncPulseDirectionSelect Function**C**

```
void PLIB_SPI_FrameSyncPulseDirectionSelect(  
    SPI_MODULE_ID index,  
    SPI_FRAME_PULSE_DIRECTION direction  
);
```

Description

This function selects the frame sync pulse direction.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
direction	One of the SPI_FRAME_PULSE_DIRECTION enumeration values as the SPI frame sync pulse polarity

Returns

None.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFrameSyncPulseDirection](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_INSTANCE, SPI_FRAME_PULSE_DIRECTION_INPUT );
```

5.6.22.5.3.6 PLIB_SPI_FrameSyncPulseEdgeSelect Function**C**

```
void PLIB_SPI_FrameSyncPulseEdgeSelect(  
    SPI_MODULE_ID index,  
    SPI_FRAME_PULSE_EDGE edge  
);
```

Description

This function selects the frame sync pulse edge.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
edge	One of the SPI_FRAME_PULSE_EDGE enumeration values as the SPI frame sync pulse edge

Returns

None.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFrameSyncPulseEdge](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_FrameSyncPulseEdgeSelect(MY_SPI_INSTANCE,  
    SPI_FRAME_PULSE_EDGE_COINCIDES_FIRST_BIT_CLOCK);
```

5.6.22.5.3.7 PLIB_SPI_FrameSyncPulsePolaritySelect Function**C**

```
void PLIB_SPI_FrameSyncPulsePolaritySelect(  
    SPI_MODULE_ID index,  
    SPI_FRAME_PULSE_POLARITY polarity  
);
```

Description

This function selects the frame sync pulse polarity.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
polarity	One of the SPI_FRAME_PULSE_POLARITY enumeration values as the SPI frame sync pulse polarity

Returns

None.

Remarks

Available only for Frame mode.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFrameSyncPulsePolarity](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_FrameSyncPulsePolaritySelect(MY_SPI_INSTANCE, SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH );
```

5.6.22.5.3.8 PLIB_SPI_FrameSyncPulseWidthSelect Function**C**

```
void PLIB_SPI_FrameSyncPulseWidthSelect(  
    SPI_MODULE_ID index,  
    SPI_FRAME_PULSE_WIDTH width  
);
```

Description

This function sets the frame sync pulse width.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
edge	One of the SPI_FRAME_PULSE_WIDTH enumeration values as the SPI frame sync pulse width.

Returns

None.

Remarks

Length of the word is dependent on the communication mode.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsFrameSyncPulseWidth](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_FrameSyncPulseWidthSelect (MY_SPI_INSTANCE, SPI_FRAME_PULSE_WIDTH_ONE_WORD_LENGTH);
```

5.6.22.5.4 Audio Mode Functions

5.6.22.5.4.1 PLIB_SPI_AudioCommunicationWidthSelect Function

C

```
void PLIB_SPI_AudioCommunicationWidthSelect(  
    SPI_MODULE_ID index,  
    SPI_AUDIO_COMMUNICATION_WIDTH mode  
);
```

Description

This function selects the data width for the SPI audio communication.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
width	One of the SPI_AUDIO_COMMUNICATION_WIDTH enumeration values as the SPI buffer width

Returns

None.

Remarks

This mode is available only when [PLIB_SPI_AudioProtocolEnable](#) is enabled.

This function implements an operation of the audio communication width feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsAudioCommunicationWidth](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_AudioCommunicationWidthSelect(MY_SPI_INSTANCE,  
                                         SPI_AUDIO_COMMUNICATION_32DATA_32FIFO_32CHANNEL);
```

5.6.22.5.4.2 PLIB_SPI_AudioErrorDisable Function**C**

```
void PLIB_SPI_AudioErrorDisable(  
    SPI_MODULE_ID index,  
    SPI_AUDIO_ERROR error  
);
```

Description

This function disables the SPI error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
error	One of the SPI_AUDIO_ERROR enumeration values as the SPI error

Returns

None.

Remarks

This function implements an operation of the audio error control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsAudioErrorControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_AudioErrorDisable (MY_SPI_INSTANCE, SPI_AUDIO_ERROR_RECEIVE_OVERFLOW);
```

5.6.22.5.4.3 PLIB_SPI_AudioErrorEnable Function**C**

```
void PLIB_SPI_AudioErrorEnable(  
    SPI_MODULE_ID index,  
    SPI_AUDIO_ERROR error  
);
```

Description

This function enables the SPI error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

error	One of the SPI_AUDIO_ERROR enumeration values as the SPI error

Returns

None.

Remarks

This function implements an operation of the audio error control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsAudioErrorControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_AudioErrorEnable (MY_SPI_INSTANCE, SPI_AUDIO_ERROR_RECEIVE_OVERFLOW);
```

5.6.22.5.4.4 PLIB_SPI_AudioProtocolDisable Function

C

```
void PLIB_SPI_AudioProtocolDisable(  
    SPI_MODULE_ID index  
);
```

Description

This function disables the audio protocol.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the audio protocol control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsAudioProtocolControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_AudioProtocolDisable(MY_SPI_INSTANCE);
```

5.6.22.5.4.5 PLIB_SPI_AudioProtocolEnable Function

C

```
void PLIB_SPI_AudioProtocolEnable(  
    SPI_MODULE_ID index  
);
```

Description

This function enables the audio protocol.

Preconditions

Disable the SPI module by calling [PLIB_SPI_Disable](#).

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the audio protocol control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsAudioProtocolControl](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_AudioProtocolEnable(MY_SPI_INSTANCE);
```

5.6.22.5.4.6 PLIB_SPI_AudioProtocolModeSelect Function**C**

```
void PLIB_SPI_AudioProtocolModeSelect(  
    SPI_MODULE_ID index,  
    SPI_AUDIO_PROTOCOL mode  
);
```

Description

This function selects the Audio Protocol mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the SPI_AUDIO_PROTOCOL enumeration values as the audio protocol

Returns

None.

Remarks

This function implements an operation of the audio protocol mode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsAudioProtocolMode](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_AudioProtocolModeSelect(MY_SPI_INSTANCE, SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED );
```

5.6.22.5.4.7 PLIB_SPI_AudioTransmitModeSelect Function

C

```
void PLIB_SPI_AudioTransmitModeSelect(  
    SPI_MODULE_ID index,  
    SPI_AUDIO_TRANSMIT_MODE mode  
);
```

Description

This function selects the transmit audio data format.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the SPI_TRANSMIT_AUDIO_MODE enumeration values as the transmit audio format

Returns

None.

Remarks

This function implements an operation of the audio transmit mode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsAudioTransmitMode](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
PLIB_SPI_AudioTransmitModeSelect (MY_SPI_INSTANCE, SPI_AUDIO_TRANSMIT_MONO);
```

5.6.22.5.5 Transmitter Functions

5.6.22.5.5.1 PLIB_SPI_TransmitBufferIsEmpty Function

C

```
bool PLIB_SPI_TransmitBufferIsEmpty(  
    SPI_MODULE_ID index  
);
```

Description

This function returns the current status of the transmit buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Transmit buffer is empty
- false - Transmit buffer is not empty

Remarks

This function implements an operation of the transmit buffer empty status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsTransmitBufferEmptyStatus](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_TransmitBufferIsEmpty(MY_SPI_INSTANCE);
```

5.6.22.5.5.2 PLIB_SPI_TransmitBufferIsFull Function**C**

```
bool PLIB_SPI_TransmitBufferIsFull(
    SPI_MODULE_ID index
);
```

Description

This function returns the current transmit buffer status of the SPI module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Transmit not yet started, transmit buffer is full
- false - Transmit started, transmit buffer is empty/not full

Remarks

In Standard Buffer mode - automatically set in hardware when SPI buffer writes occur, loading the transmit buffer. Automatically cleared in hardware when the SPI module transfers data from the transmit buffer to the shift register.

In Enhanced Buffer mode - automatically set in hardware when SPI buffer writes occur, loading the last available buffer. Automatically cleared in hardware when the buffer is available for writing.

This function implements an operation of the transmit buffer status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsTransmitBufferFullStatus](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool buffullstate = PLIB_SPI_TransmitBufferIsFull (MY_SPI_INSTANCE);
```

5.6.22.5.5.3 PLIB_SPI_TransmitUnderRunStatusGet Function**C**

```
bool PLIB_SPI_TransmitUnderRunStatusGet(
```

```
SPI_MODULE_ID index  
);
```

Description

This function returns the current status of the transmit underrun.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Transmit buffer has encountered an underrun condition
- false - Transmit buffer run has not encountered an underrun condition

Remarks

Valid in Framed Sync mode.

This function implements an operation of the transmit underrun status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsTransmitUnderRunStatus](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
bool Status = PLIB_SPI_TransmitUnderRunStatusGet(MY_SPI_INSTANCE);
```

5.6.22.5.6 Receiver Functions

5.6.22.5.6.1 PLIB_SPI_ReceiverBufferIsFull Function

C

```
bool PLIB_SPI_ReceiverBufferIsFull(  
    SPI_MODULE_ID index  
);
```

Description

This function returns the current status of the SPI receive buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Status - Receiver Buffer Full Status true - Receive complete, receive buffer is full false - Receive is not complete, receive buffer is empty

Remarks

In Standard Buffer mode - automatically set in hardware when the SPI module transfers data from the shift register to the receive

buffer. Automatically cleared in hardware when the core reads the SPI buffer, read in the receive buffer.

In Enhanced Buffer mode - automatically set in hardware when the SPI module transfers data from the shift register to the receive buffer, filling the last unread buffer. Automatically cleared in hardware when a buffer is available for a transfer from the shift register.

This function implements an operation of the receiver buffer status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsReceiveBufferStatus](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
bool receivefullstate = PLIB_SPI_ReceiverBufferIsFull (MY_SPI_INSTANCE);
```

5.6.22.5.6.2 PLIB_SPI_ReceiverFIFOIsEmpty Function

C

```
bool PLIB_SPI_ReceiverFIFOIsEmpty(  
    SPI_MODULE_ID index  
);
```

Description

This function returns the current status of the SPI receive FIFO.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Receive FIFO is empty
- false - Receive FIFO is not empty

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsReceiveFIFOStatus](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the  
// application developer.  
bool fifostate = PLIB_SPI_ReceiverFIFOIsEmpty (MY_SPI_INSTANCE);
```

5.6.22.5.6.3 PLIB_SPI_ReceiverHasOverflowed Function

C

```
bool PLIB_SPI_ReceiverHasOverflowed(  
    SPI_MODULE_ID index  
);
```

Description

This function returns the current status of the SPI receiver overflow.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Status - SPI receiver overflow status true - A new byte/word is completely received and discarded. The user software has not read the previous data in the SPI buffer register. false - No Overflow has occurred

Remarks

This function implements an operation of the receiver overflow status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsReceiverOverflow](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool overflowstate = PLIB_SPI_ReceiverHasOverflowed(MY_SPI_INSTANCE);
```

5.6.22.5.6.4 PLIB_SPI_ReceiverOverflowClear Function**C**

```
void PLIB_SPI_ReceiverOverflowClear(
    SPI_MODULE_ID index
);
```

Description

This function clears the SPI receive overflow flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function implements an operation of the receiver overflow status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "[PLIB_SPI_ExistsReceiverOverflow](#)" in your application to automatically determine whether this feature is available.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ReceiverOverflowClear(MY_SPI_INSTANCE);
```

5.6.22.5.7 Data Types and Constants

5.6.22.5.7.1 SPI_AUDIO_COMMUNICATION_WIDTH Enumeration

C

```
typedef enum {
    SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL,
    SPI_AUDIO_COMMUNICATION_32DATA_32FIFO_32CHANNEL,
    SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL,
    SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_16CHANNEL
} SPI_AUDIO_COMMUNICATION_WIDTH;
```

Description

SPI Audio Communication Width

This macro defines the list of SPI audio communication width.

Members

Members	Description
SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL	Communication is 24-bit Data,32-bit FIFO,32-bit Channel/64-bit Frame
SPI_AUDIO_COMMUNICATION_32DATA_32FIFO_32CHANNEL	Communication is 32-bit Data,32-bit FIFO,32-bit Channel/64-bit Frame
SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL	Communication is 16-bit Data,16-bit FIFO,32-bit Channel/64-bit Frame
SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_16CHANNEL	Communication is 16-bit Data,16-bit FIFO,16-bit Channel/64-bit Frame

5.6.22.5.7.2 SPI_AUDIO_ERROR Enumeration

C

```
typedef enum {
    SPI_AUDIO_ERROR_RECEIVE_OVERFLOW,
    SPI_AUDIO_ERROR_TRANSMIT_UNDERRUN
} SPI_AUDIO_ERROR;
```

Description

SPI Audio Error

This macro defines the list of SPI audio error.

Members

Members	Description
SPI_AUDIO_ERROR_RECEIVE_OVERFLOW	SPI error for receive overflow
SPI_AUDIO_ERROR_TRANSMIT_UNDERRUN	SPI error for transmit underrun

5.6.22.5.7.3 SPI_AUDIO_PROTOCOL Enumeration

C

```
typedef enum {
    SPI_AUDIO_PROTOCOL_PCM_DSP,
    SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED,
    SPI_AUDIO_PROTOCOL_LEFT_JUSTIFIED,
    SPI_AUDIO_PROTOCOL_I2S
} SPI_AUDIO_PROTOCOL;
```

Description

Audio Protocol Mode enumeration

This data type defining the audio protocol mode.

Members

Members	Description
SPI_AUDIO_PROTOCOL_PCM_DSP	Audio protocol set to PCM/DSP mode
SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED	Audio protocol set to Right-Justified mode
SPI_AUDIO_PROTOCOL_LEFT_JUSTIFIED	Audio protocol set to Left-Justified mode
SPI_AUDIO_PROTOCOL_I2S	Audio protocol set to I2C mode

Remarks

This enumeration is processor specific.

5.6.22.5.7.4 SPI_AUDIO_TRANSMIT_MODE Enumeration

C

```
typedef enum {
    SPI_AUDIO_TRANSMIT_MONO,
    SPI_AUDIO_TRANSMIT_STEREO
} SPI_AUDIO_TRANSMIT_MODE;
```

Description

Audio Transmit mode format

This macro defines the list of SPI transmit audio mode format.

Members

Members	Description
SPI_AUDIO_TRANSMIT_MONO	SPI Transmit Audio Data Format is Mono
SPI_AUDIO_TRANSMIT_STEREO	SPI Transmit Audio Data Format is Stereo

5.6.22.5.7.5 SPI_BAUD_RATE_CLOCK Enumeration

C

```
typedef enum {
    SPI_BAUD_RATE_MCLK_CLOCK,
    SPI_BAUD_RATE_PBCLK_CLOCK
} SPI_BAUD_RATE_CLOCK;
```

Description

SPI Baudrate generator

This macro defines the list of the SPI Baud Rate Generator.

Members

Members	Description
SPI_BAUD_RATE_MCLK_CLOCK	MCLK is used by the Baud Rate Generator
SPI_BAUD_RATE_PBCLK_CLOCK	Peripheral bus clock is used by the Baud Rate Generator

5.6.22.5.7.6 SPI_CLOCK_POLARITY Enumeration

C

```
typedef enum {
    SPI_CLOCK_POLARITY_IDLE_HIGH,
    SPI_CLOCK_POLARITY_IDLE_LOW
}
```

```
} SPI_CLOCK_POLARITY;
```

Description

SPI Clock polarity

This macro defines the list of SPI clock polarity.

Members

Members	Description
SPI_CLOCK_POLARITY_IDLE_HIGH	Idle state for clock is a high level;active state is a low level
SPI_CLOCK_POLARITY_IDLE_LOW	Idle state for clock is a low level;active state is a high level

5.6.22.5.7.7 SPI_COMMUNICATION_WIDTH Enumeration

C

```
typedef enum {  
    SPI_COMMUNICATION_WIDTH_32BITS,  
    SPI_COMMUNICATION_WIDTH_16BITS,  
    SPI_COMMUNICATION_WIDTH_8BITS  
} SPI_COMMUNICATION_WIDTH;
```

Description

SPI Communication Width

This macro defines the list of SPI communication width.

Members

Members	Description
SPI_COMMUNICATION_WIDTH_32BITS	Communication is 32-bit-wide
SPI_COMMUNICATION_WIDTH_16BITS	Communication is word-wide
SPI_COMMUNICATION_WIDTH_8BITS	Communication is byte-wide

5.6.22.5.7.8 SPI_DATA_TYPE Type

C

```
typedef unsigned char SPI_DATA_TYPE;
```

Description

SPI Data Type definition

This data type defines the SPI data size.

Remarks

None

5.6.22.5.7.9 SPI_ERROR_INTERRUPT Enumeration

C

```
typedef enum {  
    SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW,  
    SPI_ERROR_INTERRUPT_RECEIVE_OVERFLOW,  
    SPI_ERROR_INTERRUPT_TRANSMIT_UNDERRUN  
} SPI_ERROR_INTERRUPT;
```

Description

SPI Error Interrupt

This macro defines the list of SPI error interrupts.

Members

Members	Description
SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW	SPI interrupt for fram error
SPI_ERROR_INTERRUPT_RECEIVE_OVERFLOW	SPI interrupt for receive overflow error
SPI_ERROR_INTERRUPT_TRANSMIT_UNDERRUN	SPI interrupt for transmit underrun error

5.6.22.5.7.10 SPI_FIFO_INTERRUPT Enumeration

C

```
typedef enum {
    SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_NOT_FULL,
    SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_1HALF_EMPTY_OR_MORE,
    SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_COMPLETELY_EMPTY,
    SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_1HALF_FULL_OR_MORE,
    SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_NOT_EMPTY,
    SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_FULL,
    SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_FIFO_IS_EMPTY,
    SPI_FIFO_INTERRUPT_WHEN_TRANSMISSION_IS_COMPLETE,
    SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_FIFO_HAS_ONE_SLOT,
    SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_FULL,
    SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_3FOURTH_OR_MORE_FULL,
    SPI_FIFO_INTERRUPT_WHEN_DATA_IS_IN_RECEIVE_BUFFER,
    SPI_FIFO_INTERRUPT_WHEN_BUFFER_IS_EMPTY
} SPI_FIFO_INTERRUPT;
```

Description

SPI Buffer Interrupt Mode

This macro defines the list of SPI Buffer Interrupt mode.

Members

Members	Description
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_NOT_FULL	Interrupt when the transmit buffer is not full
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_1HALF_EMPTY_OR_MORE	Interrupt when the transmit buffer is half empty
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_COMPLETELY_EMPTY	Interrupt when the transmit buffer is empty
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_1HALF_FULL_OR_MORE	Interrupt when the receive buffer half full or more
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_NOT_EMPTY	Interrupt when the receive buffer is not empty
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_FULL	Interrupt when transmit buffer is full
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_FIFO_IS_EMPTY	Interrupt when transmit buffer is empty
SPI_FIFO_INTERRUPT_WHEN_TRANSMISSION_IS_COMPLETE	Interrupt when transmission is complete
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_FIFO_HAS_ONE_SLOT	Interrupt when one slot is left in transmit buffer
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_FULL	Interrupt when receive buffer is full
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_3FOURTH_OR_MORE_FULL	Interrupt when receive buffer is 3/4 or more full
SPI_FIFO_INTERRUPT_WHEN_DATA_IS_IN_RECEIVE_BUFFER	Interrupt when data is available in receive buffer

SPI_FIFO_INTERRUPT_WHEN_BUFFER_IS_EMPTY	Interrupt when the receive buffer is empty

5.6.22.5.7.11 SPI_FIFO_TYPE Enumeration

C

```
typedef enum {
    SPI_FIFO_TYPE_COMMON,
    SPI_FIFO_TYPE_TRANSMIT,
    SPI_FIFO_TYPE_RECEIVE
} SPI_FIFO_TYPE;
```

Description

SPI Buffer Mode

This macro defines the list of SPI buffer mode.

Members

Members	Description
SPI_FIFO_TYPE_COMMON	Buffer type is common type
SPI_FIFO_TYPE_TRANSMIT	Buffer type is transmit
SPI_FIFO_TYPE_RECEIVE	Buffer type is receive

5.6.22.5.7.12 SPI_FRAME_PULSE_DIRECTION Enumeration

C

```
typedef enum {
    SPI_FRAME_PULSE_DIRECTION_OUTPUT,
    SPI_FRAME_PULSE_DIRECTION_INPUT
} SPI_FRAME_PULSE_DIRECTION;
```

Description

SPI Frame sync pulse direction

This macro defines the list of SPI frame sync pulse direction.

Members

Members	Description
SPI_FRAME_PULSE_DIRECTION_OUTPUT	Frame sync pulse direction is output
SPI_FRAME_PULSE_DIRECTION_INPUT	Frame sync pulse direction is input

5.6.22.5.7.13 SPI_FRAME_PULSE_EDGE Enumeration

C

```
typedef enum {
    SPI_FRAME_PULSE_EDGE_COINCIDES_FIRST_BIT_CLOCK,
    SPI_FRAME_PULSE_EDGE_PRECEDES_FIRST_BIT_CLOCK
} SPI_FRAME_PULSE_EDGE;
```

Description

SPI Frame sync pulse edge

This macro defines the list of SPI frame sync pulse edge.

Members

Members	Description
SPI_FRAME_PULSE_EDGE_COINCIDES_FIRST_BIT_CLOCK	Frame sync pulse coincides with first bit clock
SPI_FRAME_PULSE_EDGE_PRECEDES_FIRST_BIT_CLOCK	Frame sync pulse precedes first bit clock

5.6.22.5.7.14 SPI_FRAME_PULSE_POLARITY Enumeration**C**

```
typedef enum {
    SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH,
    SPI_FRAME_PULSE_POLARITY_ACTIVE_LOW
} SPI_FRAME_PULSE_POLARITY;
```

Description

SPI Frame sync pulse polarity

This macro defines the list of SPI frame sync pulse polarity.

Members

Members	Description
SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH	Frame sync pulse is active high
SPI_FRAME_PULSE_POLARITY_ACTIVE_LOW	Frame sync pulse is active low

5.6.22.5.7.15 SPI_FRAME_PULSE_WIDTH Enumeration**C**

```
typedef enum {
    SPI_FRAME_PULSE_WIDTH_ONE_WORD_LENGTH,
    SPI_FRAME_PULSE_WIDTH_ONE_CLOCK_WIDE
} SPI_FRAME_PULSE_WIDTH;
```

Description

SPI Frame sync pulse width

This macro defines the list of SPI frame sync pulse width.

Members

Members	Description
SPI_FRAME_PULSE_WIDTH_ONE_WORD_LENGTH	Frame sync Pulse width as one word length wide
SPI_FRAME_PULSE_WIDTH_ONE_CLOCK_WIDE	Frame sync Pulse width as one clock wide

5.6.22.5.7.16 SPI_FRAME_SYNC_PULSE Enumeration**C**

```
typedef enum {
    SPI_FRAME_SYNC_PULSE_ON_EVERY_32_DATA_CHARACTER,
    SPI_FRAME_SYNC_PULSE_ON_EVERY_16_DATA_CHARACTER,
    SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER,
    SPI_FRAME_SYNC_PULSE_ON_EVERY_4_DATA_CHARACTER,
    SPI_FRAME_SYNC_PULSE_ON_EVERY_2_DATA_CHARACTER,
    SPI_FRAME_SYNC_PULSE_ON_EVERY_DATA_CHARACTER
} SPI_FRAME_SYNC_PULSE;
```

Description

Frame Sync Pulse Counter enumeration

This data type defining the frame sync pulse counter values.

Members

Members	Description
SPI_FRAME_SYNC_PULSE_ON_EVERY_32_DATA_CHARACTER	Generate a fram sync pulse on every 32 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_16_DATA_CHARACTER	Generate a fram sync pulse on every 16 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER	Generate a fram sync pulse on every 8 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_4_DATA_CHARACTER	Generate a fram sync pulse on every 4 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_2_DATA_CHARACTER	Generate a fram sync pulse on every 2 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_DATA_CHARACTER	Generate a fram sync pulse on every data character

5.6.22.5.7.17 SPI_INPUT_SAMPLING_PHASE Enumeration

C

```
typedef enum {
    SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE,
    SPI_INPUT_SAMPLING_PHASE_AT_END
} SPI_INPUT_SAMPLING_PHASE;
```

Description

SPI Data Input Sample Phase

This macro defines the list of SPI data input sample phase.

Members

Members	Description
SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE	SPI Data Input Sample phase in middle
SPI_INPUT_SAMPLING_PHASE_AT_END	SPI Data Input Sample phase at end

5.6.22.5.7.18 SPI_MODULE_ID Enumeration

C

```
typedef enum {
    SPI_ID_1,
    SPI_ID_2,
    SPI_ID_3,
    SPI_ID_4,
    SPI_NUMBER_OF_MODULES
} SPI_MODULE_ID;
```

Description

SPI Module ID

This enumeration identifies the SPI modules that are available on a microcontroller. This is the superset of all the possible instances that might be available on Microchip microcontrollers.

Members

Members	Description
SPI_ID_1	SPI Module 1 ID
SPI_ID_2	SPI Module 2 ID
SPI_ID_3	SPI Module 3 ID
SPI_ID_4	SPI Module 4 ID
SPI_NUMBER_OF_MODULES	Number of available SPI modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

5.6.22.5.7.19 SPI_OUTPUT_DATA_PHASE Enumeration**C**

```
typedef enum {
    SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK,
    SPI_OUTPUT_DATA_PHASE_ON_IDLE_TO_ACTIVE_CLOCK
} SPI_OUTPUT_DATA_PHASE;
```

Description

SPI Output Data Phase

This macro defines the list of SPI serial output data changes.

Members

Members	Description
SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK	Serial output data changes on transition from active clock state to idle clock state
SPI_OUTPUT_DATA_PHASE_ON_IDLE_TO_ACTIVE_CLOCK	Serial output data changes on transition from idle clock state to active clock state.

5.6.22.5.7.20 SPI_PIN Enumeration**C**

```
typedef enum {
    SPI_PIN_SERIAL_CLOCK,
    SPI_PIN_DATA_OUT,
    SPI_PIN_DATA_IN,
    SPI_PIN_SLAVE_SELECT
} SPI_PIN;
```

Description

SPI pin

This data type defining the SPI pin.

Members

Members	Description
SPI_PIN_SERIAL_CLOCK	SPI clock pin
SPI_PIN_DATA_OUT	SPI data output pin
SPI_PIN_DATA_IN	SPI data input pin
SPI_PIN_SLAVE_SELECT	SPI slave select pin

Remarks

This enumeration is processor specific.

5.6.22.5.7.21 SPI_PRESCALE_PRIMARY Enumeration

C

```
typedef enum {
    SPI_PRESCALE_PRIMARY_1,
    SPI_PRESCALE_PRIMARY_4,
    SPI_PRESCALE_PRIMARY_16,
    SPI_PRESCALE_PRIMARY_64
} SPI_PRESCALE_PRIMARY;
```

Description

Primary Prescaler Values

This macro defines the list of possible primary prescaler values.

Members

Members	Description
SPI_PRESCALE_PRIMARY_1	SPI Primary Prescaler Value 1:1
SPI_PRESCALE_PRIMARY_4	SPI Primary Prescaler Value 4:1
SPI_PRESCALE_PRIMARY_16	SPI Primary Prescaler Value 16:1
SPI_PRESCALE_PRIMARY_64	SPI Primary Prescaler Value 64:1

5.6.22.5.7.22 SPI_PRESCALE_SECONDARY Enumeration

C

```
typedef enum {
    SPI_PRESCALE_SECONDARY_1,
    SPI_PRESCALE_SECONDARY_2,
    SPI_PRESCALE_SECONDARY_3,
    SPI_PRESCALE_SECONDARY_4,
    SPI_PRESCALE_SECONDARY_5,
    SPI_PRESCALE_SECONDARY_6,
    SPI_PRESCALE_SECONDARY_7,
    SPI_PRESCALE_SECONDARY_8
} SPI_PRESCALE_SECONDARY;
```

Description

Secondary Prescaler Values

This macro defines the list of possible secondary prescaler values.

Members

Members	Description
SPI_PRESCALE_SECONDARY_1	SPI Secondary Prescaler Value 1:1
SPI_PRESCALE_SECONDARY_2	SPI Secondary Prescaler Value 2:1
SPI_PRESCALE_SECONDARY_3	SPI Secondary Prescaler Value 3:1
SPI_PRESCALE_SECONDARY_4	SPI Secondary Prescaler Value 4:1
SPI_PRESCALE_SECONDARY_5	SPI Secondary Prescaler Value 5:1
SPI_PRESCALE_SECONDARY_6	SPI Secondary Prescaler Value 6:1
SPI_PRESCALE_SECONDARY_7	SPI Secondary Prescaler Value 7:1
SPI_PRESCALE_SECONDARY_8	SPI Secondary Prescaler Value 8:1

5.6.22.5.8 Feature Existence Functions

5.6.22.5.8.1 PLIB_SPI_ExistsAudioCommunicationWidth Function

C

```
bool PLIB_SPI_ExistsAudioCommunicationWidth(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the AudioCommunicationWidth feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_AudioCommunicationWidthSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AudioCommunicationWidth feature is supported on the device
- false - The AudioCommunicationWidth feature is not supported on the device

Remarks

None.

5.6.22.5.8.2 PLIB_SPI_ExistsAudioErrorControl Function

C

```
bool PLIB_SPI_ExistsAudioErrorControl(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the AudioErrorControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_AudioErrorEnable](#)
- [PLIB_SPI_AudioErrorDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AudioErrorControl feature is supported on the device
- false - The AudioErrorControl feature is not supported on the device

Remarks

None.

5.6.22.5.8.3 PLIB_SPI_ExistsAudioProtocolControl Function

C

```
bool PLIB_SPI_ExistsAudioProtocolControl(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the AudioProtocolControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_AudioProtocolEnable](#)
- [PLIB_SPI_AudioProtocolDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AudioProtocolControl feature is supported on the device
- false - The AudioProtocolControl feature is not supported on the device

Remarks

None.

5.6.22.5.8.4 PLIB_SPI_ExistsAudioProtocolMode Function

C

```
bool PLIB_SPI_ExistsAudioProtocolMode(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the AudioProtocolMode feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_AudioProtocolModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AudioProtocolMode feature is supported on the device
- false - The AudioProtocolMode feature is not supported on the device

Remarks

None.

5.6.22.5.8.5 PLIB_SPI_ExistsAudioTransmitMode Function

C

```
bool PLIB_SPI_ExistsAudioTransmitMode(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the AudioTransmitMode feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_AudioTransmitModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AudioTransmitMode feature is supported on the device
- false - The AudioTransmitMode feature is not supported on the device

Remarks

None.

5.6.22.5.8.6 PLIB_SPI_ExistsBaudRate Function

C

```
bool PLIB_SPI_ExistsBaudRate(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the BaudRate feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_BaudRateSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BaudRate feature is supported on the device
- false - The BaudRate feature is not supported on the device

Remarks

None.

5.6.22.5.8.7 PLIB_SPI_ExistsBaudRateClock Function

C

```
bool PLIB_SPI_ExistsBaudRateClock(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the BaudRateClock feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_BaudRateClockSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BaudRateClock feature is supported on the device
- false - The BaudRateClock feature is not supported on the device

Remarks

None.

5.6.22.5.8.8 PLIB_SPI_ExistsBuffer Function

C

```
bool PLIB_SPI_ExistsBuffer(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the Buffer feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_BufferClear](#)
- [PLIB_SPI_BufferRead](#)
- [PLIB_SPI_BufferWrite](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Buffer feature is supported on the device
- false - The Buffer feature is not supported on the device

Remarks

None.

5.6.22.5.8.9 PLIB_SPI_ExistsBusStatus Function**C**

```
bool PLIB_SPI_ExistsBusStatus(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the BusStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_IsBusy](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BusStatus feature is supported on the device
- false - The BusStatus feature is not supported on the device

Remarks

None.

5.6.22.5.8.10 PLIB_SPI_ExistsClockPolarity Function**C**

```
bool PLIB_SPI_ExistsClockPolarity(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the ClockPolarity feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_ClockPolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ClockPolarity feature is supported on the device
- false - The ClockPolarity feature is not supported on the device

Remarks

None.

5.6.22.5.8.11 PLIB_SPI_ExistsCommunicationWidth Function**C**

```
bool PLIB_SPI_ExistsCommunicationWidth(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the CommunicationWidth feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_CommunicationWidthSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CommunicationWidth feature is supported on the device
- false - The CommunicationWidth feature is not supported on the device

Remarks

None.

5.6.22.5.8.12 PLIB_SPI_ExistsEnableControl Function**C**

```
bool PLIB_SPI_ExistsEnableControl(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_Enable](#)
- [PLIB_SPI_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnableControl feature is supported on the device

- false - The EnableControl feature is not supported on the device

Remarks

None.

5.6.22.5.8.13 PLIB_SPI_ExistsErrorInterruptControl Function**C**

```
bool PLIB_SPI_ExistsErrorInterruptControl(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the ErrorInterruptControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_ErrorInterruptEnable](#)
- [PLIB_SPI_ErrorInterruptDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ErrorInterruptControl feature is supported on the device
- false - The ErrorInterruptControl feature is not supported on the device

Remarks

None.

5.6.22.5.8.14 PLIB_SPI_ExistsFIFOControl Function**C**

```
bool PLIB_SPI_ExistsFIFOControl(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FIFOControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_FIFOEnable](#)
- [PLIB_SPI_FIFODisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FIFOControl feature is supported on the device
- false - The FIFOControl feature is not supported on the device

Remarks

None.

5.6.22.5.8.15 PLIB_SPI_ExistsFIFOCount Function**C**

```
bool PLIB_SPI_ExistsFIFOCount(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FIFOCount feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_FIFOCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FIFOCount feature is supported on the device
- false - The FIFOCount feature is not supported on the device

Remarks

None.

5.6.22.5.8.16 PLIB_SPI_ExistsFIFOInterruptMode Function**C**

```
bool PLIB_SPI_ExistsFIFOInterruptMode(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FIFOInterruptMode feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_FIFOInterruptModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FIFOInterruptMode feature is supported on the device
- false - The FIFOInterruptMode feature is not supported on the device

Remarks

None.

5.6.22.5.8.17 PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus Function**C**

```
bool PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FIFOShiftRegisterEmptyStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_FIFOShiftRegisterIsEmpty](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FIFOShiftRegisterEmptyStatus feature is supported on the device
- false - The FIFOShiftRegisterEmptyStatus feature is not supported on the device

Remarks

None.

5.6.22.5.8.18 PLIB_SPI_ExistsFramedCommunication Function**C**

```
bool PLIB_SPI_ExistsFramedCommunication(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FramedCommunication feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_FramedCommunicationEnable](#)
- [PLIB_SPI_FramedCommunicationDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FramedCommunication feature is supported on the device
- false - The FramedCommunication feature is not supported on the device

Remarks

None.

5.6.22.5.8.19 PLIB_SPI_ExistsFrameErrorStatus Function**C**

```
bool PLIB_SPI_ExistsFrameErrorStatus(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FrameErrorStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_FrameErrorStatusGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FrameErrorStatus feature is supported on the device
- false - The FrameErrorStatus feature is not supported on the device

Remarks

None.

5.6.22.5.8.20 PLIB_SPI_ExistsFrameSyncPulseCounter Function**C**

```
bool PLIB_SPI_ExistsFrameSyncPulseCounter(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FrameSyncPulseCounter feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_FrameSyncPulseCounterSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FrameSyncPulseCounter feature is supported on the device
- false - The FrameSyncPulseCounter feature is not supported on the device

Remarks

None.

5.6.22.5.8.21 PLIB_SPI_ExistsFrameSyncPulseDirection Function**C**

```
bool PLIB_SPI_ExistsFrameSyncPulseDirection(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FrameSyncPulseDirection feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_FrameSyncPulseDirectionSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FrameSyncPulseDirection feature is supported on the device
- false - The FrameSyncPulseDirection feature is not supported on the device

Remarks

None.

5.6.22.5.8.22 PLIB_SPI_ExistsFrameSyncPulseEdge Function**C**

```
bool PLIB_SPI_ExistsFrameSyncPulseEdge(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FrameSyncPulseEdge feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_FrameSyncPulseEdgeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FrameSyncPulseEdge feature is supported on the device
- false - The FrameSyncPulseEdge feature is not supported on the device

Remarks

None.

5.6.22.5.8.23 PLIB_SPI_ExistsFrameSyncPulsePolarity Function**C**

```
bool PLIB_SPI_ExistsFrameSyncPulsePolarity(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FrameSyncPulsePolarity feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_FrameSyncPulsePolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FrameSyncPulsePolarity feature is supported on the device
- false - The FrameSyncPulsePolarity feature is not supported on the device

Remarks

None.

5.6.22.5.8.24 PLIB_SPI_ExistsFrameSyncPulseWidth Function**C**

```
bool PLIB_SPI_ExistsFrameSyncPulseWidth(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the FrameSyncPulseWidth feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_FrameSyncPulseWidthSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FrameSyncPulseWidth feature is supported on the device
- false - The FrameSyncPulseWidth feature is not supported on the device

Remarks

None.

5.6.22.5.8.25 PLIB_SPI_ExistsInputSamplePhase Function**C**

```
bool PLIB_SPI_ExistsInputSamplePhase(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the InputSamplePhase feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_InputSamplePhaseSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InputSamplePhase feature is supported on the device
- false - The InputSamplePhase feature is not supported on the device

Remarks

None.

5.6.22.5.8.26 PLIB_SPI_ExistsMasterControl Function**C**

```
bool PLIB_SPI_ExistsMasterControl(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the MasterControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_MasterEnable](#)
- [PLIB_SPI_SlaveEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The MasterControl feature is supported on the device
- false - The MasterControl feature is not supported on the device

Remarks

None.

5.6.22.5.8.27 PLIB_SPI_ExistsOutputDataPhase Function**C**

```
bool PLIB_SPI_ExistsOutputDataPhase(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the OutputDataPhase feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_OutputDataPhaseSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OutputDataPhase feature is supported on the device
- false - The OutputDataPhase feature is not supported on the device

Remarks

None.

5.6.22.5.8.28 PLIB_SPI_ExistsPinControl Function**C**

```
bool PLIB_SPI_ExistsPinControl(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the PinControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_PinEnable](#)

- [PLIB_SPI_PinDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PinControl feature is supported on the device
- false - The PinControl feature is not supported on the device

Remarks

None.

5.6.22.5.8.29 PLIB_SPI_ExistsPrimaryPrescale Function

C

```
bool PLIB_SPI_ExistsPrimaryPrescale(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the PrimaryPrescale feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_PrescalePrimarySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PrimaryPrescale feature is supported on the device
- false - The PrimaryPrescale feature is not supported on the device

Remarks

None.

5.6.22.5.8.30 PLIB_SPI_ExistsReadDataSignStatus Function

C

```
bool PLIB_SPI_ExistsReadDataSignStatus(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the ReadDataSignStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_ReadDataIsSignExtended](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReadDataSignStatus feature is supported on the device
- false - The ReadDataSignStatus feature is not supported on the device

Remarks

None.

5.6.22.5.8.31 PLIB_SPI_ExistsReceiveBufferStatus Function**C**

```
bool PLIB_SPI_ExistsReceiveBufferStatus(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveBufferStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_ReceiverBufferIsFull](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiveBufferStatus feature is supported on the device
- false - The ReceiveBufferStatus feature is not supported on the device

Remarks

None.

5.6.22.5.8.32 PLIB_SPI_ExistsReceiveFIFOStatus Function**C**

```
bool PLIB_SPI_ExistsReceiveFIFOStatus(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveFIFOStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_ReceiverFIFOIsEmpty](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiveFIFOStatus feature is supported on the device
- false - The ReceiveFIFOStatus feature is not supported on the device

Remarks

None.

5.6.22.5.8.33 PLIB_SPI_ExistsReceiverOverflow Function**C**

```
bool PLIB_SPI_ExistsReceiverOverflow(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiverOverflow feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_ReceiverHasOverflowed](#)
- [PLIB_SPI_ReceiverOverflowClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverOverflow feature is supported on the device
- false - The ReceiverOverflow feature is not supported on the device

Remarks

None.

5.6.22.5.8.34 PLIB_SPI_ExistsSecondaryPrescale Function**C**

```
bool PLIB_SPI_ExistsSecondaryPrescale(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the SecondaryPrescale feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_PrescaleSecondarySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SecondaryPrescale feature is supported on the device
- false - The SecondaryPrescale feature is not supported on the device

Remarks

None.

5.6.22.5.8.35 PLIB_SPI_ExistsSlaveSelectControl Function

C

```
bool PLIB_SPI_ExistsSlaveSelectControl(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the SlaveSelectControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_SlaveSelectEnable](#)
- [PLIB_SPI_SlaveSelectDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SlaveSelectControl feature is supported on the device
- false - The SlaveSelectControl feature is not supported on the device

Remarks

None.

5.6.22.5.8.36 PLIB_SPI_ExistsStopInIdleControl Function

C

```
bool PLIB_SPI_ExistsStopInIdleControl(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- [PLIB_SPI_StopInIdleEnable](#)
- [PLIB_SPI_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Remarks

None.

5.6.22.5.8.37 PLIB_SPI_ExistsTransmitBufferEmptyStatus Function

C

```
bool PLIB_SPI_ExistsTransmitBufferEmptyStatus(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitBufferEmptyStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_TransmitBufferIsEmpty](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitBufferEmptyStatus feature is supported on the device
- false - The TransmitBufferEmptyStatus feature is not supported on the device

Remarks

None.

5.6.22.5.8.38 PLIB_SPI_ExistsTransmitBufferFullStatus Function

C

```
bool PLIB_SPI_ExistsTransmitBufferFullStatus(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitBufferFullStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_TransmitBufferIsFull](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitBufferFullStatus feature is supported on the device
- false - The TransmitBufferFullStatus feature is not supported on the device

Remarks

None.

5.6.22.5.8.39 PLIB_SPI_ExistsTransmitUnderRunStatus Function

C

```
bool PLIB_SPI_ExistsTransmitUnderRunStatus(  
    SPI_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitUnderRunStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- [PLIB_SPI_TransmitUnderRunStatusGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitUnderRunStatus feature is supported on the device
- false - The TransmitUnderRunStatus feature is not supported on the device

Remarks

None.

5.6.22.6 Files

Files

Name	Description
plib_spi.h	SPI PLIB Interface Header for common definitions

Description

5.6.22.6.1 plib_spi.h

SPI Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the SPI PLIB.

Functions

	Name	Description
≡	<u>PLIB_SPI_AudioCommunicationWidthSelect</u>	Selects the data width for the SPI audio communication.
≡	<u>PLIB_SPI_AudioErrorDisable</u>	Disables the SPI error.
≡	<u>PLIB_SPI_AudioErrorEnable</u>	Enables the SPI error.
≡	<u>PLIB_SPI_AudioProtocolDisable</u>	Audio protocol is disabled.
≡	<u>PLIB_SPI_AudioProtocolEnable</u>	Audio protocol is enabled.
≡	<u>PLIB_SPI_AudioProtocolModeSelect</u>	Selects the Audio Protocol mode.
≡	<u>PLIB_SPI_AudioTransmitModeSelect</u>	Selects the transmit audio data format.
≡	<u>PLIB_SPI_BaudRateClockSelect</u>	Selects the type of clock is used by the Baud Rate Generator.
≡	<u>PLIB_SPI_BaudRateSet</u>	Sets the baud rate to the desired value.
≡	<u>PLIB_SPI_BufferClear</u>	Clears the SPI buffer.
≡	<u>PLIB_SPI_BufferRead</u>	Returns the SPI buffer value.
≡	<u>PLIB_SPI_BufferWrite</u>	Write the data to the SPI buffer.
≡	<u>PLIB_SPI_ClockPolaritySelect</u>	Enables clock polarity.
≡	<u>PLIB_SPI_CommunicationWidthSelect</u>	Selects the data width for the SPI communication.
≡	<u>PLIB_SPI_Disable</u>	Disables the SPI module.
≡	<u>PLIB_SPI_Enable</u>	Enables the SPI module.
≡	<u>PLIB_SPI_ErrorInterruptDisable</u>	Enables SPI error interrupts.
≡	<u>PLIB_SPI_ErrorInterruptEnable</u>	Enables SPI error interrupts
≡	<u>PLIB_SPI_ExistsAudioCommunicationWidth</u>	Identifies whether the AudioCommunicationWidth feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsAudioErrorControl</u>	Identifies whether the AudioErrorControl feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsAudioProtocolControl</u>	Identifies whether the AudioProtocolControl feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsAudioProtocolMode</u>	Identifies whether the AudioProtocolMode feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsAudioTransmitMode</u>	Identifies whether the AudioTransmitMode feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsBaudRate</u>	Identifies whether the BaudRate feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsBaudRateClock</u>	Identifies whether the BaudRateClock feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsBuffer</u>	Identifies whether the Buffer feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsBusStatus</u>	Identifies whether the BusStatus feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsClockPolarity</u>	Identifies whether the ClockPolarity feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsCommunicationWidth</u>	Identifies whether the CommunicationWidth feature exists on the SPI module.
≡	<u>PLIB_SPI_ExistsEnableControl</u>	Identifies whether the EnableControl feature exists on the SPI module.

	<u>PLIB_SPI_ExistsErrorInterruptControl</u>	Identifies whether the ErrorInterruptControl feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFIFOControl</u>	Identifies whether the FIFOControl feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFIFOCount</u>	Identifies whether the FIFOCount feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFIFOInterruptMode</u>	Identifies whether the FIFOInterruptMode feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus</u>	Identifies whether the FIFOShiftRegisterEmptyStatus feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFramedCommunication</u>	Identifies whether the FramedCommunication feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFrameErrorStatus</u>	Identifies whether the FrameErrorStatus feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFrameSyncPulseCounter</u>	Identifies whether the FrameSyncPulseCounter feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFrameSyncPulseDirection</u>	Identifies whether the FrameSyncPulseDirection feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFrameSyncPulseEdge</u>	Identifies whether the FrameSyncPulseEdge feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFrameSyncPulsePolarity</u>	Identifies whether the FrameSyncPulsePolarity feature exists on the SPI module.
	<u>PLIB_SPI_ExistsFrameSyncPulseWidth</u>	Identifies whether the FrameSyncPulseWidth feature exists on the SPI module.
	<u>PLIB_SPI_ExistsInputSamplePhase</u>	Identifies whether the InputSamplePhase feature exists on the SPI module.
	<u>PLIB_SPI_ExistsMasterControl</u>	Identifies whether the MasterControl feature exists on the SPI module.
	<u>PLIB_SPI_ExistsOutputDataPhase</u>	Identifies whether the OutputDataPhase feature exists on the SPI module.
	<u>PLIB_SPI_ExistsPinControl</u>	Identifies whether the PinControl feature exists on the SPI module.
	<u>PLIB_SPI_ExistsPrimaryPrescale</u>	Identifies whether the PrimaryPrescale feature exists on the SPI module.
	<u>PLIB_SPI_ExistsReadDataSignStatus</u>	Identifies whether the ReadDataSignStatus feature exists on the SPI module.
	<u>PLIB_SPI_ExistsReceiveBufferStatus</u>	Identifies whether the ReceiveBufferStatus feature exists on the SPI module.
	<u>PLIB_SPI_ExistsReceiveFIFOStatus</u>	Identifies whether the ReceiveFIFOStatus feature exists on the SPI module.
	<u>PLIB_SPI_ExistsReceiverOverflow</u>	Identifies whether the ReceiverOverflow feature exists on the SPI module.
	<u>PLIB_SPI_ExistsSecondaryPrescale</u>	Identifies whether the SecondaryPrescale feature exists on the SPI module.
	<u>PLIB_SPI_ExistsSlaveSelectControl</u>	Identifies whether the SlaveSelectControl feature exists on the SPI module.
	<u>PLIB_SPI_ExistsStopInIdleControl</u>	Identifies whether the StopInIdle feature exists on the SPI module.
	<u>PLIB_SPI_ExistsTransmitBufferEmptyStatus</u>	Identifies whether the TransmitBufferEmptyStatus feature exists on the SPI module.
	<u>PLIB_SPI_ExistsTransmitBufferFullStatus</u>	Identifies whether the TransmitBufferFullStatus feature exists on the SPI module.

	PLIB_SPI_ExistsTransmitUnderRunStatus	Identifies whether the TransmitUnderRunStatus feature exists on the SPI module.
	PLIB_SPI_FIFOCountGet	Reads the SPI Buffer Element Count bits for either receive or transmit.
	PLIB_SPI_FIFODisable	Disables the SPI enhanced buffer.
	PLIB_SPI_FIFOEnable	Enables the SPI enhanced buffer.
	PLIB_SPI_FIFOInterruptModeSelect	Selects the SPI buffer interrupt mode.
	PLIB_SPI_FIFOShiftRegisterIsEmpty	Returns the current status of the SPI shift register.
	PLIB_SPI_FramedCommunicationDisable	Disables framed SPI support.
	PLIB_SPI_FramedCommunicationEnable	Enables framed SPI support.
	PLIB_SPI_FrameErrorStatusGet	Returns the current status of the SPI frame error.
	PLIB_SPI_FrameSyncPulseCounterSelect	Selects at which character the SPI frame sync pulse is generated.
	PLIB_SPI_FrameSyncPulseDirectionSelect	Selects the frame sync pulse direction.
	PLIB_SPI_FrameSyncPulseEdgeSelect	Selects the frame sync pulse edge.
	PLIB_SPI_FrameSyncPulsePolaritySelect	Selects the frame sync pulse polarity.
	PLIB_SPI_FrameSyncPulseWidthSelect	Sets the frame sync pulse width.
	PLIB_SPI_InputSamplePhaseSelect	Selects the SPI data input sample phase.
	PLIB_SPI_IsBusy	Returns the current SPI module activity status.
	PLIB_SPI_MasterEnable	Enables the SPI in Master mode.
	PLIB_SPI_OutputDataPhaseSelect	Selects serial output data change.
	PLIB_SPI_PinDisable	Enables the selected SPI pins.
	PLIB_SPI_PinEnable	Enables the selected SPI pins.
	PLIB_SPI_PrescalePrimarySelect	Selects the primary prescale for SPI Master mode.
	PLIB_SPI_PrescaleSecondarySelect	Selects the secondary prescale for SPI Master mode.
	PLIB_SPI_ReadDataIsSignExtended	Returns the current status of the receive (RX) FIFO sign-extended data.
	PLIB_SPI_ReceiverBufferIsFull	Returns the current status of the SPI receive buffer.
	PLIB_SPI_ReceiverFIFOIsEmpty	Returns the current status of the SPI receive FIFO.
	PLIB_SPI_ReceiverHasOverflowed	Returns the current status of the SPI receiver overflow.
	PLIB_SPI_ReceiverOverflowClear	Clears the SPI receive overflow flag.
	PLIB_SPI_SlaveEnable	Enables the SPI in Slave mode.
	PLIB_SPI_SlaveSelectDisable	Disables Master mode slave select.
	PLIB_SPI_SlaveSelectEnable	Enables Master mode slave select.
	PLIB_SPI_StopInIdleDisable	Continues module operation when the device enters Idle mode.
	PLIB_SPI_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.
	PLIB_SPI_TransmitBufferIsEmpty	Returns the current status of the transmit buffer.
	PLIB_SPI_TransmitBufferIsFull	Returns the current transmit buffer status of the SPI module.
	PLIB_SPI_TransmitUnderRunStatusGet	Returns the current status of the transmit underrun.

Types

	Name	Description
	SPI_DATA_TYPE	Data type defining the SPI data size.

File Name

plib_spi.h

Company

Microchip Technology Inc.

5.6.23 SQI Peripheral Library

5.6.23.1 Introduction

SQI Peripheral Library for Microchip Microcontrollers

This [library](#) provides a low-level abstraction of the Serial Quad Interface (SQI) Peripheral that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the SQI module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The SQI module is a synchronous serial interface that provides access to serial Flash memories and other serial devices. The SQI bus interface consists of four data lines (SQID3-SQID0), a clock line (SQICLK), two select lines (SQICS0 and SQICS1) and the module supports single Lane (identical to SPI), dual Lane, and quad Lane interface modes. SQI operates in only master mode. The SQI module has configurable transmit and receive buffers, programmable baud rates through the internal clock divider, clock phase, and clock polarity control for efficient data operations. Transmit and receive buffers can be accessed through SQITXDATA and SQIRXDATA registers. Similarly, the control buffer can be accessed through the SQI1CON register and is mainly used to pipeline the operations.

The SQI module operates in three transfer modes: DMA, PIO, and XIP using three data flow modes: SPI Mode 0 and Mode 3, and a high-speed Serial Flash mode . All three modes use the control buffer to pipeline the command/data sequences on the SQI bus. Following diagrams describe typical SQI hardware interface and software flow. Figure 1 shows the typical hardware interface of SQI module:

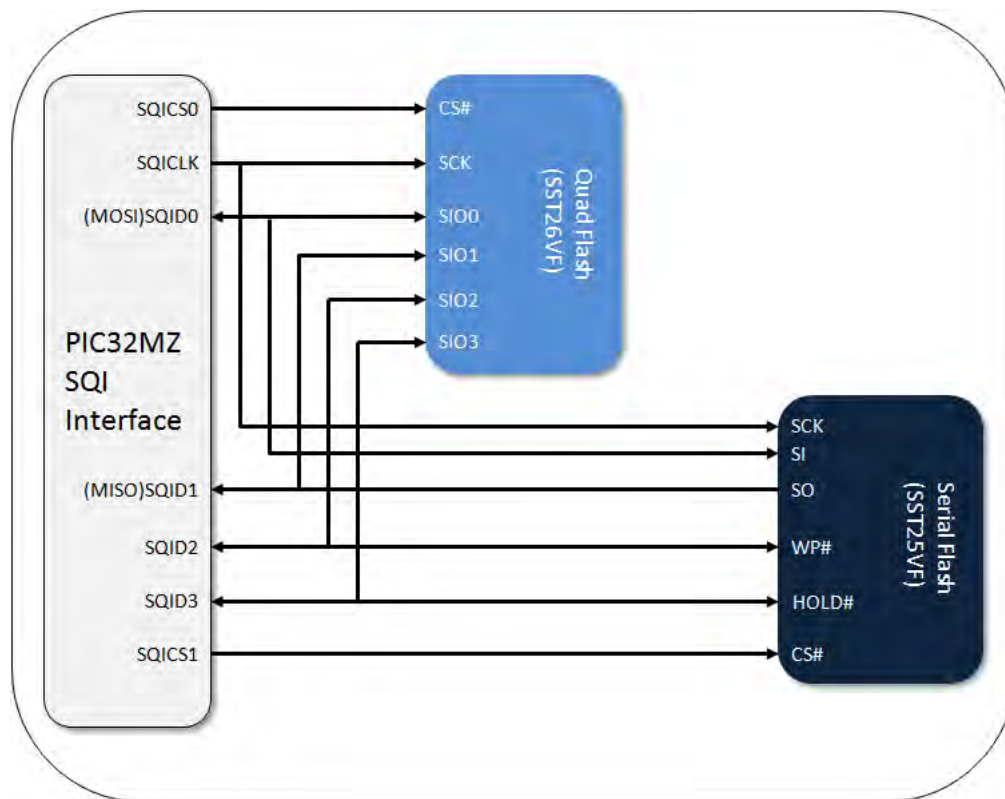


Figure 1: Hardware Interface Diagram

Refer to SQI Family Reference Manual section, which is available on [Microchip website](#) for additional details on control register information and operation.

5.6.23.2 Release Notes

MPLAB Harmony Version: v0.70b **SQI Peripheral Library Version :**0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.23.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip, its subsidiaries and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO

YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.23.4 Using the Library

This topic describes the basic architecture of the SQI Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_sqi.h](#)

The interface to the SQI peripheral library is defined in the "[plib_sqi.h](#)" header file. This file is included by the "peripheral.h" file. Any C language source (.c) file that uses the SQI peripheral library should include "peripheral.h".

Library File:

The SQI peripheral library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the library interacts with the framework.

5.6.23.4.1 Hardware Abstraction Model

This library provides the low-level abstraction of the SQI module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Before jumping into the software abstraction, it is important to understand the SQI hardware a little more in detail.

Figure 2 provides a glimpse of SQI hardware block diagram. As shown the diagram the features of SQI are divided into multiple logic blocks and the SQI peripheral library provides APIs to configure the various logic block thus the features.

The control and status SFRs block contains all the registers that are used to configure the SQI in a specific mode of operation. The PIO mode of operation of SQI requires a set of registers to be configured by the Host processor and writing to/ reading from the TXDATA and RXDATA registers. The XIP logic facilitates the SQI modules eXecute-In-Place mode of operation, where Host processor programs the set of registers that configure the XIP mode and the engine automatically reads the external flash and the read data gets mapped into a direct memory region. Where as DMA mode requires the Host processor to program a set of buffer descriptors and instructs SQI to point to the address of those buffer descriptors for the reads and writes. As mentioned earlier there are 3 buffers; transmit, receive and control that are accessed through SQIXTXDATA, SQIXRXDATA and SQIXCON registers respectively.

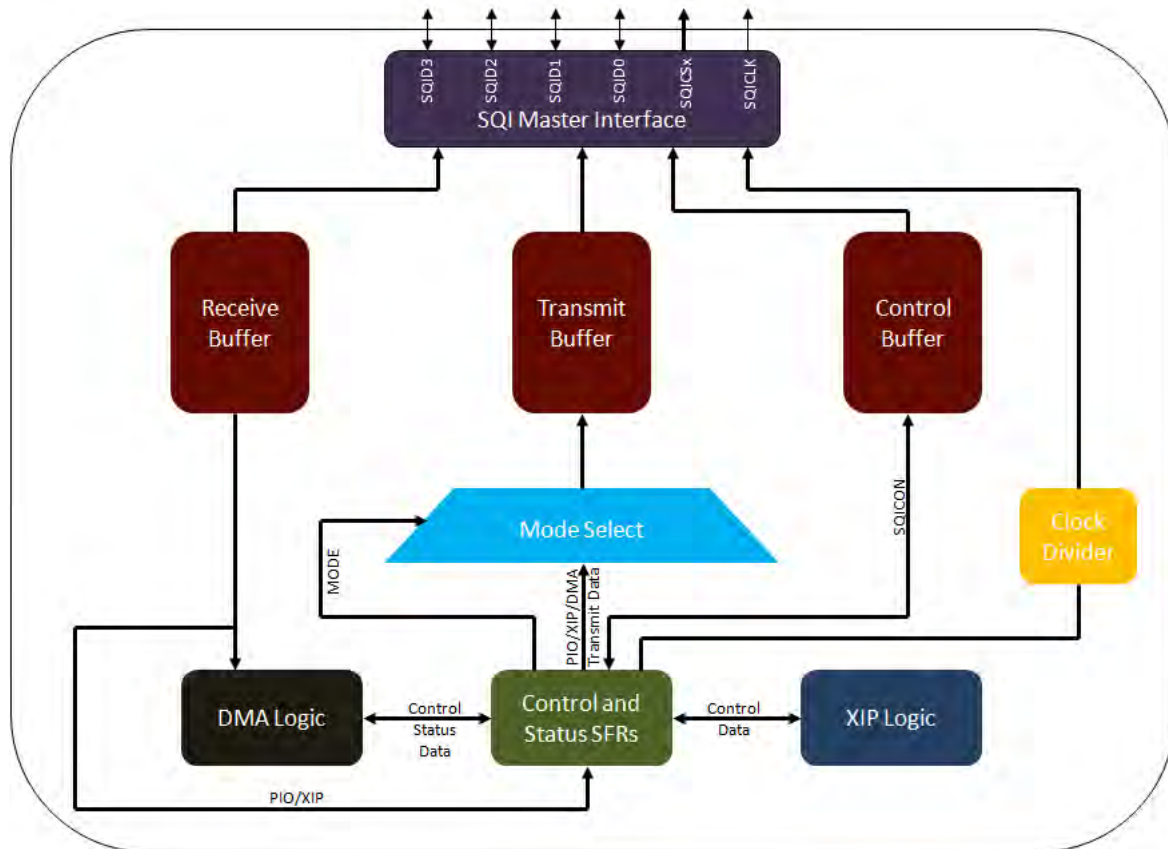


Figure 2: SQI Hardware Block Diagram

The SQI Peripheral Library takes requests from application software or the SQI device driver and controls the SQI hardware as per the inputs passed to the [Library Interface](#). Figure 3 provides the SQI peripheral library software abstraction layer diagram.

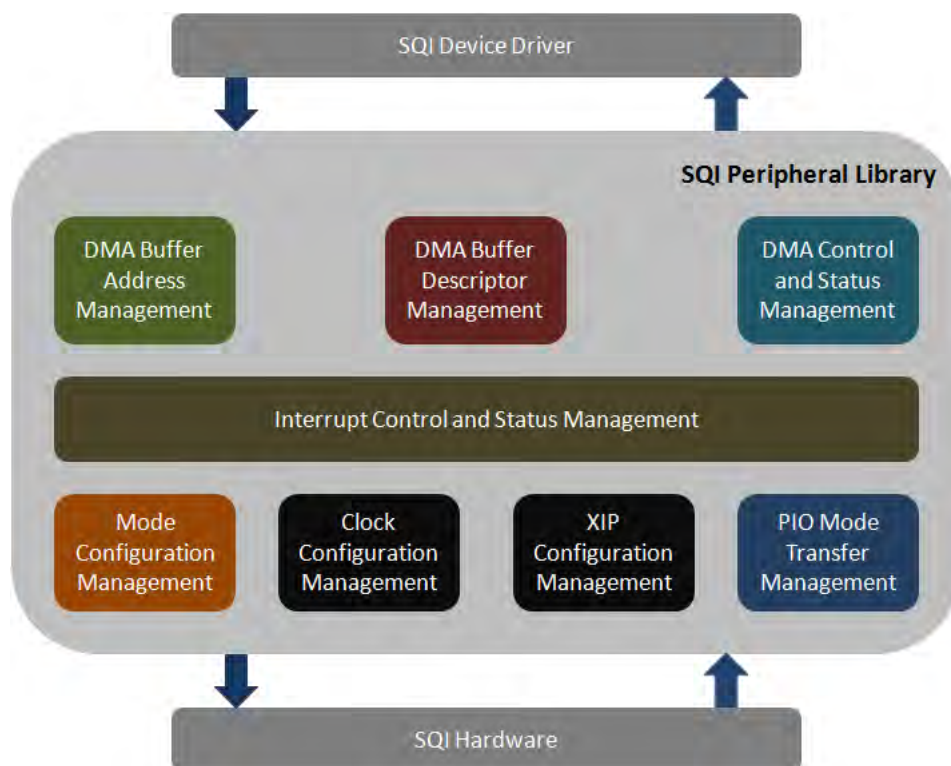


Figure 3: SQI Software Abstraction Model

The major components are SQI Peripheral Library are:

- Configuration Management
- Interrupt Control and Status Management
- DMA Mode Management

5.6.23.4.2 Library Usage Model

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the SQI module.

Library Interface Section	Description
Mode Configuration Management Functions	These functions are used to setup transfer mode (DMA/XIP/PIO), data mode (SPI Mode 0/Mode 3/ Serial Flash), lane mode (single/dual/quad) and XIP control registers.
Clock Configuration Management Functions	These functions are used to enable the clock, setup a clock divider and check the status of the clock to see if it's stable.
XIP Configuration Management Functions	These functions are used to setup parameters for XIP mode of operation.
PIO Mode Transfer Management Functions	These functions are used to setup TXDATA, RXDATA registers and status checks.
Interrupt Control and Status Management Functions	These functions are used to control the interrupts. SQI supports only status type of interrupts (no error interrupts).
DMA Buffer Address Management Functions	These functions are used to setup the address of the base descriptor, check the address of the current descriptor

DMA Buffer Descriptor Management Functions	These functions are used to setup and control 4 words (BD_CTRL, BD_STATUS (reserved), BD_BUF_ADDR and BD_NEXT_PTR)
DMA Control and Status Management Functions	These functions are used to control the buffer descriptor fetch process and check the status during the buffer descriptor fetches.

5.6.23.4.3 How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

5.6.23.4.3.1 Core Functionality

SQI modules core functionality is to transfer data between the serial device (mostly quad flash) attached and the PIC microcontroller. In order to achieve this, SQI module uses 3 transfer modes: DMA, PIO and XIP. Each transfer mode can be configured to use any of the 3 available data flow modes (Mode 0, Mode 3 and Serial Flash Mode) to achieve the transfers. DMA and PIO modes are typically used to transfer data and XIP mode is used for code execution.

Each mode of operation requires setting-up the clock, selecting data flow mode and selecting lane mode and other parameters as required by the device driver or application. SQI peripheral library includes complete set of API necessary to perform these operations.

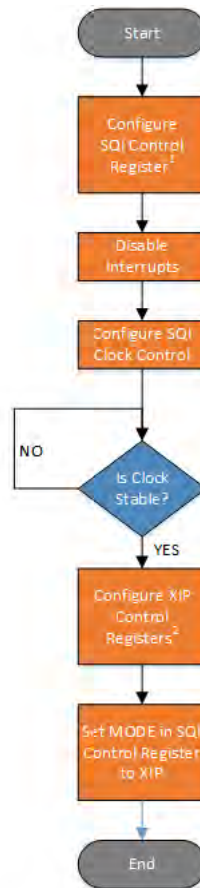
The following sections describe the SQI core functionality in each mode of operation.

5.6.23.4.3.1.1 XIP Mode

XIP (eXecute-In-Place) mode is mostly used to execute the code out of the attached serial flash device. However, this mode can also be used to transfer data as deemed necessary. In order for SQI to operate in XIP mode, host processor would have to setup the two SFRs dedicated to the XIP mode in addition to SQI configuration and clock control SFRs. Please refer to the specific data sheet and family reference manual for the configuration details.

Please note that all the XIP configuration values depend on the parameters of the interfaced serial flash memory device. In XIP mode, reading values of any other SFRs return the contents of the serial flash device, the way it is mapped.

Figure 4 provides the XIP mode software flow.



Note 1: This step configures SQI configuration fields (SQIEN, CSEN and DATAEN) except the MODE<1:0>
 Note 2: There are two XIP control registers

Figure 4: XIP Mode Flow Diagram

Example:

```

// Example is written to access SST26VF032 flash device
// Example assumes the Flash Device is connected to Chip Select 0

//Global defines
#define SST26VF_HS_READ    0x0B
#define SST26VF_MODE_CODE  0x0
#define SST26VF_MODE_BYTES 0x0

// Setup SQI Configuration Bits
PLIB_SQI_Enable(SQI_ID_1);
PLIB_SQI_CSOutputEnableSelect(SQI_ID_1, SQI_CS_OEN_0);
PLIB_SQI_DataOutputEnableSelect(SQI_ID_1, SQI_DATA_OEN_QUAD);

// Disable All Interrupts
PLIB_SQI_InterruptDisable(SQI_ID_1, SQI_ALL_INT);
PLIB_SQI_InterruptSignalDisable(SQI_ID_1, SQI_ALL_INT);

// Setup SQI Transfer Clock
PLIB_SQI_ClockDividerSet(SQI_ID_1, CLK_DIV_1);
PLIB_SQI_ClockEnable(SQI_ID_1);
while (!PLIB_SQI_ClockIsStable(SQI_ID_1));
// Configure XIP Control 1 SFR
PLIB_SQI_XIPControlWord1Set (SQI_ID_1,

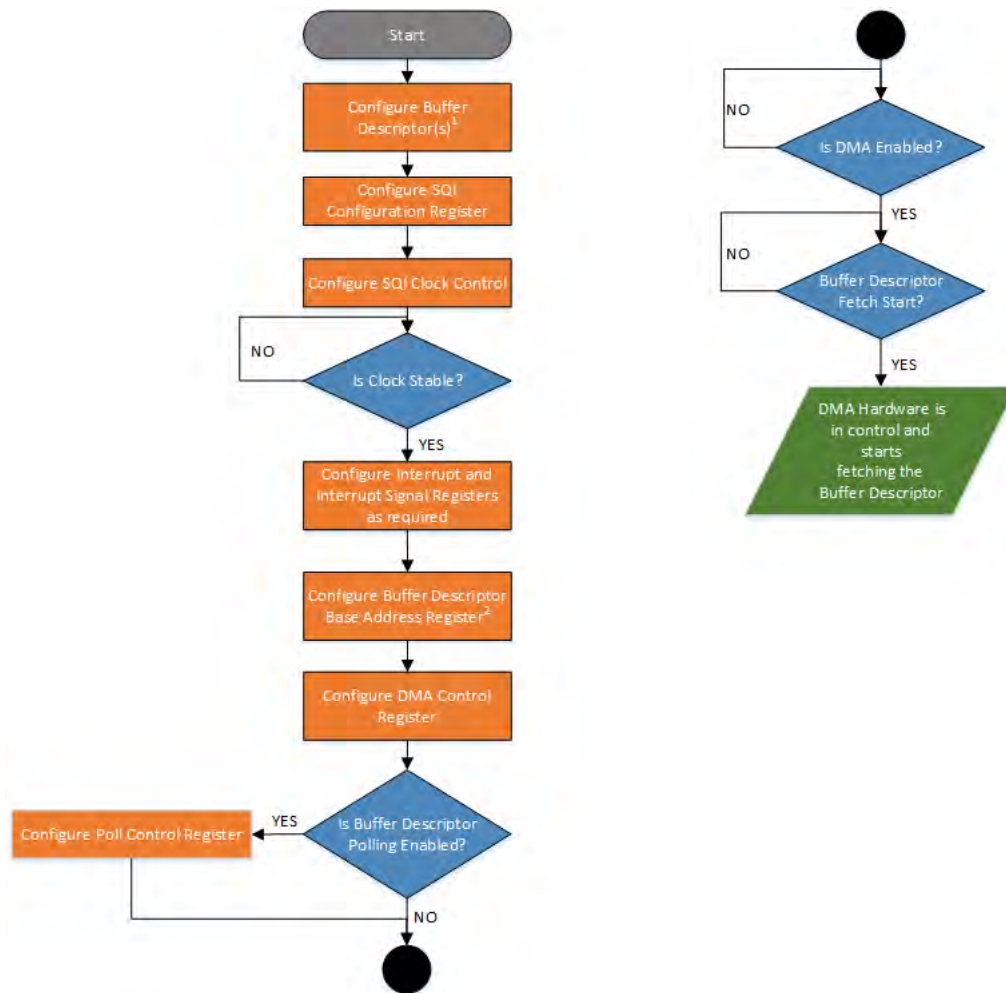
```

```
        SQI_LANE_QUAD,  
        SQI_LANE_QUAD,  
        SQI_LANE_QUAD,  
        SQI_LANE_QUAD,  
        SQI_LANE_QUAD,  
        SST26VF_HS_READ,  
        ADDR_BYTES_3,  
        DUMMY_BYTES_2  
    );  
  
    // Configure XIP Control 2 SFR  
    PLIB_SQI_XIPControlWord2Set (SQI_ID_1,  
        SST26VF_MODE_CODE,  
        MODE_BYTES_0,  
        SQI_CS_0  
    );  
  
    // Set Transfer Mode to XIP  
    PLIB_SQI_TransferModeSet(SQI_ID_1, SQI_XFER_MODE_XIP);  
  
    // The SQI is now in XIP mode and should start fetching the code
```

5.6.23.4.3.1.2 DMA Mode

DMA mode is used for higher throughput data transfers. In DMA mode, the SQI DMA engine controls the data transfers off-loading the host processor. However, the host processor would have to configure the buffer descriptor and some SQI control SFRs to initiate the DMA process. Once the transactions are in progress, the DMA will be in control and the host processor can get the control through the interrupts or by disabling the DMA.

Figure 5 provides the DMA mode software flow.



Note 1: Buffer descriptor is a data structure in memory containing 4 words (BD_CTRL, BD_STAT, BD_BUFADDR and BD_NXTPTR).

Note 2: Base address register points to first buffer descriptor in the chain.

Figure 5: DMA Mode Flow Diagram

Example:

```

// Example is written to access SST26VF032 flash device
// Example assumes the Flash Device is connected to Chip Select 0

//Global defines
#define SST26VF_EQIO      0x38
#define SST26VF_HS_READ   0x0B
#define SST26VF_MODE_CODE 0x0
#define SST26VF_MODE_BYTES 0x0
#define NUMBER_OF_BDs     5
#define POLL_CON_VALUE    10

{

// Setup the Buffer Descriptors
for ( i=0; i <(NUMBER_OF_BDs-1); i++)
{
// Setup Buffer Descriptors. Handle structures as application
  
```

```

// needs.

// Configure SQI Control Fields
PLIB_SQI_Enable(SQI_ID_1);
PLIB_SQI_CSOutputEnableSelect(SQI_ID_1, SQI_CS_OEN_0);
PLIB_SQI_DataOutputEnableSelect(SQI_ID_1, SQI_DATA_OEN_2);
PLIB_SQI_BurstEnable(SQI_ID_1);
PLIB_SQI_SerialModeSet(SQI_ID_1);
PLIB_SQI_TransferModeSet(SQI_ID_1, SQI_XFER_MODE_PIO);

// Configure the SQI Transfer Clock
PLIB_SQI_ClockDividerSet(SQI_ID_1, CLK_DIV_4);
PLIB_SQI_ClockEnable(SQI_ID_1);
while (!PLIB_SQI_ClockIsStable(SQI_ID_1));

//Enable Buffer Descriptor Done and Packet Done Interrupts
PLIB_SQI_InterruptEnable(SQI_ID_1, BDDONE);
PLIB_SQI_InterruptSignalEnable(SQI_ID_1, BDDONE);
PLIB_SQI_InterruptEnable(SQI_ID_1, PKTCOMP);
PLIB_SQI_InterruptSignalEnable(SQI_ID_1, PKTCOMP);

// Send EQIO(0x38) Command to SST26VF032 to Enable Quad Lane Mode
// Note: These steps are not in the flow as these are specific to
// one Flash Device (SST26VF032)
PLIB_SQI_ChipSelectSet(SQI_ID_1, SQI_CS_0);
PLIB_SQI_LaneModeSet(SQI_ID_1, SQI_SINGLE_LANE);
PLIB_SQI_TransferCommandSet(SQI_ID_1, SQI_CMD_TRANSMIT);
// Transfers 1 Command Byte, No Address/Dummy Bytes
PLIB_SQI_TransferCountSet(SQI_ID_1, 0x1);
PLIB_SQI_TransmitterDataSend(SQI_ID_1, SST26VF_EQIO);

// Set the SQI in DMA Mode
PLIB_SQI_TransferModeSet(SQI_ID_1, SQI_XFER_MODE_DMA);

// Setup the Base Buffer Descriptor Address
if (!PLIB_SQI_DMAIsActive(SQI_ID_1))
    PLIB_SQI_DMABDBaseAddressSet(SQI_ID_1, (void *) (&MY_BASE_BD_STRUCT));

//Setup DMA Control Register
PLIB_SQI_DMABDEnable(SQI_ID_1);
PLIB_SQI_DMABDPollEnable(SQI_ID_1);

// Setup DMA Polling
If (PLIB_SQI_DMABDPollIsEnabled(SQI_ID_1))
    // Poll DMA every 10 cycles
    PLIB_SQI_PollControlSet(SQI_ID_1, MY_POLL_CONTROL_VALUE);

// Start the DMA Process - DMA Engine is now in Control
If (PLIB_SQI_DMAIsEnabled(SQI_ID_1))
{
    if (!PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
        PLIB_SQI_DMABDFetchStart(SQI_ID_1);
}
}

```

5.6.23.4.3.1.3 PIO Mode

PIO mode is mostly used for data transfers. In PIO mode, the SQI module is entirely under the control of host processor. For the SQI module to operate in PIO mode, the host processor would have to set up the configuration, control, and clock control SFRs. Once the transactions are in progress, the host processor can use interrupts and status flags to control the operation. Please refer to the specific device data sheet and family reference manual for the configuration details.

Figure 6 provides the PIO mode software flow.

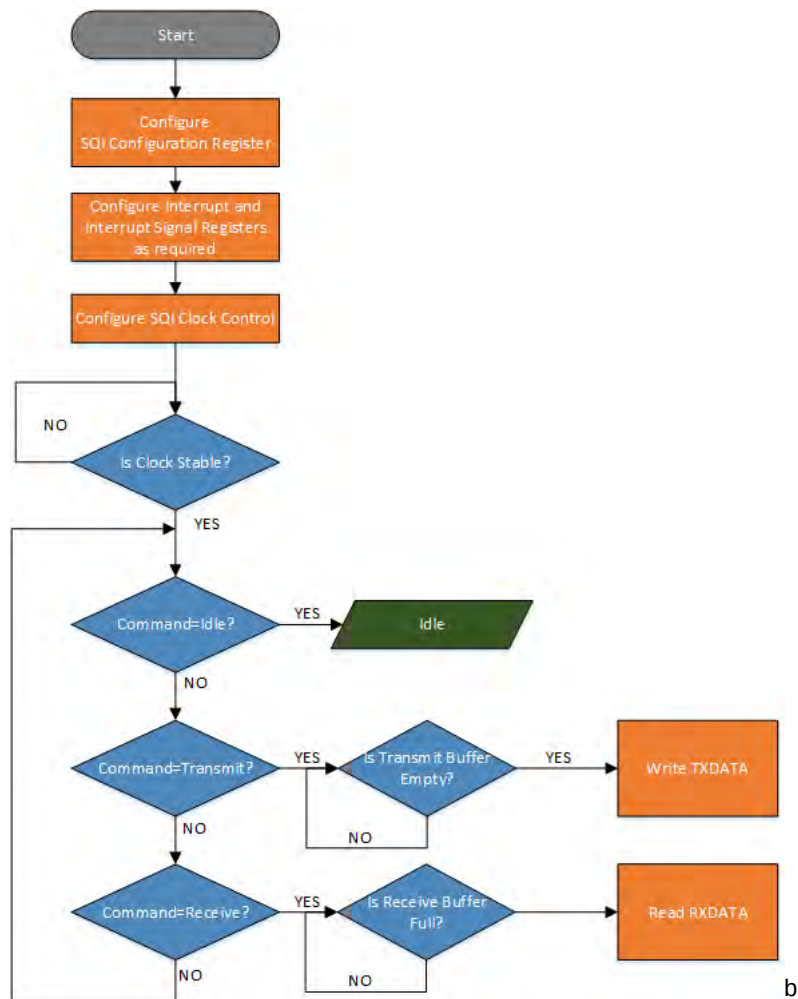


Figure 6: PIO Mode Flow Diagram

Example:

*// Example is written to access SST26VF032 flash device
 // Example assumes the Flash Device is connected to Chip Select 0
 // Example reads the device ID of the SST26VF032*

//Global defines

```

#define FALSE 0
#define TRUE 1
#define SST26VF_EQIO 0x38
#define SST26VF_HS_READ 0x0B
#define SST26VF_QJEDID_READ 0x9F
#define SST26VF_JEDEC_ID 0xBF2602
  
```

```

{
int32_t jedecID;
  
```

// Setup SQI Configuration (SQI1CFG) Register

```

PLIB_SQI_ConfigWordSet(MY_SQI_INSTANCE,
    1,
    SQI_CS_OEN_0,
    SQI_DATA_OEN_QUAD,
    0,
    1,
  
```

```

        0,
        0,
        0,
        SQI_DATA_FORMAT_MSBF,
        SQI_DATA_MODE_3,
        SQI_XFER_MODE_PIO
    );

    PLIB_SQI_ControlBufferThresholdSet(SQI_ID_1, 4);

    // Enable Specific Interrupts
    PLIB_SQI_InterruptEnable(SQI_ID_1, TXEMPTY|RXFULL|TXTHRIE|RXTHRIE);

    // Setup SQI Transfer Clock
    PLIB_SQI_ClockDividerSet(SQI_ID_1, CLK_DIV_1);
    PLIB_SQI_ClockEnable(SQI_ID_1);
    while (!PLIB_SQI_ClockIsStable(SQI_ID_1));

    // Setup SQI Transmit Command Threshold
    PLIB_SQI_TxCommandThresholdSet(SQI_ID_1, 3);

    // Setup SQI Receive Buffer Threshold
    PLIB_SQI_TxBufferThresholdIntSet(SQI_ID_1, 3);

    // Configure SQI Control Fields and Send EQIO(0x38) Command to
    // SST26VF032 to Enable Quad Lane Mode
    PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
        1,
        SQI_CS_1,
        SQI_LANE_SINGLE,
        SQI_CMD_TRANSMIT,
        1
    );
    PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
        0,
        SQI_CS_1,
        SQI_LANE_QUAD,
        SQI_CMD_TRANSMIT,
        1
    );
    PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
        0,
        SQI_CS_1,
        SQI_LANE_QUAD,
        SQI_CMD_RECIEVE,
        3
    );
    if ( PLIB_SQI_InterruptFlagGet(SQI_ID_1, TXEMPTY) )
        PLIB_SQI_TransmitData(SQI_ID_1, SST26VF_EQIO);

    //Configure SQI to read JEDEC-ID in Quad Mode
    if ( PLIB_SQI_InterruptFlagGet(SQI_ID_1, TXEMPTY) )
        PLIB_SQI_TransmitData(SQI_ID_1, SST26VF_QJEDID_READ);

    if (PLIB_SQI_InterruptFlagGet(SQI_ID_1, RXFULL))
        jedecID = PLIB_SQI_ReceiveData(SQI_ID_1);

    if (jedecID == SST26VF_JEDEC_ID)
        return TRUE;
    else
        return FALSE;
}

```

5.6.23.4.4 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.23.5 Library Interface

Data Types and Constants

	Name	Description
	<u>SQI_ADDR_BYTES</u>	Defines the list of SQI Address Bytes
	<u>SQI_BD_CTRL_WORD</u>	Defines the list of SQI Buffer Descriptor Control Word
	<u>SQI_BD_STATE</u>	Defines the list of SQI Buffer Descriptor State
	<u>SQI_CLK_DIV</u>	Defines the list of SQI Clock Divider Values
	<u>SQI_CS_NUM</u>	Defines the list of SQI Chip Selects
	<u>SQI_CS_OEN</u>	Defines the list of SQI Chip Selects on which output is enabled
	<u>SQI_DATA_FORMAT</u>	Defines the Data Format Options available (LSBF/MSBF)
	<u>SQI_DATA_MODE</u>	Defines the list of SQI Data Modes
	<u>SQI_DATA_OEN</u>	Defines the list of SQI Data pins on which output is enabled
	<u>SQI_DATA_TYPE</u>	Data type defining the SQI Data size.
	<u>SQI_DUMMY_BYTES</u>	Defines the list of SQI Dummy Bytes
	<u>SQI_INTERRUPTS</u>	Defines the list of SQI Interrupts
	<u>SQI_LANE_MODE</u>	Defines the list of SQI Lane Modes (single/dual/quad)
	<u>SQI_MODE_BYTES</u>	Defines the list of SQI Mode Bytes
	<u>SQI_MODULE_ID</u>	Identifies the SQI modules supported
	<u>SQI_XFER_CMD</u>	Defines the list of SQI Transfer Commands
	<u>SQI_XFER_MODE</u>	Defines the list of SQI Transfer Modes

Clock Configuration Management Functions

	Name	Description
	<u>PLIB_SQI_ClockDividerGet</u>	Returns the SQI clock divider value.
	<u>PLIB_SQI_ClockDividerSet</u>	Sets the SQI clock (that drives the SQI protocol) divider value. Divides the base clock to generate the SQI clock.
	<u>PLIB_SQI_ClockEnable</u>	Enables the SQI transfer clock.
	<u>PLIB_SQI_ClockIsStable</u>	Returns SQI transfer clock state.

DMA Buffer Address Management Functions

	Name	Description
	<u>PLIB_SQI_DMABDBaseAddressGet</u>	Returns the address of the base buffer descriptor.
	<u>PLIB_SQI_DMABDBaseAddressSet</u>	Sets the address of the base buffer descriptor.
	<u>PLIB_SQI_DMABDCurrentAddressGet</u>	Returns the address of the current buffer descriptor in process.

DMA Buffer Descriptor Management Functions

	Name	Description
	<u>PLIB_SQI_DMABDControlWordGet</u>	Returns Current Buffer Descriptor Control Word Information.
	<u>PLIB_SQI_DMABDReceiveBufferCountGet</u>	Returns the receive buffer space in bytes.
	<u>PLIB_SQI_DMABDReceiveBufferLengthGet</u>	Returns the receive length in bytes.

	PLIB_SQI_DMABDReceiveStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMABDTransmitBufferCountGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_DMABDTransmitBufferLengthGet	Returns the transmit length in bytes.
	PLIB_SQI_DMABDTransmitStateGet	Returns the current state of the buffer descriptor in progress.

DMA Control and Status Management Functions

	Name	Description
	PLIB_SQI_DMABDFetchStart	Start the DMA buffer descriptor fetch process.
	PLIB_SQI_DMABDIsBusy	Returns true if DMA Buffer Descriptor is busy.
	PLIB_SQI_DMABDPollCounterSet	Sets the poll counter value.
	PLIB_SQI_DMABDPollDisable	Disables the buffer descriptor polling.
	PLIB_SQI_DMABDPollEnable	Enables the buffer descriptor polling.
	PLIB_SQI_DMABDPollsEnabled	Returns true if DAM buffer descriptor poll is eanbled and false if it's disabled.
	PLIB_SQI_DMABDStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMADisable	Disables the built-in DMA logic.
	PLIB_SQI_DMAEnable	Enables the built-in DMA logic.
	PLIB_SQI_DMAHasStarted	Returns true if DMA process has started if not returns false.
	PLIB_SQI_DMAIsEnabled	Returns true if DAM is eanbled and false if it's disabled.

Feature Existence Functions

	Name	Description
	PLIB_SQI_ExistsBDBaseAddress	Identifies whether the BDBaseAddress feature exists on the SQI module
	PLIB_SQI_ExistsBDControlWord	Identifies whether the BDControlWord feature exists on the SQI module
	PLIB_SQI_ExistsBDCurrentAddress	Identifies whether the BDCurrentAddress feature exists on the SQI module
	PLIB_SQI_ExistsBDPollCount	Identifies whether the BDPollCount feature exists on the SQI module
	PLIB_SQI_ExistsBDPollingEnable	Identifies whether the BDPollingEnable feature exists on the SQI module
	PLIB_SQI_ExistsBDProcessState	Identifies whether the BDProcessState feature exists on the SQI module
	PLIB_SQI_ExistsBDRxBufCount	Identifies whether the BDRxBufCount feature exists on the SQI module
	PLIB_SQI_ExistsBDRxLength	Identifies whether the BDRxLength feature exists on the SQI module
	PLIB_SQI_ExistsBDRxState	Identifies whether the BDRxState feature exists on the SQI module
	PLIB_SQI_ExistsBDTxBufCount	Identifies whether the BDTxBufCount feature exists on the SQI module
	PLIB_SQI_ExistsBDTxLength	Identifies whether the BDTxLength feature exists on the SQI module
	PLIB_SQI_ExistsBDTxState	Identifies whether the BDTxState feature exists on the SQI module
	PLIB_SQI_ExistsBurstControl	Identifies whether the BurstControl feature exists on the SQI module
	PLIB_SQI_ExistsChipSelect	Identifies whether the ChipSelect feature exists on the SQI module
	PLIB_SQI_ExistsClockControl	Identifies whether the ClockControl feature exists on the SQI module
	PLIB_SQI_ExistsClockDivider	Identifies whether the ClockDivider feature exists on the SQI module
	PLIB_SQI_ExistsClockReady	Identifies whether the ClockReady feature exists on the SQI module
	PLIB_SQI_ExistsConBufThreshold	Identifies whether the ConBufThreshold feature exists on the SQI module
	PLIB_SQI_ExistsConfigWord	Identifies whether the ConfigWord feature exists on the SQI module

	PLIB_SQI_ExistsControlWord	Identifies whether the ControlWord feature exists on the SQI module
	PLIB_SQI_ExistsCSDeassert	Identifies whether the CSDeassert feature exists on the SQI module
	PLIB_SQI_ExistsCSOutputEnable	Identifies whether the CSOutputEnable feature exists on the SQI module
	PLIB_SQI_ExistsDataFormat	Identifies whether the DataFormat feature exists on the SQI module
	PLIB_SQI_ExistsDataModeControl	Identifies whether the DataModeControl feature exists on the SQI module
	PLIB_SQI_ExistsDataOutputEnable	Identifies whether the DataOutputEnable feature exists on the SQI module
	PLIB_SQI_ExistsDataPinStatus	Identifies whether the DataPinStatus feature exists on the SQI module
	PLIB_SQI_ExistsDmaEnable	Identifies whether the DMAEnable feature exists on the SQI module
	PLIB_SQI_ExistsDMAEngineBusy	Identifies whether the DMAEngineBusy feature exists on the SQI module
	PLIB_SQI_ExistsDMAProcessInProgress	Identifies whether the DMAProcessInProgress feature exists on the SQI module
	PLIB_SQI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SQI module
	PLIB_SQI_ExistsHoldPinControl	Identifies whether the HoldPinControl feature exists on the SQI module
	PLIB_SQI_ExistsInterruptControl	Identifies whether the InterruptControl feature exists on the SQI module
	PLIB_SQI_ExistsInterruptSignalControl	Identifies whether the InterruptSignalControl feature exists on the SQI module
	PLIB_SQI_ExistsInterruptStatus	Identifies whether the InterruptStatus feature exists on the SQI module
	PLIB_SQI_ExistsLaneMode	Identifies whether the LaneMode feature exists on the SQI module
	PLIB_SQI_ExistsReceiveLatch	Identifies whether the ReceiveLatch feature exists on the SQI module
	PLIB_SQI_ExistsRxBufferCount	Identifies whether the RxBufferCount feature exists on the SQI module
	PLIB_SQI_ExistsRxBufIntThreshold	Identifies whether the RxBufIntThreshold feature exists on the SQI module
	PLIB_SQI_ExistsRxBufThreshold	Identifies whether the RxBufThreshold feature exists on the SQI module
	PLIB_SQI_ExistsRxData	Identifies whether the RxData feature exists on the SQI module
	PLIB_SQI_ExistsRxUnderRun	Identifies whether the RxUnderRun feature exists on the SQI module
	PLIB_SQI_ExistsSoftReset	Identifies whether the SoftReset feature exists on the SQI module
	PLIB_SQI_ExistsStartDMA	Identifies whether the StartDMA feature exists on the SQI module
	PLIB_SQI_ExistsTransferCommand	Identifies whether the TransferCommand feature exists on the SQI module
	PLIB_SQI_ExistsTransferCount	Identifies whether the TransferCount feature exists on the SQI module
	PLIB_SQI_ExistsTransferModeControl	Identifies whether the TransferModeControl feature exists on the SQI module
	PLIB_SQI_ExistsTxBufferFree	Identifies whether the TxBufferFree feature exists on the SQI module
	PLIB_SQI_ExistsTxBufIntThreshold	Identifies whether the TxBufIntThreshold feature exists on the SQI module
	PLIB_SQI_ExistsTxBufThreshold	Identifies whether the TxBufThreshold feature exists on the SQI module

	PLIB_SQI_ExistsTxData	Identifies whether the TxData feature exists on the SQI module
	PLIB_SQI_ExistsTxOverFlow	Identifies whether the TxOverFlow feature exists on the SQI module
	PLIB_SQI_ExistsWPPinControl	Identifies whether the WPPinControl feature exists on the SQI module
	PLIB_SQI_ExistsXIPChipSelect	Identifies whether the XIPChipSelect feature exists on the SQI module
	PLIB_SQI_ExistsXIPControlWord1	Identifies whether the XIPControlWord1 feature exists on the SQI module
	PLIB_SQI_ExistsXIPControlWord2	Identifies whether the XIPControlWord2 feature exists on the SQI module
	PLIB_SQI_ExistsXIPLaneMode	Identifies whether the XIPLaneMode feature exists on the SQI module
	PLIB_SQI_ExistsXIPModeBytes	Identifies whether the XIPModeBytes feature exists on the SQI module
	PLIB_SQI_ExistsXIPModeCode	Identifies whether the XIPModeCode feature exists on the SQI module
	PLIB_SQI_ExistsXIPNumberOfAddressBytes	Identifies whether the XIPNumberOfAddressBytes feature exists on the SQI module
	PLIB_SQI_ExistsXIPNumberOfDummyBytes	Identifies whether the XIPNumberOfDummyBytes feature exists on the SQI module
	PLIB_SQI_ExistsXIPReadOpCode	Identifies whether the XIPReadOpCode feature exists on the SQI module

Interrupt Control and Status Management Functions

	Name	Description
	PLIB_SQI_DataLineStatus	Returns the logical status of the SQI data lines.
	PLIB_SQI_InterruptDisable	Disables the interrupt source.
	PLIB_SQI_InterruptEnable	Enables the interrupt source passed.
	PLIB_SQI_InterruptFlagGet	Return SQI Interrupt flag status.
	PLIB_SQI_InterruptIsEnabled	Returns the interrupt state (eanbled/disabled).
	PLIB_SQI_InterruptSignalDisable	Disables the interrupt signal source.
	PLIB_SQI_InterruptSignalEnable	Enables the interrupt signal source passed.
	PLIB_SQI_InterruptSignalIsEnabled	Returns the interrupt signal state (eanbled/disabled).

Mode Configuration Management Functions

	Name	Description
	PLIB_SQI_BurstEnable	This bit sets the burst enable (BURSTEN) function for higher throughput. This function is artifact of the system bus architecture.
	PLIB_SQI_ClockDisable	Disables the SQI transfer clock.
	PLIB_SQI_ControlBufferThresholdGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_ControlBufferThresholdSet	Sets the control buffer threshold value.
	PLIB_SQI_ControlWordGet	Get the SQI control word.
	PLIB_SQI_ControlWordSet	Sets SQI Control Word.
	PLIB_SQI_CSOutputEnableSelect	Selects the output enables on SQI chip select pins.
	PLIB_SQI_DataFormatGet	Returns the data format to Least Significant Bit First or LITTLE-ENDIAN.
	PLIB_SQI_DataFormatSet	Sets the data format to Least Significant Bit First or LITTLE-ENDIAN.
	PLIB_SQI_DataModeGet	Returns the SQI data mode of operation (SPI Mode0/Mode3).
	PLIB_SQI_DataModeSet	Sets the SQI data mode of operation (SPI Mode0/Mode3/Serial Flash).

	PLIB_SQI_DataOutputEnableSelect	Selects the output enables on SQI data outputs.
	PLIB_SQI_Disable	Disables the SQI module.
	PLIB_SQI_Enable	Enables the SQI module.
	PLIB_SQI_HoldClear	This bit clears the hold function to be disabled on SQID3 in single and dual lane modes.
	PLIB_SQI_HoldGet	This bit gets the status of HOLD function on SQID3 pin.
	PLIB_SQI_HoldSet	This bit sets the hold function to be enabled on SQID3 in single or dual lane modes.
	PLIB_SQI_LaneModeGet	Returns the lane mode (single/dual/quad).
	PLIB_SQI_LaneModeSet	Sets the data lane mode (single/dual/quad).
	PLIB_SQI_NumberOfReceiveBufferReads	Returns the number of receive buffer reads.
	PLIB_SQI_ReceiveData	Reads the data from the receive buffer.
	PLIB_SQI_ReceiveLatchDisable	Disables the receive latch so receive data is discarded when in transmit mode.
	PLIB_SQI_ReceiveLatchEnable	Enable the receive latch so receive data is latched during transmit mode.
	PLIB_SQI_ReceiveLatchGet	Returns the receive latch status in transmit mode.
	PLIB_SQI_SoftReset	This bit issues a soft reset to the SQI module.
	PLIB_SQI_TransferModeGet	Returns the SQI transfer mode (PIO/DMA/XIP) of operation.
	PLIB_SQI_TransferModeSet	Sets the SQI transfer mode (PIO/DMA/XIP) of operation.
	PLIB_SQI_TransmitData	Writes data into the SQI transmit buffer.
	PLIB_SQI_WriteProtectClear	This bit clears the Write-Protect function to be disabled on SQID2 in single or dual lane modes.
	PLIB_SQI_WriteProtectGet	This bit gets the Write-Protect state of Write_Protect Feature on SQID2.
	PLIB_SQI_WriteProtectSet	This bit sets the Write-Protect function to be enabled on SQID2 in single or dual Lane modes only.

PIO Mode Transfer Management Functions

	Name	Description
	PLIB_SQI_ByteCountGet	Returns the current transmit/receive count.
	PLIB_SQI_ByteCountSet	Sets the transmit/receive count.
	PLIB_SQI_ChipSelectDeassertDisable	Clears the chip select de-assert.
	PLIB_SQI_ChipSelectDeassertEnable	Sets the chip select de-assert.
	PLIB_SQI_ChipSelectGet	Returns the chip select that is currently active.
	PLIB_SQI_ChipSelectSet	Activates the chip select.
	PLIB_SQI_ConfigWordGet	Get the SQI configuration word.
	PLIB_SQI_ConfigWordSet	Sets SQI Configuration Word.
	PLIB_SQI_ReceiveBufferIsUnderrun	Returns the status of receive buffer.
	PLIB_SQI_RxBufferThresholdGet	Returns the receive command threshold.
	PLIB_SQI_RxBufferThresholdIntGet	Sets the receive buffer threshold interrupt.
	PLIB_SQI_RxBufferThresholdIntSet	Sets the receive buffer threshold for interrupt.
	PLIB_SQI_RxBufferThresholdSet	Sets the receive command threshold.
	PLIB_SQI_TransferDirectionGet	Returns the transfer command.
	PLIB_SQI_TransferDirectionSet	Sets the transfer command.
	PLIB_SQI_TransmitBufferFreeSpaceGet	Returns the number of transmit buffer words available.
	PLIB_SQI_TransmitBufferHasOverflowed	Returns the current status of the transmit buffer.
	PLIB_SQI_TxBufferThresholdGet	Returns the transmit command threshold value.

	PLIB_SQI_TxBufferThresholdIntGet	Returns the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdIntSet	Sets the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdSet	Sets the transmit command threshold.

XIP Configuration Management Functions

	Name	Description
	PLIB_SQI_XIPAddressBytesGet	Returns the number of address bytes.
	PLIB_SQI_XIPAddressBytesSet	Sets the number of address bytes.
	PLIB_SQI_XIPChipSelectGet	Returns the current active chip select.
	PLIB_SQI_XIPChipSelectSet	Activates the chip select in XIP mode.
	PLIB_SQI_XIPControlWord1Get	Get the XIP mode control word 1.
	PLIB_SQI_XIPControlWord1Set	Sets the XIP mode control word 1.
	PLIB_SQI_XIPControlWord2Get	Get the XIP mode control word 2.
	PLIB_SQI_XIPControlWord2Set	Sets the XIP mode control word 2.
	PLIB_SQI_XIPDummyBytesGet	Sets the number of dummy bytes.
	PLIB_SQI_XIPDummyBytesSet	Sets the number of dummy bytes.
	PLIB_SQI_XIPLaneModeGet	Returns the lane (single/dual/quad) mode in XIP mode.
	PLIB_SQI_XIPLaneModeSet	Selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.
	PLIB_SQI_XIPModeBytesGet	Returns the number of bytes used for mode code command.
	PLIB_SQI_XIPModeBytesSet	Allocates the bytes for mode code command.
	PLIB_SQI_XIPModeCodeGet	Returns the mode code op-code.
	PLIB_SQI_XIPModeCodeSet	Sets the mode code command.
	PLIB_SQI_XIPReadOpcodeGet	Returns the read opcode in XIP mode.
	PLIB_SQI_XIPReadOpcodeSet	Sets the read opcode in XIP mode.

Description

This section describes the Application Programming Interface (API) functions of the SQI Peripheral Library. Refer to each section for a detailed description.

5.6.23.5.1 Mode Configuration Management Functions

5.6.23.5.1.1 PLIB_SQI_BurstEnable Function

C

```
void PLIB_SQI_BurstEnable(
    SQI_MODULE_ID index
);
```

Description

This routine enables burst mode for higher throughput. Burst mode should always be enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_BurstEnable(MY_SQI_INSTANCE);
```

5.6.23.5.1.2 PLIB_SQI_ClockDisable Function**C**

```
void PLIB_SQI_ClockDisable(  
    SQI_MODULE_ID index  
);
```

Description

This routine disables the SQI transfer clock (divided clock).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

SQICLK is disabled.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_ClockDisable(MY_SQI_INSTANCE);
```

5.6.23.5.1.3 PLIB_SQI_ControlBufferThresholdGet Function**C**

```
uint8_t PLIB_SQI_ControlBufferThresholdGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the threshold value for the control buffer in bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Control buffer threshold space.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t conBufThres = PLIB_SQI_ControlBufferThresholdGet(MY_SQI_INSTANCE);
```

5.6.23.5.1.4 PLIB_SQI_ControlBufferThresholdSet Function**C**

```
void PLIB_SQI_ControlBufferThresholdSet(  
    SQI_MODULE_ID index,  
    uint8_t threshold  
);
```

Description

This routine sets the control buffer threshold value in bytes, that is used to signal control buffer threshold interrupts.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Control buffer threshold

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer and MY_CONBUF_THRESHOLD is the threshold value.  
PLIB_SQI_ControlBufferThresholdSet(MY_SQI_INSTANCE, MY_CONBUF_THRESHOLD);
```

5.6.23.5.1.5 PLIB_SQI_ControlWordGet Function**C**

```
uint32_t PLIB_SQI_ControlWordGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the SQI control word.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint32_t sqiCon = PLIB_SQI_ControlWordGet (MY_SQI_INSTANCE);
```

5.6.23.5.1.6 PLIB_SQI_ControlWordSet Function**C**

```
void PLIB_SQI_ControlWordSet(  
    SQI_MODULE_ID index,  
    bool csDeassert,  
    SQI_CS_NUM csNum,  
    SQI_LANE_MODE laneMode,  
    SQI_XFER_CMD command,  
    uint16_t xferCount  
);
```

Description

This routine sets SQI Control Word. This API is cumulation of multiple APIs, in case driver plans to write the complete control word. In PIO Mode Control word is before each transfer

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csDeassert	Chip select deassert after transfer
csNum	Active chip select number (0/1)
laneMode	SQI lane mode (QUAL/DUAL/SINGLE)
command	Transfer command (TRANSMIT/RECIEVE/IDLE)
xferCount	Number of bytes to be transmitted/received

Returns

None.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,  
    1,  
    SQI_CS_1,
```

```
SQI_LANE_QUAD,  
SQI_CMD_TRANSMIT,  
5  
);
```

5.6.23.5.1.7 PLIB_SQI_CSOutputEnableSelect Function

C

```
void PLIB_SQI_CSOutputEnableSelect(  
    SQI_MODULE_ID index,  
    SQI_CS_OEN csPins  
);
```

Description

This routine enables the selected SQI chip selects as outputs.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csPins	Chip select pins for which outputs are enabled

Returns

None.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_CSOutputEnableSelect(MY_SQI_INSTANCE, SQI_CS_OEN_0);
```

5.6.23.5.1.8 PLIB_SQI_DataFormatGet Function

C

```
SQI_DATA_FORMAT PLIB_SQI_DataFormatGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the SQI data format (LSBF/MSBF).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

True if Data Format is LSBF, else false.

Remarks

Typical data format in most of the Systems is LITTLE ENDIAN.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_DATA_FORMAT dataFormat = PLIB_SQI_DataFormatGet(MY_SQI_INSTANCE);
```

5.6.23.5.1.9 PLIB_SQI_DataFormatSet Function**C**

```
void PLIB_SQI_DataFormatSet(  
    SQI_MODULE_ID index,  
    SQI_DATA_FORMAT dataformat  
);
```

Description

This routine sets the SQI data format to LSBF (LITTLE-ENDIAN).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Typical data format in most of the Systems is LITTLE ENDIAN.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_DataFormatSet(MY_SQI_INSTANCE, SQI_DATA_FORMAT_LSBF);
```

5.6.23.5.1.10 PLIB_SQI_DataModeGet Function**C**

```
SQI_DATA_MODE PLIB_SQI_DataModeGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the SQI data mode (SPI Mode 0/Mode 3) of operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Data mode (SPI Mode 0/Mode 3).

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_DATA_MODE dataMode = PLIB_SQI_DataModeGet(MY_SQI_INSTANCE);
```

5.6.23.5.1.11 PLIB_SQI_DataModeSet Function**C**

```
void PLIB_SQI_DataModeSet(  
    SQI_MODULE_ID index,  
    SQI_DATA_MODE mode  
);
```

Description

This routine sets the data mode to be SPI Mode 0, SPI Mode 3 or Serial Flash.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Data mode (Mode 0/ Mode 3)

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer and SQI_DATA_MODE_3 is an enum element.  
PLIB_SQI_DataModeSet(MY_SQI_INSTANCE, SQI_DATA_MODE_SF);
```

5.6.23.5.1.12 PLIB_SQI_DataOutputEnableSelect Function**C**

```
void PLIB_SQI_DataOutputEnableSelect(  
    SQI_MODULE_ID index,  
    SQI_DATA_OEN dataPins  
);
```

Description

This routine enables the selected SQI data lines as outputs.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dataPins	Data pins for which outputs are enabled

Returns

None.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_DataOutputEnableSelect(MY_SQI_INSTANCE, SQI_DATA_OEN_DUAL);
```

5.6.23.5.1.13 PLIB_SQI_Disable Function**C**

```
void PLIB_SQI_Disable(  
    SQI_MODULE_ID index  
);
```

Description

This routine disables the SQI module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_Disable(MY_SQI_INSTANCE);
```

5.6.23.5.1.14 PLIB_SQI_Enable Function**C**

```
void PLIB_SQI_Enable(  
    SQI_MODULE_ID index  
);
```

Description

This routine enables the SQI module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

The SQICLK, SQICSx, SQID0, SQID1, SQID2 and SQID3 pins must be assigned to available RPN pins before use.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_Enable(MY_SQI_INSTANCE);
```

5.6.23.5.1.15 PLIB_SQI_HoldClear Function**C**

```
void PLIB_SQI_HoldClear(  
    SQI_MODULE_ID index  
);
```

Description

This routine sets SQID3 to be controlled by SQI for normal data operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_HoldClear(MY_SQI_INSTANCE);
```

5.6.23.5.1.16 PLIB_SQI_HoldGet Function**C**

```
void PLIB_SQI_HoldGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine gets the status of HOLD function on SQID3 pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
bool holdStatus = PLIB_SQI_HoldGet(MY_SQI_INSTANCE);
```

5.6.23.5.1.17 PLIB_SQI_HoldSet Function**C**

```
void PLIB_SQI_HoldSet(  
    SQI_MODULE_ID index  
);
```

Description

This routine sets the SQID3 pin to HIGH/LOW to be used for hold function in single and dual lane modes for supported flash memories.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function should be used only when SQI is in single/dual lane modes.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_HoldSet(MY_SQI_INSTANCE);
```

5.6.23.5.1.18 PLIB_SQI_LaneModeGet Function**C**

```
SQI_LANE_MODE PLIB_SQI_LaneModeGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the number of lanes (single/dual/quad) used for transfers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Lane Mode.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_LANE_MODE laneMode = PLIB_SQI_LaneModeGet(MY_SQI_INSTANCE);
```

5.6.23.5.1.19 PLIB_SQI_LaneModeSet Function**C**

```
void PLIB_SQI_LaneModeSet(  
    SQI_MODULE_ID index,  
    SQI_LANE_MODE mode  
);
```

Description

This routine sets the number of lanes (single/dual/quad) used for transfers.

Preconditions

Make sure the output enable is selected on the data lines ([PLIB_SQI_DataOutputEnableSelect](#)). The device needs to be programmed to the same mode that the SQI controller is set to (might require special commands).

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Lane mode (single/dual/quad)

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_LaneModeSet(MY_SQI_INSTANCE, SQI_LANE_QUAD);
```

5.6.23.5.1.20 PLIB_SQI_NumberOfReceiveBufferReads Function

C

```
uint8_t PLIB_SQI_NumberOfReceiveBufferReads(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the number of receive buffer reads for debug purpose.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Number of Receive Buffer Reads.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t rxBufReads = PLIB_SQI_ReceiveBufferReadsGet(MY_SQI_INSTANCE);
```

5.6.23.5.1.21 PLIB_SQI_ReceiveData Function

C

```
uint32_t PLIB_SQI_ReceiveData(  
    SQI_MODULE_ID index  
);
```

Description

This routine reads the data from the SQI receive buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint32_t receivedData= PLIB_SQI_ReceiveData(MY_SQI_INSTANCE);
```

5.6.23.5.1.22 PLIB_SQI_ReceiveLatchDisable Function

C

```
void PLIB_SQI_ReceiveLatchDisable(  
    SQI_MODULE_ID index  
);
```

Description

This routine disables the receive latch, which disables the receive data to be latched in transmit mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_ReceiveLatchDisable(MY_SQI_INSTANCE);
```

5.6.23.5.1.23 PLIB_SQI_ReceiveLatchEnable Function

C

```
void PLIB_SQI_ReceiveLatchEnable(  
    SQI_MODULE_ID index  
);
```

Description

This routine enables the receive latch, which latches receive data in transmit mode. Otherwise receive data in transmit mode is discarded.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

As most of the SQI communication is half-duplex, enable this function only when it's absolutely required.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.
```



```
PLIB_SQI_ReceiveLatchEnable(MY_SQI_INSTANCE);
```

5.6.23.5.1.24 PLIB_SQI_ReceiveLatchGet Function

C

```
bool PLIB_SQI_ReceiveLatchGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the receive latch status in transmit mode. Returns true, if latch is set (enabling latching of receive buffer data), false if latch is not set (disabling the latching of the receive buffer data).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

True if Receive latch is set, false if not.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
bool rxLatch = PLIB_SQI_ReceiveLatchGet(MY_SQI_INSTANCE);
```

5.6.23.5.1.25 PLIB_SQI_SoftReset Function

C

```
void PLIB_SQI_SoftReset(  
    SQI_MODULE_ID index  
);
```

Description

This routine issues a software reset to the SQI module clearing all the read/write register, internal state machines and data buffers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_SoftReset(MY_SQI_INSTANCE);
```

5.6.23.5.1.26 PLIB_SQI_TransferModeGet Function**C**

```
SQI_XFER_MODE PLIB_SQI_TransferModeGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the SQI transfer mode (PIO/DMA/XIP) of operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured.

Returns

Transfer mode (PIO/DMA/XIP).

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_XFER_MODE xferMode = PLIB_SQI_TransferModeGet(MY_SQI_INSTANCE);
```

5.6.23.5.1.27 PLIB_SQI_TransferModeSet Function**C**

```
void PLIB_SQI_TransferModeSet(  
    SQI_MODULE_ID index,  
    SQI_XFER_MODE mode  
);
```

Description

This routine sets the SQI transfer mode (PIO/DMA/XIP) of operation.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Transfer mode (PIO/DMA/XIP)

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer and SQI_XFER_MODE_DMA is an enum element.  
PLIB_SQI_TransferModeSet(MY_SQI_INSTANCE, SQI_XFER_MODE_DMA);
```

5.6.23.5.1.28 PLIB_SQI_TransmitData Function**C**

```
void PLIB_SQI_TransmitData(  
    SQI_MODULE_ID index,  
    uint32_t data  
);
```

Description

This routine writes the data into the SQI transmit buffer, which will be eventually sent out on SQI bus.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be transmitted

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer and MY_TRANSMIT_DATA is the data to be sent.  
PLIB_SQI_TransmitData(MY_SQI_INSTANCE, MY_TRANSMIT_DATA);
```

5.6.23.5.1.29 PLIB_SQI_WriteProtectClear Function**C**

```
void PLIB_SQI_WriteProtectClear(  
    SQI_MODULE_ID index  
);
```

Description

This routine disables the SQID2 pin to be used for write-protect function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_WriteProtectClear(MY_SQI_INSTANCE);
```

5.6.23.5.1.30 PLIB_SQI_WriteProtectGet Function**C**

```
bool PLIB_SQI_WriteProtectGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine gets the Write-Protect feature status on SQID2 pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
bool wpStatus = PLIB_SQI_WriteProtectGet(MY_SQI_INSTANCE);
```

5.6.23.5.1.31 PLIB_SQI_WriteProtectSet Function**C**

```
void PLIB_SQI_WriteProtectSet(  
    SQI_MODULE_ID index  
);
```

Description

This routine enables the SQID2 pin to be used for write-protect in single and dual lane modes for supported flash memories.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function should be used only when SQI is in single/dual lane modes.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_WriteProtectSet (MY_SQI_INSTANCE);
```

5.6.23.5.2 Clock Configuration Management Functions

5.6.23.5.2.1 PLIB_SQI_ClockDividerGet Function

C

```
SQI_CLK_DIV PLIB_SQI_ClockDividerGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the SQI clock divider value. The returned value in conjunction with the SQI base clock can be used to get the clock rate of SQI clock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Clock Divider Value.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_CLK_DIV clkDiv = PLIB_SQI_ClockDividerGet(MY_SQI_INSTANCE);
```

5.6.23.5.2.2 PLIB_SQI_ClockDividerSet Function

C

```
void PLIB_SQI_ClockDividerSet(  
    SQI_MODULE_ID index,  
    SQI_CLK_DIV clkDivider  
);
```

Description

This routine sets the SQI clock divider value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clkDivider	Clock divider value

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_ClockDividerSet(MY_SQI_INSTANCE, CLK_DIV_1);
```

5.6.23.5.2.3 PLIB_SQI_ClockEnable Function**C**

```
void PLIB_SQI_ClockEnable(  
    SQI_MODULE_ID index  
);
```

Description

This routine enables the SQI transfer clock (divided clock).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

SQICLK is enabled.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_ClockEnable(MY_SQI_INSTANCE);
```

5.6.23.5.2.4 PLIB_SQI_ClockIsStable Function**C**

```
bool PLIB_SQI_ClockIsStable(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the SQI transfer clock state.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

True if clock is stable and false if it's not.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
bool clockState = PLIB_SQI_ClockIsStable(MY_SQI_INSTANCE);
```

5.6.23.5.3 XIP Configuration Management Functions

5.6.23.5.3.1 PLIB_SQI_XIPAddressBytesGet Function

C

```
SQI_ADDR_BYTES PLIB_SQI_XIPAddressBytesGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the number of address bytes to be sent to the flash.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Number of Address Bytes.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t addressBytes = PLIB_SQI_XIPAddressBytesGet(MY_SQI_INSTANCE);
```

5.6.23.5.3.2 PLIB_SQI_XIPAddressBytesSet Function

C

```
void PLIB_SQI_XIPAddressBytesSet(  
    SQI_MODULE_ID index,  
    SQI_ADDR_BYTES bytes
```

```
);
```

Description

This routine sets the number of address bytes to be sent to the flash. Typical flash address bytes are 3 (24-bit address).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bytes	Number of address bytes

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_XIPAddressBytesSet(MY_SQI_INSTANCE, ADDR_BYTES_3);
```

5.6.23.5.3.3 PLIB_SQI_XIPChipSelectGet Function**C**

```
SQI_CS_NUM PLIB_SQI_XIPChipSelectGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the active chip select in XIP mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_CS_NUM xipCSActive = (PLIB_SQI_XIPChipSelectGet(MY_SQI_INSTANCE);
```

5.6.23.5.3.4 PLIB_SQI_XIPChipSelectSet Function**C**

```
void PLIB_SQI_XIPChipSelectSet(  

```



```
    SQI_MODULE_ID index,  
    SQI_CS_NUM csNum  
);
```

Description

This routine sets the chip select that is active in XIP mode.

Preconditions

Make sure the chip select output enable is selected on the CS lines ([PLIB_SQI_CSOutputEnableSelect](#)).

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csNum	Chip select number

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_XIPChipSelectSet(MY_SQI_INSTANCE, SQI_CS_0);
```

5.6.23.5.3.5 PLIB_SQI_XIPControlWord1Get Function**C**

```
uint32_t PLIB_SQI_XIPControlWord1Get(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the XIP mode control word 1.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint32_t xipCon1 = PLIB_SQI_XIPControlWord1Set (MY_SQI_INSTANCE);
```

5.6.23.5.3.6 PLIB_SQI_XIPControlWord1Set Function

C

```
void PLIB_SQI_XIPControlWord1Set(  
    SQI_MODULE_ID index,  
    SQI_DUMMY_BYTES dummyBytes,  
    SQI_ADDR_BYTES addressBytes,  
    uint8_t readOpcode,  
    SQI_LANE_MODE dataLaneMode,  
    SQI_LANE_MODE dummyLaneMode,  
    SQI_LANE_MODE modeCodeLaneMode,  
    SQI_LANE_MODE addressLaneMode,  
    SQI_LANE_MODE cmdLaneMode  
);
```

Description

This routine sets XIP mode control word 1. This routine combines work of multiple PLIB APIs and can be used by the driver where complete XIP control word 1 is being modified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dummyBytes	Number of dummy bytes (0-7)
addressBytes	Number of address bytes (0-4)
readOpcode	Quad flash read opcode (ex: 0x0B)
dataLaneMode	Number of SQI data lanes used for sending data bytes
dummyLaneMode	Number of SQI data lanes used for sending dummy bytes
modeCodeLaneMode	Number of SQI data lanes used for sending mode code
addressLaneMode	Number of SQI data lanes used for sending address
cmdLaneMode	Number of SQI data lanes used for sending command

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_XIPControlWord1Set (MY_SQI_INSTANCE,  
                             DUMMY_BYTES_2,  
                             ADDR_BYTES_3,  
                             0x0B,  
                             SQI_LANE_QUAD,  
                             SQI_LANE_QUAD,  
                             SQI_LANE_QUAD,  
                             SQI_LANE_QUAD,  
                             SQI_LANE_QUAD,  
                             );
```

5.6.23.5.3.7 PLIB_SQI_XIPControlWord2Get Function

C

```
uint32_t PLIB_SQI_XIPControlWord2Get(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the XIP mode control word 2.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint32_t xipCon2 = PLIB_SQI_XIPControlWord2Set (MY_SQI_INSTANCE);
```

5.6.23.5.3.8 PLIB_SQI_XIPControlWord2Set Function

C

```
void PLIB_SQI_XIPControlWord2Set(  
    SQI_MODULE_ID index,  
    SQI_CS_NUM devSelect,  
    SQI_MODE_BYTES modeBytes,  
    uint8_t modeCode  
);
```

Description

This routine sets XIP mode control word 2. This routine combines work of multiple PLIB APIs and can be used by the driver where complete XIP control word 2 is being modified.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
modeCode	Mode code used for supported flash devices
modeBytes	Number of mode code bytes
devSelect	Chip select for XIP mode

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_XIPControlWord2Set (MY_SQI_INSTANCE,  
                             SQI_CS_0,  
                             MODE_BYTES_0,  
                             0x0  
                             );
```

5.6.23.5.3.9 PLIB_SQI_XIPDummyBytesGet Function**C**

```
SQI_DUMMY_BYTES PLIB_SQI_XIPDummyBytesGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the number of dummy bytes to be sent to the flash after the address bytes, i.e., before doing a fast read.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t dummyBytes = PLIB_SQI_XIPDummyBytesGet(MY_SQI_INSTANCE);
```

5.6.23.5.3.10 PLIB_SQI_XIPDummyBytesSet Function**C**

```
void PLIB_SQI_XIPDummyBytesSet(  
    SQI_MODULE_ID index,  
    SQI_DUMMY_BYTES bytes  
);
```

Description

This routine sets the number of dummy bytes to be sent to the flash after the address bytes, i.e., before doing a fast read.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bytes	Number of dummy bytes

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_XIPDummyBytesSet(MY_SQI_INSTANCE, DUMMY_BYTE_1);
```

5.6.23.5.3.11 PLIB_SQI_XIPLaneModeGet Function**C**

```
SQI_LANE_MODE PLIB_SQI_XIPLaneModeGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the the number of SQI lanes ((single/dual/quad)) used in XIP mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Lane Mode (Single/Dual/Quad) that SQI Command uses.

Remarks

This routine can't be called when in XIP mode.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_LANE_MODE laneMode = PLIB_SQI_XIPCommandTypeGet(MY_SQI_INSTANCE);
```

5.6.23.5.3.12 PLIB_SQI_XIPLaneModeSet Function**C**

```
void PLIB_SQI_XIPLaneModeSet(  
    SQI_MODULE_ID index,  
    SQI_LANE_MODE dataLanes,  
    SQI_LANE_MODE dummyLanes,  
    SQI_LANE_MODE modeLanes,  
    SQI_LANE_MODE addrLanes,  
    SQI_LANE_MODE cmdLanes  
);
```

Description

This routine selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dataLanes	Data lane mode (single/dual/quad)
dummyLanes	Dummy lane mode (single/dual/quad)
modeLanes	Mode lane mode (single/dual/quad)
addrLanes	Address lane mode (single/dual/quad)
cmdLanes	Command lane mode (single/dual/quad)

Returns

None.

Remarks

This routine can't be called when in XIP mode.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_XIPLaneModeSet(MY_SQI_INSTANCE,  
                          SQI_QUAD_SINGLE,  
                          SQI_QUAD_SINGLE,  
                          SQI_QUAD_SINGLE,  
                          SQI_QUAD_SINGLE,  
                          SQI_QUAD_SINGLE  
                          );
```

5.6.23.5.3.13 PLIB_SQI_XIPModeBytesGet Function**C**

```
SQI_MODE_BYTES PLIB_SQI_XIPModeBytesGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the number of bytes for the mode code command in XIP mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Number of Bytes used for Mode Code Command.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_MODE_BYTES modeCodeBytes = PLIB_SQI_XIPModeBytesGet(MY_SQI_INSTANCE);
```

5.6.23.5.3.14 PLIB_SQI_XIPModeBytesSet Function**C**

```
void PLIB_SQI_XIPModeBytesSet(  
    SQI_MODULE_ID index,  
    SQI_MODE_BYTES bytes  
);
```

Description

This routine sets the number of bytes for the mode code command in XIP mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bytes	Number of bytes of Mode code

Returns

None.

Remarks

Refer to serial device datasheet for the details on this command.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_XIPModeBytesSet(MY_SQI_INSTANCE, MODE_BYTES_0);
```

5.6.23.5.3.15 PLIB_SQI_XIPModeCodeGet Function**C**

```
uint8_t PLIB_SQI_XIPModeCodeGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the mode code command (opcode) in XIP mode for the devices that support it.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Mode Code Opcode.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t modeCode = PLIB_SQI_XIPModeCodeGet(MY_SQI_INSTANCE);
```

5.6.23.5.3.16 PLIB_SQI_XIPModeCodeSet Function**C**

```
void PLIB_SQI_XIPModeCodeSet(  
    SQI_MODULE_ID index,  
    uint8_t code  
);
```

Description

This routine sets the mode code command in XIP mode for the supported flash devices.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
code	Mode code (byte)

Returns

None.

Remarks

Some of the devices seems to support this command, refer to specific serial device data sheet for op-code and sequence details.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_XIPModeCodeSet(MY_SQI_INSTANCE, MY_MODE_CODE);
```

5.6.23.5.3.17 PLIB_SQI_XIPReadOpcodeGet Function**C**

```
uint8_t PLIB_SQI_XIPReadOpcodeGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the read opcode used in XIP mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Read Opcode.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t readOpcode = PLIB_SQI_XIPReadOpcodeGet(MY_SQI_INSTANCE);
```

5.6.23.5.3.18 PLIB_SQI_XIPReadOpcodeSet Function**C**

```
void PLIB_SQI_XIPReadOpcodeSet(  
    SQI_MODULE_ID index,  
    uint8_t opCode  
);
```

Description

This routine sets the flash read opcode in XIP mode. Value of read opcode depends on the flash device attached.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
opCode	Flash read opcode

Returns

None.

Remarks

Refer to the flash device datasheet for supported read opcodes.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer and MY_READ_OPCODE is the opcode dependant on  
// attached serial device.  
PLIB_SQI_XIPReadOpcodeSet(MY_SQI_INSTANCE, MY_READ_OPCODE);
```

5.6.23.5.4 PIO Mode Transfer Management Functions**5.6.23.5.4.1 PLIB_SQI_ByteCountGet Function****C**

```
uint16_t PLIB_SQI_ByteCountGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the transmit/receive count, which is set by software and is actively controlled and maintained by hardware.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
count	Transmit/Receive count

Returns

Transfer Count.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint16_t xferCount = PLIB_SQI_ByteCountGet(MY_SQI_INSTANCE);
```

5.6.23.5.4.2 PLIB_SQI_ByteCountSet Function**C**

```
void PLIB_SQI_ByteCountSet(  
    SQI_MODULE_ID index,  
    uint16_t xferCount  
);
```

Description

This routine sets the number of bytes to transmit or receive, which is set by software and is actively controlled and maintained by hardware.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
count	Transmit/Receive count

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer and MY_XFER_COUNT is the transfer count.  
PLIB_SQI_ByteCountSet(MY_SQI_INSTANCE, MY_XFER_COUNT);
```

5.6.23.5.4.3 PLIB_SQI_ChipSelectDeassertDisable Function**C**

```
void PLIB_SQI_ChipSelectDeassertDisable(  
    SQI_MODULE_ID index
```

```
);
```

Description

This routine disables the chip select de-assert. Chip select stays asserted after transmission or reception of specified number of bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_ChipSelectDeassertDisable(MY_SQI_INSTANCE);
```

5.6.23.5.4.4 PLIB_SQI_ChipSelectDeassertEnable Function**C**

```
void PLIB_SQI_ChipSelectDeassertEnable(  
    SQI_MODULE_ID index  
);
```

Description

This routine enables chip select de-assert. Chip select is de-asserted after transmission or reception of the specified number of bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_ChipSelectDeassertEnable(MY_SQI_INSTANCE);
```

5.6.23.5.4.5 PLIB_SQI_ChipSelectGet Function

C

```
SQI_CS_NUM PLIB_SQI_ChipSelectGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the chip select that is currently active.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Chip Select (2-bit) - Current chip select active (0/1).

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_CS_NUM csNum = PLIB_SQI_ChipSelectGet(MY_SQI_INSTANCE);
```

5.6.23.5.4.6 PLIB_SQI_ChipSelectSet Function

C

```
void PLIB_SQI_ChipSelectSet(  
    SQI_MODULE_ID index,  
    SQI_CS_NUM csNum  
);
```

Description

This routine sets the chip select to be activated on the next transaction.

Preconditions

Make sure the chip select output enable is selected on the CS lines ([PLIB_SQI_CSOutputEnableSelect](#)).

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csNum	Chip select number

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
```

```
// application developer and SQI_CS_NUM_0 is the enum element.  
PLIB_SQI_ChipSelectSet(MY_SQI_INSTANCE, SQI_CS_NUM_0);
```

5.6.23.5.4.7 PLIB_SQI_ConfigWordGet Function

C

```
uint32_t PLIB_SQI_ConfigWordGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the SQI configuration word.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint32_t sqiCfg = PLIB_SQI_ConfigWordGet (MY_SQI_INSTANCE);
```

5.6.23.5.4.8 PLIB_SQI_ConfigWordSet Function

C

```
void PLIB_SQI_ConfigWordSet(  
    SQI_MODULE_ID index,  
    bool sqiEnable,  
    SQI_CS_OEN csPins,  
    SQI_DATA_OEN dataPins,  
    bool reset,  
    bool burstEn,  
    bool hold,  
    bool writeProtect,  
    bool rxLatch,  
    SQI_DATA_FORMAT lsbF,  
    SQI_DATA_MODE dataMode,  
    SQI_XFER_MODE xferMode  
);
```

Description

This routine sets SQI Configuration Word. This API is cumulation of multiple APIs, in case driver plans to write the complete configuration word.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
sqiEnable	Enables/Disables the SQI module
csPins	Chip Select Output Enable
dataPins	Data Output Enable
reset	Resets control, transmit, receive buffers and state machines
burstEn	Burst Enable (always set to '1')
hold	SQID2 to act as HOLD# signal in single and dual lane modes
writeProtect	SQID3 to act as WP# signal in single and dual lane modes
rxLatch	Activates receive latch in transmit mode
lsbf	Sets data endian mode to least significant bit first (LSBF)
dataMode	Sets data mode to mode 0/mode 1/serial flash mode
xferMode	Sets transfer mode to XIP/DMA/PIO mode

Returns

None.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ConfigWordSet(MY_SQI_INSTANCE,
    1,
    SQI_CS_OEN_0,
    SQI_DATA_OEN_QUAD,
    0,
    1,
    0,
    0,
    0,
    0,
    SQI_DATA_FORMAT_LSBF,
    SQI_DATA_MODE_3,
    SQI_XFER_MODE_PIO
);
```

5.6.23.5.4.9 PLIB_SQI_ReceiveBufferIsUnderrun Function**C**

```
bool PLIB_SQI_ReceiveBufferIsUnderrun(
    SQI_MODULE_ID index
);
```

Description

This routine returns the status of the receive buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

True if Receive Buffer is Underrun, False if not.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
bool rxun = PLIB_SQI_ReceiveBufferIsUnderrun(MY_SQI_INSTANCE);
```

5.6.23.5.4.10 PLIB_SQI_RxBufferThresholdGet Function**C**

```
uint8_t PLIB_SQI_RxBufferThresholdGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the receive command threshold that is used to monitor receives based on the receive buffer space availability.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Receive Buffer Threshold value.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t rxBufThres = PLIB_SQI_RxCommandThresholdGet(MY_SQI_INSTANCE);
```

5.6.23.5.4.11 PLIB_SQI_RxBufferThresholdIntGet Function**C**

```
uint8_t PLIB_SQI_RxBufferThresholdIntGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the receive buffer threshold used to set an interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Receive interrupt (buffer) threshold

Returns

Receive Buffer Threshold value (used to trigger an interrupt).

Remarks

This is a 5-bit field and bits 7,6,5 are ignored in the char.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t rxBufIntThres = PLIB_SQI_RxBufferThresholdIntGet(MY_SQI_INSTANCE);
```

5.6.23.5.4.12 PLIB_SQI_RxBufferThresholdIntSet Function**C**

```
void PLIB_SQI_RxBufferThresholdIntSet(
    SQI_MODULE_ID index,
    uint8_t threshold
);
```

Description

This routine sets the receive buffer threshold used to trigger an interrupt. Sets an interrupt condition when receive buffer count is larger than or equal to the receive interrupt threshold bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Receive buffer threshold for interrupt

Returns

None.

Remarks

This is a 5-bit field and bits 7,6,5 are ignored in the char.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_RX_INT_THRESHOLD is the threshold value.
PLIB_SQI_RxBufferThresholdIntSet(MY_SQI_INSTANCE, MY_RX_INT_THRESHOLD);
```

5.6.23.5.4.13 PLIB_SQI_RxBufferThresholdSet Function**C**

```
void PLIB_SQI_RxBufferThresholdSet(
    SQI_MODULE_ID index,
    uint8_t threshold
);
```


Description

This routine sets the receive command threshold that is used to monitor receives based on the receive buffer space availability.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Receive command (buffer) threshold

Returns

None.

Remarks

Valid threshold values are 0-31.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer and MY_RECEIVE_THRESHOLD is the receive threshold value.  
PLIB_SQI_RxCommandThresholdSet(MY_SQI_INSTANCE, MY_RECEIVE_THRESHOLD);
```

5.6.23.5.4.14 PLIB_SQI_TransferDirectionGet Function**C**

```
SQI_XFER_CMD PLIB_SQI_TransferDirectionGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the transfer command that is active currently.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Transfer Command (Idle/Receive/Transmit).

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
SQI_XFER_CMD xferDirection = PLIB_SQI_TransferDirectionGet(MY_SQI_INSTANCE);
```

5.6.23.5.4.15 PLIB_SQI_TransferDirectionSet Function**C**

```
void PLIB_SQI_TransferDirectionSet(  
    SQI_MODULE_ID index,
```

```
    SQI_XFER_CMD  command
);
```

Description

This routine sets the transfer command to Idle/Transmit/Receive.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
command	Transfer command

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer and SQI_CMD_TRANSMIT is an enum element.  
PLIB_SQI_TransferDirectionSet(MY_SQI_INSTANCE, SQI_CMD_TRANSMIT);
```

5.6.23.5.4.16 PLIB_SQI_TransmitBufferFreeSpaceGet Function

C

```
uint8_t PLIB_SQI_TransmitBufferFreeSpaceGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the number of transmit buffer bytes available.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Amount of transmit buffer space free in bytes.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t txBufFreeSpace = PLIB_SQI_TransmitBufferFreeSpaceGet(MY_SQI_INSTANCE);
```

5.6.23.5.4.17 PLIB_SQI_TransmitBufferHasOverflowed Function

C

```
bool PLIB_SQI_TransmitBufferHasOverflowed(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the current state of the transmit buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

True if Transmit Buffer has Overflowed, false if not.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
bool tx0v = PLIB_SQI_TransmitBufferHasOverflowed(MY_SQI_INSTANCE);
```

5.6.23.5.4.18 PLIB_SQI_TxBufferThresholdGet Function

C

```
uint8_t PLIB_SQI_TxBufferThresholdGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the transmit command threshold value that is used to monitor transmits based on the transmit buffer space availability.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Transmit buffer threshold value.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.
```

```
uint8_t txThreshold = PLIB_SQI_TxBufferThresholdGet(MY_SQI_INSTANCE);
```

5.6.23.5.4.19 PLIB_SQI_TxBufferThresholdIntGet Function

C

```
uint8_t PLIB_SQI_TxBufferThresholdIntGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the transmit buffer threshold used to set an interrupt. When enabled, interrupt is triggered when transmit buffer has more space than the transmit interrupt threshold bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Transmit buffer threshold for interrupt.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t txBufIntThres = PLIB_SQI_TxBufferThresholdIntGet(MY_SQI_INSTANCE);
```

5.6.23.5.4.20 PLIB_SQI_TxBufferThresholdIntSet Function

C

```
void PLIB_SQI_TxBufferThresholdIntSet(  
    SQI_MODULE_ID index,  
    uint8_t threshold  
);
```

Description

This routine sets the transmit buffer threshold used for an interrupt. When enabled, interrupt is triggered when transmit buffer has more space than the transmit interrupt threshold bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Transmit interrupt (buffer) threshold

Returns

None.

Remarks

This is a 5-bit field and bits 7,6,5 are ignored in the char.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_TX_INT_THRESHOLD is the threshold value.
PLIB_SQI_TxBufferThresholdIntSet(MY_SQI_INSTANCE, MY_TX_INT_THRESHOLD);
```

5.6.23.5.4.21 PLIB_SQI_TxBufferThresholdSet Function

C

```
void PLIB_SQI_TxBufferThresholdSet(
    SQI_MODULE_ID index,
    uint8_t threshold
);
```

Description

This routine sets the transmit command threshold, which is used to control transmits based on the transmit buffer space availability.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Transmit command (buffer) threshold

Returns

None.

Remarks

Valid threshold values are 0-31.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_TRANSMIT_THRESHOLD is the threshold value.
PLIB_SQI_TxBufferThresholdSet(MY_SQI_INSTANCE, MY_TRANSMIT_THRESHOLD);
```

5.6.23.5.5 Interrupt Control and Status Management Functions

5.6.23.5.5.1 PLIB_SQI_DataLineStatus Function

C

```
bool PLIB_SQI_DataLineStatus(
    SQI_MODULE_ID index,
    bool dataPin
);
```

Description

This routine returns the logical status of the data lines(0/1).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dataPin	Data pin for which status will be returned (0/1/2/3)

Returns

SQIDx Status (High/Low).

Remarks

Parsing values other than 0/1/2/3 returns SQID0 pin status.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool sqiDataLineStatus = PLIB_SQI_DataLineStatus(MY_SQI_INSTANCE, 3);
```

5.6.23.5.5.2 PLIB_SQI_InterruptDisable Function**C**

```
void PLIB_SQI_InterruptDisable(
    SQI_MODULE_ID index,
    SQI_INTERRUPTS interruptFlag
);
```

Description

This routine disables the interrupt source passed into the function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be disabled

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and TXFULL is an enum element.
PLIB_SQI_InterruptDisable(MY_SQI_INSTANCE, SQI_TXFULL);
```

5.6.23.5.5.3 PLIB_SQI_InterruptEnable Function**C**

```
void PLIB_SQI_InterruptEnable(
    SQI_MODULE_ID index,
    SQI_INTERRUPTS interruptFlag
);
```

Description

This routine enables the interrupt source passed into the function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be enabled

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_InterruptEnable(MY_SQI_INSTANCE, SQI_TXFULL);
```

5.6.23.5.5.4 PLIB_SQI_InterruptFlagGet Function**C**

```
bool PLIB_SQI_InterruptFlagGet(
    SQI_MODULE_ID index,
    SQI_INTERRUPTS interruptFlag
);
```

Description

This routine returns the SQI interrupt source flag status (set/cleared).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt flag of interest

Returns

Interrupt status.

Remarks

None.

Example

```
if ( PLIB_SQI_InterruptFlagGet(MY_SQI_INSTANCE, SQI_INT_ANY) )
    if ( PLIB_SQI_InterruptFlagGet(MY_SQI_INSTANCE, SQI_TXFULL) )
    {
        ...
    }
    if ( PLIB_SQI_InterruptFlagGet(MY_SQI_INSTANCE, SQI_RXFULL) )
    {
```

```
        ...
    }
    .
    .
    .
}
```

5.6.23.5.5.5 PLIB_SQI_InterruptIsEnabled Function

C

```
bool PLIB_SQI_InterruptIsEnabled(
    SQI_MODULE_ID index,
    SQI_INTERRUPTS interruptFlag
);
```

Description

This routine returns the interrupt state.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt under check

Returns

True if Interrupt is Enabled, else false.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
if (PLIB_SQI_InterruptIsEnabled(MY_SQI_INSTANCE, SQI_TXFULL))
{
    ..
}
```

5.6.23.5.5.6 PLIB_SQI_InterruptSignalDisable Function

C

```
void PLIB_SQI_InterruptSignalDisable(
    SQI_MODULE_ID index,
    SQI_INTERRUPTS interruptFlag
);
```

Description

This routine disables the interrupt signals source passed into the function, thus prohibiting it from reaching to the external interrupt controller.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be disabled

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_InterruptSignalDisable(MY_SQI_INSTANCE, SQI_TXFULL);
```

5.6.23.5.5.7 PLIB_SQI_InterruptSignalEnable Function**C**

```
void PLIB_SQI_InterruptSignalEnable(  
    SQI_MODULE_ID index,  
    SQI_INTERRUPTS interruptFlag  
);
```

Description

This routine enables the interrupt signal source passed into the function, thus letting it go out to the external interrupt controller.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be enabled

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_InterruptSignalEnable(MY_SQI_INSTANCE, SQI_TXFULL);
```

5.6.23.5.5.8 PLIB_SQI_InterruptSignalIsEnabled Function**C**

```
bool PLIB_SQI_InterruptSignalIsEnabled(  
    SQI_MODULE_ID index,  
    SQI_INTERRUPTS interruptFlag  
);
```

Description

This routine returns the interrupt signal state.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt signal under check

Returns

Boolean flag - Interrupt state true - Interrupt signal is enabled false - Interrupt signal is disabled

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
if (PLIB_SQI_InterruptSignalIsEnabled(MY_SQI_INSTANCE, SQI_TXFULL))  
{  
    ..  
}
```

5.6.23.5.6 DMA Buffer Address Management Functions

5.6.23.5.6.1 PLIB_SQI_DMABDBaseAddressGet Function

C

```
void* PLIB_SQI_DMABDBaseAddressGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the address of the base DMA buffer descriptor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Base Buffer Descriptor Address.

Remarks

Check to make sure if DMA Buffer Descriptor fetch is in progress.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
void *baseBDAddress;
```

```
If (!PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
{
    baseBDAddress = PLIB_SQI_DMABDBaseAddressGet(MY_SQI_INSTANCE);
}
```

5.6.23.5.6.2 PLIB_SQI_DMABDBaseAddressSet Function

C

```
void PLIB_SQI_DMABDBaseAddressSet(
    SQI_MODULE_ID index,
    void * baseBDAddress
);
```

Description

This routine writes the address of the base (first/only) buffer descriptor into the buffer descriptor base address register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baseBDAddress	Base buffer descriptor address

Returns

None.

Remarks

Check to make sure if DMA Buffer Descriptor fetch is in progress.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
if (!PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
    PLIB_SQI_DMABDBaseAddressSet(MY_SQI_INSTANCE, (void *) (&MY_BD_STRUCT));
```

5.6.23.5.6.3 PLIB_SQI_DMABDCurrentAddressGet Function

C

```
void* PLIB_SQI_DMABDCurrentAddressGet(
    SQI_MODULE_ID index
);
```

Description

This routine returns the address of the DMA buffer descriptor that is currently in progress.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Current Buffer Descriptor Address.

Remarks

Check to make sure if DMA Buffer Descriptor fetch is in progress.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint32_t currentBDAddress;  
  
If (PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))  
{  
    void* currentBDAddress = PLIB_SQI_DMABDCurrentAddressGet(MY_SQI_INSTANCE);  
}
```

5.6.23.5.7 DMA Buffer Descriptor Management Functions

5.6.23.5.7.1 PLIB_SQI_DMABDControlWordGet Function

C

```
uint16_t PLIB_SQI_DMABDControlWordGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns current buffer descriptor control word information excluding buffer length. This information is returned in transmit and receive status functions.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Control Word - DMA Buffer Descriptor Control Word

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint16_t dmabdconword;  
  
dmabdconword = PLIB_SQI_DMABDControlWordGet(MY_SQI_INSTANCE);  
  
switch (dmabdconword)  
{  
    case BD_ENABLED: ...;  
    case BD_DISABLED: ...;  
    .  
    .  
    .  
}
```

5.6.23.5.7.2 PLIB_SQI_DMABDReceiveBufferCountGet Function

C

```
uint8_t PLIB_SQI_DMABDReceiveBufferCountGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the current receive buffer space in bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Receive buffer space in bytes.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t bdRxBufCount = PLIB_SQI_DMABDReceiveBufferCountGet(MY_SQI_INSTANCE);
```

5.6.23.5.7.3 PLIB_SQI_DMABDReceiveBufferLengthGet Function

C

```
uint8_t PLIB_SQI_DMABDReceiveBufferLengthGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the current receive length in bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Receive buffer space in bytes.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t bdRxBufLength = PLIB_SQI_DMABDReceiveBufferLengthGet(MY_SQI_INSTANCE);
```

5.6.23.5.7.4 PLIB_SQI_DMABDReceiveStateGet Function

C

```
SQI_BD_STATE PLIB_SQI_DMABDReceiveStateGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the current state of the buffer descriptor in progress.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Status - DMA Buffer Descriptor State

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t bdrxState = PLIB_SQI_DMABDReceiveStateGet(MY_SQI_INSTANCE);  
  
switch (bdrxState)  
{  
    case BD_IDLE: ...;  
    case BD_STATE_FETCH_REQ_PENDING: ...;  
    .  
    .  
    .  
}
```

5.6.23.5.7.5 PLIB_SQI_DMABDTransmitBufferCountGet Function

C

```
uint8_t PLIB_SQI_DMABDTransmitBufferCountGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the current transmit buffer space in bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Transmit buffer space in bytes.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t bdTxBufCount = PLIB_SQI_DMABDTransmitBufferCountGet(MY_SQI_INSTANCE);
```

5.6.23.5.7.6 PLIB_SQI_DMABDTransmitBufferLengthGet Function**C**

```
uint8_t PLIB_SQI_DMABDTransmitBufferLengthGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the current transmit length in bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Transmit buffer space in bytes.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
uint8_t bdTxBufLength = PLIB_SQI_DMABDTransmitBufferLengthGet(MY_SQI_INSTANCE);
```

5.6.23.5.7.7 PLIB_SQI_DMABDTransmitStateGet Function**C**

```
SQI_BD_STATE PLIB_SQI_DMABDTransmitStateGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the current state of the buffer descriptor in progress.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Status - DMA Buffer Descriptor State

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdTxState = PLIB_SQI_DMABDTransmitStateGet(MY_SQI_INSTANCE);

switch (bdTxState)
{
    case BD_IDLE: ...;
    case BD_FETCH_REQ_PENDING: ...;
    .
    .
    .
}
```

5.6.23.5.8 DMA Control and Status Management Functions

5.6.23.5.8.1 PLIB_SQI_DMABDFetchStart Function

C

```
void PLIB_SQI_DMABDFetchStart(
    SQI_MODULE_ID index
);
```

Description

This routine starts the DMA buffer descriptor fetch process.

Preconditions

Make sure the buffer descaptors are setup and the buffer descriptor base address register is pointing to the first/only buffer descriptor. Also ensure any previous BD processing is fixed.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMABDFetchStart(MY_SQI_INSTANCE);
```

5.6.23.5.8.2 PLIB_SQI_DMABDIsBusy Function

C

```
bool PLIB_SQI_DMABDIsBusy(
    SQI_MODULE_ID index
);
```


Description

This routine returns true if DMA buffer descriptor process is busy, if not returns false.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

True if DMA Buffer Descriptor is Busy.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
bool bdBusy = PLIB_SQI_DMABDIsBusy(MY_SQI_INSTANCE)
```

5.6.23.5.8.3 PLIB_SQI_DMABDPollCounterSet Function**C**

```
void PLIB_SQI_DMABDPollCounterSet(  
    SQI_MODULE_ID index,  
    uint16_t pollCount  
);
```

Description

This routine sets the poll counter value that indicates the number of cycles the DMA would wait before fetching the next descriptor word, if the current descriptor fetched was disabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pollControl	Polling value

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
if (PLIB_SQI_DMABDPollIsEnabled(MY_SQI_INSTANCE)  
{  
    PLIB_SQI_DMABDPollCounterSet(MY_SQI_INSTANCE, MY_POLL_VALUE);  
}
```

5.6.23.5.8.4 PLIB_SQI_DMABDPollDisable Function

C

```
void PLIB_SQI_DMABDPollDisable(  
    SQI_MODULE_ID index  
);
```

Description

This routine disables the buffer descriptor polling.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_DMABDPollDisable(MY_SQI_INSTANCE);
```

5.6.23.5.8.5 PLIB_SQI_DMABDPollEnable Function

C

```
void PLIB_SQI_DMABDPollEnable(  
    SQI_MODULE_ID index  
);
```

Description

This routine enables the buffer descriptor polling and works in tandem with poll control register.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Enable this control bit only when you are planning to have dead descriptors in the linked list.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_DMABDPollEnable(MY_SQI_INSTANCE);
```

5.6.23.5.8.6 PLIB_SQI_DMABDPollIsEnabled Function

C

```
bool PLIB_SQI_DMABDPollIsEnabled(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns true/false if DMA buffer descriptor poll is enabled/disabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

True if DMA Poll Control is Enabled, else it's false.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
If (PLIB_SQI_DMABDPollIsEnabled(MY_SQI_INSTANCE))  
{  
    PLIB_SQI_PollControlSet(MY_SQI_INSTANCE, MY_POLL_CONTROL_VALUE);  
}
```

5.6.23.5.8.7 PLIB_SQI_DMABDStateGet Function

C

```
SQI_BD_STATE PLIB_SQI_DMABDStateGet(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns the current state of the buffer descriptor in progress.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Status - DMA Buffer Descriptor State

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdState = PLIB_SQI_DMABDStateGet(MY_SQI_INSTANCE);

switch (bdState)
{
    case BD_IDLE: ...;
    case BD_FETCH_REQ_PENDING: ...;
    .
    .
    .
}
```

5.6.23.5.8.8 PLIB_SQI_DMADisable Function

C

```
void PLIB_SQI_DMADisable(
    SQI_MODULE_ID index
);
```

Description

This routine disables the built-in DMA logic for data transfer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMADisable(MY_SQI_INSTANCE);
```

5.6.23.5.8.9 PLIB_SQI_DMAEnable Function

C

```
void PLIB_SQI_DMAEnable(
    SQI_MODULE_ID index
);
```

Description

This routine enables the built-in DMA logic for data transfer.

Preconditions

DMA buffer descriptors need to be setup before enabling the DMA.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
PLIB_SQI_DMAEnable(MY_SQI_INSTANCE);
```

5.6.23.5.8.10 PLIB_SQI_DMAHasStarted Function**C**

```
bool PLIB_SQI_DMAHasStarted(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns true if DMA process started and false if DMA process has not started.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

True if DMA process has started, false if not.

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the  
// application developer.  
bool dmaStarted = PLIB_SQI_DMAHasStarted (MY_SQI_INSTANCE)
```

5.6.23.5.8.11 PLIB_SQI_DMAIsEnabled Function**C**

```
bool PLIB_SQI_DMAIsEnabled(  
    SQI_MODULE_ID index  
);
```

Description

This routine returns true/false if DMA is enabled/disabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

Boolean Flag - DMA Enable Status true - DMA is enabled false - DMA is disabled

Remarks

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
If (PLIB_SQI_DMAIEnabled(MY_SQI_INSTANCE))
{
    ...
}
```

5.6.23.5.9 Data Types and Constants**5.6.23.5.9.1 SQI_ADDR_BYTES Enumeration****C**

```
typedef enum {
    ADDR_BYTES_4,
    ADDR_BYTES_3,
    ADDR_BYTES_2,
    ADDR_BYTES_1,
    ADDR_BYTES_0
} SQI_ADDR_BYTES;
```

Description

SQI Number of Address Bytes

This macro defines the list of SQI Address Bytes

Members

Members	Description
ADDR_BYTES_4	SQI is set to Transmit 4 Address Bytes
ADDR_BYTES_3	SQI is set to Transmit 3 Address Bytes
ADDR_BYTES_2	SQI is set to Transmit 2 Address Bytes
ADDR_BYTES_1	SQI is set to Transmit 1 Address Bytes
ADDR_BYTES_0	SQI is set to Transmit No Address Bytes

5.6.23.5.9.2 SQI_BD_CTRL_WORD Enumeration**C**

```
typedef enum {
    BD_ENABLE,
    BD_DISABLE,
    CS_1_ACTIVE,
    CS_0_ACTIVE,
    FLASH_SC_ENABLE,
    FLASH_SC_DISABLE,
    SET_LSBF,
```

```

SET_MSBF,
DMA_LANE_QUAD,
DMA_LANE_DUAL,
DMA_LANE_SINGLE,
DMA_IN_TRANSMIT,
DMA_IN_RECEIVE,
LAST_BD,
NOT_LAST_BD,
LAST_PKT,
NOT_LAST_PKT,
PKT_INT_ENABLE,
PKT_INT_DISABLE,
BD_DONE_INT_ENABLE,
BD_DONE_INT_DISABLE,
BD_BUF_LENGTH
} SQI_BD_CTRL_WORD;

```

Description

SQI Buffer Descriptor Control Words

This macro defines the list of SQI Buffer Descriptor Control Word

Members

Members	Description
BD_ENABLE	Current Buffer Descriptor is Enabled
BD_DISABLE	Current Buffer Descriptor is Disabled
CS_1_ACTIVE	Chip Select 1 is Active
CS_0_ACTIVE	Chip Select 0 is Active
FLASH_SC_ENABLE	Automatic Status Check Enabled
FLASH_SC_DISABLE	Automatic Status Check Disabled
SET_LSBF	Least Significant Bit First
SET_MSBF	Most Significant Bit First
DMA_LANE_QUAD	DMA writes/reads in Quad Lane Mode
DMA_LANE_DUAL	DMA writes/reads in Dual Lane Mode
DMA_LANE_SINGLE	DMA writes/reads in Single Lane Mode
DMA_IN_TRANSMIT	DMA is Transmitting
DMA_IN_RECEIVE	DMA is Receiving
LAST_BD	Indicates Last Buffer Descriptor in the List
NOT_LAST_BD	Indicates Non Last Buffer Descriptor in the List
LAST_PKT	Indicates Last Packet of the Data Chunk
NOT_LAST_PKT	Indicates Non Last Packet of the Data Chunk
PKT_INT_ENABLE	Indicates Packet Interrupt is Enabled
PKT_INT_DISABLE	Indicates Packet Interrupt is Disabled
BD_DONE_INT_ENABLE	Indicates Buffer Descriptor Done Interrupt is Enabled
BD_DONE_INT_DISABLE	Indicates Buffer Descriptor Done Interrupt is Disabled
BD_BUF_LENGTH	Indicates Buffer Length

5.6.23.5.9.3 SQI_BD_STATE Enumeration

C

```

typedef enum {
    BD_STATE_DISABLED,
    BD_STATE_DONE,
    BD_STATE_PROCESSING_DATA,

```

```

    BD_STATE_BEING_FETCHED,
    BD_STATE_FETCH_REQ_PENDING,
    BD_STATE_IDLE
} SQI_BD_STATE;

```

Description

SQI Buffer Descriptor (BD) State

This macro defines the list of SQI Buffer Descriptor states

Members

Members	Description
BD_STATE_DISABLED	Fetches Buffer Descriptor is Disabled
BD_STATE_DONE	Fetches Buffer Descriptor Processed
BD_STATE_PROCESSING_DATA	Fetches Buffer Descriptor is in Data transfer phase
BD_STATE_BEING_FETCHED	In the process of Fetching the Buffer Descriptor
BD_STATE_FETCH_REQ_PENDING	Buffer Descriptor Fetch Request Pending
BD_STATE_IDLE	Buffer Descriptor process is idle

5.6.23.5.9.4 SQI_CLK_DIV Enumeration**C**

```

typedef enum {
    CLK_DIV_256,
    CLK_DIV_128,
    CLK_DIV_64,
    CLK_DIV_32,
    CLK_DIV_16,
    CLK_DIV_8,
    CLK_DIV_4,
    CLK_DIV_2,
    CLK_DIV_1
} SQI_CLK_DIV;

```

Description

SQI Clock Divider

This macro defines the list of SQI Clock Divider Values

Members

Members	Description
CLK_DIV_256	SQI Clock is divided by 256
CLK_DIV_128	SQI Clock is divided by 128
CLK_DIV_64	SQI Clock is divided by 64
CLK_DIV_32	SQI Clock is divided by 32
CLK_DIV_16	SQI Clock is divided by 16
CLK_DIV_8	SQI Clock is divided by 8
CLK_DIV_4	SQI Clock is divided by 4
CLK_DIV_2	SQI Clock is divided by 2
CLK_DIV_1	SQI Clock is not divided

5.6.23.5.9.5 SQI_CS_NUM Enumeration

C

```
typedef enum {
    SQI_CS_1,
    SQI_CS_0
} SQI_CS_NUM;
```

Description

SQI Device Select or Chip Select

This macro defines the list of SQI Chip Selects

Members

Members	Description
SQI_CS_1	SQI Chip Select 1
SQI_CS_0	SQI Chip Select 0

5.6.23.5.9.6 SQI_CS_OEN Enumeration

C

```
typedef enum {
    SQI_CS_OEN_BOTH,
    SQI_CS_OEN_1,
    SQI_CS_OEN_0,
    SQI_CS_OEN_NONE
} SQI_CS_OEN;
```

Description

SQI Chip Select Output Enable

This macro defines the list of SQI Chip Select Output Enables

Members

Members	Description
SQI_CS_OEN_BOTH	SQI chip select 0 and 1 are enabled
SQI_CS_OEN_1	SQI chip select 1 is enabled but chip select 0 is disabled
SQI_CS_OEN_0	SQI chip select 0 is enabled but chip select 1 is disabled
SQI_CS_OEN_NONE	SQI chip select 0 and 1 are disabled

5.6.23.5.9.7 SQI_DATA_FORMAT Enumeration

C

```
typedef enum {
    SQI_DATA_FORMAT_LSBF,
    SQI_DATA_FORMAT_MSBF
} SQI_DATA_FORMAT;
```

Description

SQI Data Mode

This macro defines the Data Formats (LSBF/MSBF)

Members

Members	Description
SQI_DATA_FORMAT_LSBF	SQI Data is Least Significant Bit First Formatted
SQI_DATA_FORMAT_MSBF	SQI Data is Most Significant Bit First Formatted

5.6.23.5.9.8 SQI_DATA_MODE Enumeration**C**

```
typedef enum {  
    SQI_DATA_MODE_SF,  
    SQI_DATA_MODE_3,  
    SQI_DATA_MODE_0  
} SQI_DATA_MODE;
```

Description

SQI Data Mode

This macro defines the list of SQI Data Modes

Members

Members	Description
SQI_DATA_MODE_SF	SQI is in Serial Flash Mode
SQI_DATA_MODE_3	SQI is in SPI mode 3
SQI_DATA_MODE_0	SQI is in SPI mode 0

5.6.23.5.9.9 SQI_DATA_OEN Enumeration**C**

```
typedef enum {  
    SQI_DATA_OEN_QUAD,  
    SQI_DATA_OEN_DUAL,  
    SQI_DATA_OEN_SINGLE  
} SQI_DATA_OEN;
```

Description

SQI Data Output Enable

This macro defines the list of SQI Data Output Enables

Members

Members	Description
SQI_DATA_OEN_QUAD	SQI data outputs 3 to 0 are enabled (quad lane mode)
SQI_DATA_OEN_DUAL	SQI data outputs 1 and 0 are enabled (dual lane mode)
SQI_DATA_OEN_SINGLE	SQI data output 0 is enabled (single lane mode)

5.6.23.5.9.10 SQI_DATA_TYPE Type**C**

```
typedef uint32_t SQI_DATA_TYPE;
```

Description

SQI Data Type definition

This data type defines the SQI Data size

Remarks

None

5.6.23.5.9.11 SQI_DUMMY_BYTES Enumeration

C

```
typedef enum {
    DUMMY_BYTES_7,
    DUMMY_BYTES_6,
    DUMMY_BYTES_5,
    DUMMY_BYTES_4,
    DUMMY_BYTES_3,
    DUMMY_BYTES_2,
    DUMMY_BYTE_1,
    DUMMY_BYTES_0
} SQI_DUMMY_BYTES;
```

Description

SQI Number of Dummy Bytes

This macro defines the list of SQI Dummy Bytes after Address Bytes

Members

Members	Description
DUMMY_BYTES_7	SQI is set to Transmit 7 Dummy Bytes
DUMMY_BYTES_6	SQI is set to Transmit 6 Dummy Bytes
DUMMY_BYTES_5	SQI is set to Transmit 5 Dummy Bytes
DUMMY_BYTES_4	SQI is set to Transmit 4 Dummy Bytes
DUMMY_BYTES_3	SQI is set to Transmit 3 Dummy Bytes
DUMMY_BYTES_2	SQI is set to Transmit 2 Dummy Bytes
DUMMY_BYTE_1	SQI is set to Transmit 1 Dummy Bytes
DUMMY_BYTES_0	SQI is set to Transmit No Dummy Bytes

5.6.23.5.9.12 SQI_INTERRUPTS Enumeration

C

```
typedef enum {
    SQI_PKTCOMP,
    SQI_BDDONE,
    SQI_CONTHR,
    SQI_CONEMPTY,
    SQI_CONFULL,
    SQI_RXTHR,
    SQI_RXEMPTY,
    SQI_RXFULL,
    SQI_TXTHR,
    SQI_TXEMPTY,
    SQI_TXFULL,
    SQI_ALL_INT
} SQI_INTERRUPTS;
```

Description

SQI Interrupt List

This macro defines the list of SQI Interrupts

Members

Members	Description
SQI_PKTCOMP	SQI Packet Complete Interrupt (used in DMA mode)
SQI_BDDONE	SQI Buffer Descriptor Done Interrupt (used in DMA mode)
SQI_CONTHR	SQI Control Buffer Threshold Interrupt
SQI_CONEMPTY	SQI Control Buffer Empty Interrupt
SQI_CONFULL	SQI Control Buffer Full Interrupt
SQI_RXTHR	SQI Receive Buffer Threshold Interrupt
SQI_RXEMPTY	SQI Receive Buffer Empty Interrupt
SQI_RXFULL	SQI Receive Buffer Full Interrupt
SQI_TXTHR	SQI Transmit Buffer Threshold Interrupt
SQI_TXEMPTY	SQI Transmit Buffer Empty Interrupt
SQI_TXFULL	SQI Transmit Buffer Full Interrupt
SQI_ALL_INT	All or any of the above interrupts

5.6.23.5.9.13 SQI_LANE_MODE Enumeration**C**

```
typedef enum {
    SQI_LANE_QUAD,
    SQI_LANE_DUAL,
    SQI_LANE_SINGLE
} SQI_LANE_MODE;
```

Description

SQI Lane Mode

This macro defines the list of SQI Lane Modes

Members

Members	Description
SQI_LANE_QUAD	SQI is set into Quad Lane Mode
SQI_LANE_DUAL	SQI is set into Dual Lane Mode
SQI_LANE_SINGLE	SQI is set into Single Lane Mode

5.6.23.5.9.14 SQI_MODE_BYTES Enumeration**C**

```
typedef enum {
    MODE_BYTES_3,
    MODE_BYTES_2,
    MODE_BYTES_1,
    MODE_BYTES_0
} SQI_MODE_BYTES;
```

Description

SQI Number of Mode Bytes

This macro defines the list of SQI Mode Bytes allocated for Mode Code

Members

Members	Description
MODE_BYTES_3	SQI is set to Transmit 3 Mode Bytes
MODE_BYTES_2	SQI is set to Transmit 2 Mode Bytes
MODE_BYTES_1	SQI is set to Transmit 1 Mode Bytes
MODE_BYTES_0	SQI is set to Transmit No Mode Bytes

5.6.23.5.9.15 SQI_MODULE_ID Enumeration**C**

```
typedef enum {  
    SQI_ID_1,  
    SQI_NUMBER_OF_MODULES  
} SQI_MODULE_ID;
```

Description

SQI Module ID

This enumeration identifies the SQI modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

Members

Members	Description
SQI_ID_1	SQI Module 1 ID
SQI_NUMBER_OF_MODULES	Number of available SQI modules

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

5.6.23.5.9.16 SQI_XFER_CMD Enumeration**C**

```
typedef enum {  
    SQI_CMD_RECEIVE,  
    SQI_CMD_TRANSMIT,  
    SQI_CMD_IDLE  
} SQI_XFER_CMD;
```

Description

SQI Transfer Command

This macro defines the list of SQI Transfer Commands

Members

Members	Description
SQI_CMD_RECEIVE	SQI issues a Receive Command
SQI_CMD_TRANSMIT	SQI issues a Transmit Command
SQI_CMD_IDLE	SQI is Idle

5.6.23.5.9.17 SQI_XFER_MODE Enumeration

C

```
typedef enum {  
    SQI_XFER_MODE_XIP,  
    SQI_XFER_MODE_DMA,  
    SQI_XFER_MODE_PIO  
} SQI_XFER_MODE;
```

Description

SQI Transfer Mode

This macro defines the list of SQI Transfer Modes

Members

Members	Description
SQI_XFER_MODE_XIP	SQI is in XIP mode
SQI_XFER_MODE_DMA	SQI is in DMA mode
SQI_XFER_MODE_PIO	SQI is in PIO mode

5.6.23.5.10 Feature Existence Functions

5.6.23.5.10.1 PLIB_SQI_ExistsBDBaseAddress Function

C

```
bool PLIB_SQI_ExistsBDBaseAddress(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDBaseAddress feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDBaseAddressSet](#)
- [PLIB_SQI_DMABDBaseAddressGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDBaseAddress feature is supported on the device
- false - The BDBaseAddress feature is not supported on the device

Remarks

None.

5.6.23.5.10.2 PLIB_SQI_ExistsBDControlWord Function

C

```
bool PLIB_SQI_ExistsBDControlWord(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDControlWord feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDControlWordGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDControlWord feature is supported on the device
- false - The BDControlWord feature is not supported on the device

Remarks

None.

5.6.23.5.10.3 PLIB_SQI_ExistsBDCurrentAddress Function

C

```
bool PLIB_SQI_ExistsBDCurrentAddress(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDCurrentAddress feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDCurrentAddressGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDCurrentAddress feature is supported on the device
- false - The BDCurrentAddress feature is not supported on the device

Remarks

None.

5.6.23.5.10.4 PLIB_SQI_ExistsBDPollCount Function

C

```
bool PLIB_SQI_ExistsBDPollCount(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDPollCount feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDPollCounterSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDPollCount feature is supported on the device
- false - The BDPollCount feature is not supported on the device

Remarks

None.

5.6.23.5.10.5 PLIB_SQI_ExistsBDPollingEnable Function

C

```
bool PLIB_SQI_ExistsBDPollingEnable(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDPollingEnable feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDPollEnable](#)
- [PLIB_SQI_DMABDPollDisable](#)
- [PLIB_SQI_DMABDPollIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDPollingEnable feature is supported on the device
- false - The BDPollingEnable feature is not supported on the device

Remarks

None.

5.6.23.5.10.6 PLIB_SQI_ExistsBDProcessState Function**C**

```
bool PLIB_SQI_ExistsBDProcessState(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDProcessState feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDStateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDProcessState feature is supported on the device
- false - The BDProcessState feature is not supported on the device

Remarks

None.

5.6.23.5.10.7 PLIB_SQI_ExistsBDRxBufCount Function**C**

```
bool PLIB_SQI_ExistsBDRxBufCount(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDRxBufCount feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDReceiveBufferCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDRxBufCount feature is supported on the device
- false - The BDRxBufCount feature is not supported on the device

Remarks

None.

5.6.23.5.10.8 PLIB_SQI_ExistsBDRxLength Function**C**

```
bool PLIB_SQI_ExistsBDRxLength(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDRxLength feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDReceiveBufferLengthGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDRxLength feature is supported on the device
- false - The BDRxLength feature is not supported on the device

Remarks

None.

5.6.23.5.10.9 PLIB_SQI_ExistsBDRxState Function**C**

```
bool PLIB_SQI_ExistsBDRxState(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDRxState feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDReceiveStateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDRxState feature is supported on the device
- false - The BDRxState feature is not supported on the device

Remarks

None.

5.6.23.5.10.10 PLIB_SQI_ExistsBDTxBufCount Function**C**

```
bool PLIB_SQI_ExistsBDTxBufCount(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDTxBufCount feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDTransmitBufferCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDTxBufCount feature is supported on the device
- false - The BDTxBufCount feature is not supported on the device

Remarks

None.

5.6.23.5.10.11 PLIB_SQI_ExistsBDTxLength Function**C**

```
bool PLIB_SQI_ExistsBDTxLength(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDTxLength feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDTransmitBufferLengthGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDTxLength feature is supported on the device
- false - The BDTxLength feature is not supported on the device

Remarks

None.

5.6.23.5.10.12 PLIB_SQI_ExistsBDTxState Function**C**

```
bool PLIB_SQI_ExistsBDTxState(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BDTxState feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDTransmitStateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDTxState feature is supported on the device
- false - The BDTxState feature is not supported on the device

Remarks

None.

5.6.23.5.10.13 PLIB_SQI_ExistsBurstControl Function**C**

```
bool PLIB_SQI_ExistsBurstControl(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the BurstControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_BurstEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BurstControl feature is supported on the device
- false - The BurstControl feature is not supported on the device

Remarks

None.

5.6.23.5.10.14 PLIB_SQI_ExistsChipSelect Function**C**

```
bool PLIB_SQI_ExistsChipSelect(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the ChipSelect feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ChipSelectSet](#)
- [PLIB_SQI_ChipSelectGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ChipSelect feature is supported on the device
- false - The ChipSelect feature is not supported on the device

Remarks

None.

5.6.23.5.10.15 PLIB_SQI_ExistsClockControl Function**C**

```
bool PLIB_SQI_ExistsClockControl(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the ClockControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ClockEnable](#)
- [PLIB_SQI_ClockDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ClockControl feature is supported on the device
- false - The ClockControl feature is not supported on the device

Remarks

None.

5.6.23.5.10.16 PLIB_SQI_ExistsClockDivider Function**C**

```
bool PLIB_SQI_ExistsClockDivider(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the ClockDivider feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ClockDividerSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ClockDivider feature is supported on the device
- false - The ClockDivider feature is not supported on the device

Remarks

None.

5.6.23.5.10.17 PLIB_SQI_ExistsClockReady Function**C**

```
bool PLIB_SQI_ExistsClockReady(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the ClockReady feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ClockIsStable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ClockReady feature is supported on the device
- false - The ClockReady feature is not supported on the device

Remarks

None.

5.6.23.5.10.18 PLIB_SQI_ExistsConBufThreshold Function**C**

```
bool PLIB_SQI_ExistsConBufThreshold(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the ConBufThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ControlBufferThresholdSet](#)
- [PLIB_SQI_ControlBufferThresholdGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ConBufThreshold feature is supported on the device
- false - The ConBufThreshold feature is not supported on the device

Remarks

None.

5.6.23.5.10.19 PLIB_SQI_ExistsConfigWord Function**C**

```
bool PLIB_SQI_ExistsConfigWord(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the ConfigWord feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ConfigWordSet](#)
- [PLIB_SQI_ConfigWordGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ConfigWord feature is supported on the device
- false - The ConfigWord feature is not supported on the device

Remarks

None.

5.6.23.5.10.20 PLIB_SQI_ExistsControlWord Function**C**

```
bool PLIB_SQI_ExistsControlWord(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the ControlWord feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ControlWordSet](#)
- [PLIB_SQI_ControlWordGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ControlWord feature is supported on the device
- false - The ControlWord feature is not supported on the device

Remarks

None.

5.6.23.5.10.21 PLIB_SQI_ExistsCSDeassert Function**C**

```
bool PLIB_SQI_ExistsCSDeassert(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the CSDeassert feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ChipSelectDeassertEnable](#)
- [PLIB_SQI_ChipSelectDeassertDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CSDeassert feature is supported on the device
- false - The CSDeassert feature is not supported on the device

Remarks

None.

5.6.23.5.10.22 PLIB_SQI_ExistsCSOutputEnable Function**C**

```
bool PLIB_SQI_ExistsCSOutputEnable(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the CSOutputEnable feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_CSOutputEnableSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CSOutputEnable feature is supported on the device
- false - The CSOutputEnable feature is not supported on the device

Remarks

None.

5.6.23.5.10.23 PLIB_SQI_ExistsDataFormat Function**C**

```
bool PLIB_SQI_ExistsDataFormat(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the DataFormat feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DataFormatSet](#)

- [PLIB_SQI_DataFormatGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataFormat feature is supported on the device
- false - The DataFormat feature is not supported on the device

Remarks

None.

5.6.23.5.10.24 PLIB_SQI_ExistsDataModeControl Function**C**

```
bool PLIB_SQI_ExistsDataModeControl(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the DataModeControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DataModeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataModeControl feature is supported on the device
- false - The DataModeControl feature is not supported on the device

Remarks

None.

5.6.23.5.10.25 PLIB_SQI_ExistsDataOutputEnable Function**C**

```
bool PLIB_SQI_ExistsDataOutputEnable(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the DataOutputEnable feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DataOutputEnableSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataOutputEnable feature is supported on the device
- false - The DataOutputEnable feature is not supported on the device

Remarks

None.

5.6.23.5.10.26 PLIB_SQI_ExistsDataPinStatus Function**C**

```
bool PLIB_SQI_ExistsDataPinStatus(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the DataPinStatus feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DataLineStatus](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DataPinStatus feature is supported on the device
- false - The DataPinStatus feature is not supported on the device

Remarks

None.

5.6.23.5.10.27 PLIB_SQI_ExistsDmaEnable Function**C**

```
bool PLIB_SQI_ExistsDmaEnable(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the DMAEnable feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMAEnable](#)

- [PLIB_SQI_DMADisable](#)
- [PLIB_SQI_DMAIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DMAEnable feature is supported on the device
- false - The DMAEnable feature is not supported on the device

Remarks

None.

5.6.23.5.10.28 PLIB_SQI_ExistsDMAEngineBusy Function**C**

```
bool PLIB_SQI_ExistsDMAEngineBusy(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the DMAEngineBusy feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDIsBusy](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DMAEngineBusy feature is supported on the device
- false - The DMAEngineBusy feature is not supported on the device

Remarks

None.

5.6.23.5.10.29 PLIB_SQI_ExistsDMAProcessInProgress Function**C**

```
bool PLIB_SQI_ExistsDMAProcessInProgress(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the DMAProcessInProgress feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMAHasStarted](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DMAProcessInProgress feature is supported on the device
- false - The DMAProcessInProgress feature is not supported on the device

Remarks

None.

5.6.23.5.10.30 PLIB_SQI_ExistsEnableControl Function**C**

```
bool PLIB_SQI_ExistsEnableControl(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_Enable](#)
- [PLIB_SQI_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Remarks

None.

5.6.23.5.10.31 PLIB_SQI_ExistsHoldPinControl Function**C**

```
bool PLIB_SQI_ExistsHoldPinControl(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the HoldPinControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_HoldSet](#)
- [PLIB_SQI_HoldClear](#)
- [PLIB_SQI_HoldGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The HoldPinControl feature is supported on the device
- false - The HoldPinControl feature is not supported on the device

Remarks

None.

5.6.23.5.10.32 PLIB_SQI_ExistsInterruptControl Function**C**

```
bool PLIB_SQI_ExistsInterruptControl(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the InterruptControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_InterruptEnable](#)
- [PLIB_SQI_InterruptDisable](#)
- [PLIB_SQI_InterruptIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InterruptControl feature is supported on the device
- false - The InterruptControl feature is not supported on the device

Remarks

None.

5.6.23.5.10.33 PLIB_SQI_ExistsInterruptSignalControl Function**C**

```
bool PLIB_SQI_ExistsInterruptSignalControl(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the InterruptSignalControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_InterruptSignalEnable](#)
- [PLIB_SQI_InterruptSignalDisable](#)
- [PLIB_SQI_InterruptSignalIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InterruptSignalControl feature is supported on the device
- false - The InterruptSignalControl feature is not supported on the device

Remarks

None.

5.6.23.5.10.34 PLIB_SQI_ExistsInterruptStatus Function**C**

```
bool PLIB_SQI_ExistsInterruptStatus(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the InterruptStatus feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_InterruptFlagGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The InterruptStatus feature is supported on the device
- false - The InterruptStatus feature is not supported on the device

Remarks

None.

5.6.23.5.10.35 PLIB_SQI_ExistsLaneMode Function

C

```
bool PLIB_SQI_ExistsLaneMode(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the LaneMode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_LaneModeSet](#)
- [PLIB_SQI_LaneModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LaneMode feature is supported on the device
- false - The LaneMode feature is not supported on the device

Remarks

None.

5.6.23.5.10.36 PLIB_SQI_ExistsReceiveLatch Function

C

```
bool PLIB_SQI_ExistsReceiveLatch(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiveLatch feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ReceiveLatchEnable](#)
- [PLIB_SQI_ReceiveLatchDisable](#)
- [PLIB_SQI_ReceiveLatchGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiveLatch feature is supported on the device
- false - The ReceiveLatch feature is not supported on the device

Remarks

None.

5.6.23.5.10.37 PLIB_SQI_ExistsRxBufferCount Function**C**

```
bool PLIB_SQI_ExistsRxBufferCount(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the RxBufferCount feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_NumberOfReceiveBufferReads](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RxBufferCount feature is supported on the device
- false - The RxBufferCount feature is not supported on the device

Remarks

None.

5.6.23.5.10.38 PLIB_SQI_ExistsRxBufIntThreshold Function**C**

```
bool PLIB_SQI_ExistsRxBufIntThreshold(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the RxBufIntThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_RxBufferThresholdIntSet](#)
- [PLIB_SQI_RxBufferThresholdIntGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RxBufIntThreshold feature is supported on the device

- false - The RxBufIntThreshold feature is not supported on the device

Remarks

None.

5.6.23.5.10.39 PLIB_SQI_ExistsRxBufThreshold Function**C**

```
bool PLIB_SQI_ExistsRxBufThreshold(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the RxBufThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_RxBufferThresholdSet](#)
- [PLIB_SQI_RxBufferThresholdGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RxBufThreshold feature is supported on the device
- false - The RxBufThreshold feature is not supported on the device

Remarks

None.

5.6.23.5.10.40 PLIB_SQI_ExistsRxData Function**C**

```
bool PLIB_SQI_ExistsRxData(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the RxData feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ReceiveData](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RxData feature is supported on the device

- false - The RxData feature is not supported on the device

Remarks

None.

5.6.23.5.10.41 PLIB_SQI_ExistsRxUnderRun Function**C**

```
bool PLIB_SQI_ExistsRxUnderRun(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the RxUnderRun feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ReceiveBufferIsUnderrun](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The RxUnderRun feature is supported on the device
- false - The RxUnderRun feature is not supported on the device

Remarks

None.

5.6.23.5.10.42 PLIB_SQI_ExistsSoftReset Function**C**

```
bool PLIB_SQI_ExistsSoftReset(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the SoftReset feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_SoftReset](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SoftReset feature is supported on the device
- false - The SoftReset feature is not supported on the device

Remarks

None.

5.6.23.5.10.43 PLIB_SQI_ExistsStartDMA Function**C**

```
bool PLIB_SQI_ExistsStartDMA(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the StartDMA feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_DMABDFetchStart](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StartDMA feature is supported on the device
- false - The StartDMA feature is not supported on the device

Remarks

None.

5.6.23.5.10.44 PLIB_SQI_ExistsTransferCommand Function**C**

```
bool PLIB_SQI_ExistsTransferCommand(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the TransferCommand feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_TransferDirectionSet](#)
- [PLIB_SQI_TransferDirectionGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransferCommand feature is supported on the device

- false - The TransferCommand feature is not supported on the device

Remarks

None.

5.6.23.5.10.45 PLIB_SQI_ExistsTransferCount Function**C**

```
bool PLIB_SQI_ExistsTransferCount(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the TransferCount feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_ByteCountSet](#)
- [PLIB_SQI_ByteCountGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransferCount feature is supported on the device
- false - The TransferCount feature is not supported on the device

Remarks

None.

5.6.23.5.10.46 PLIB_SQI_ExistsTransferModeControl Function**C**

```
bool PLIB_SQI_ExistsTransferModeControl(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the TransferModeControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_TransferModeSet](#)
- [PLIB_SQI_TransferModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransferModeControl feature is supported on the device
- false - The TransferModeControl feature is not supported on the device

Remarks

None.

5.6.23.5.10.47 PLIB_SQI_ExistsTxBufferFree Function**C**

```
bool PLIB_SQI_ExistsTxBufferFree(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the TxBufferFree feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_TransmitBufferFreeSpaceGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TxBufferFree feature is supported on the device
- false - The TxBufferFree feature is not supported on the device

Remarks

None.

5.6.23.5.10.48 PLIB_SQI_ExistsTxBufIntThreshold Function**C**

```
bool PLIB_SQI_ExistsTxBufIntThreshold(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the TxBufIntThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_TxBufferThresholdIntSet](#)
- [PLIB_SQI_TxBufferThresholdIntGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TxBufIntThreshold feature is supported on the device
- false - The TxBufIntThreshold feature is not supported on the device

Remarks

None.

5.6.23.5.10.49 PLIB_SQI_ExistsTxBufThreshold Function**C**

```
bool PLIB_SQI_ExistsTxBufThreshold(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the TxBufThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_TxBufferThresholdSet](#)
- [PLIB_SQI_TxBufferThresholdGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TxBufThreshold feature is supported on the device
- false - The TxBufThreshold feature is not supported on the device

Remarks

None.

5.6.23.5.10.50 PLIB_SQI_ExistsTxData Function**C**

```
bool PLIB_SQI_ExistsTxData(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the TxData feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_TransmitData](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TxData feature is supported on the device
- false - The TxData feature is not supported on the device

Remarks

None.

5.6.23.5.10.51 PLIB_SQI_ExistsTxOverFlow Function**C**

```
bool PLIB_SQI_ExistsTxOverFlow(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the TxOverFlow feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_TransmitBufferHasOverflowed](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TxOverFlow feature is supported on the device
- false - The TxOverFlow feature is not supported on the device

Remarks

None.

5.6.23.5.10.52 PLIB_SQI_ExistsWPPinControl Function**C**

```
bool PLIB_SQI_ExistsWPPinControl(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the WPPinControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_WriteProtectSet](#)

- [PLIB_SQI_WriteProtectClear](#)
- [PLIB_SQI_WriteProtectGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WPPinControl feature is supported on the device
- false - The WPPinControl feature is not supported on the device

Remarks

None.

5.6.23.5.10.53 PLIB_SQI_ExistsXIPChipSelect Function**C**

```
bool PLIB_SQI_ExistsXIPChipSelect(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the XIPChipSelect feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_XIPChipSelectSet](#)
- [PLIB_SQI_XIPChipSelectGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The XIPChipSelect feature is supported on the device
- false - The XIPChipSelect feature is not supported on the device

Remarks

None.

5.6.23.5.10.54 PLIB_SQI_ExistsXIPControlWord1 Function**C**

```
bool PLIB_SQI_ExistsXIPControlWord1(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the XIPControlWord1 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_XIPControlWord1Set](#)
- [PLIB_SQI_XIPControlWord1Get](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The XIPControlWord1 feature is supported on the device
- false - The XIPControlWord1 feature is not supported on the device

Remarks

None.

5.6.23.5.10.55 PLIB_SQI_ExistsXIPControlWord2 Function**C**

```
bool PLIB_SQI_ExistsXIPControlWord2(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the XIPControlWord2 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_XIPControlWord2Set](#)
- [PLIB_SQI_XIPControlWord2Get](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The XIPControlWord2 feature is supported on the device
- false - The XIPControlWord2 feature is not supported on the device

Remarks

None.

5.6.23.5.10.56 PLIB_SQI_ExistsXIPLaneMode Function

C

```
bool PLIB_SQI_ExistsXIPLaneMode(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the XIPLaneMode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_XIPLaneModeSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The XIPLaneMode feature is supported on the device
- false - The XIPLaneMode feature is not supported on the device

Remarks

None.

5.6.23.5.10.57 PLIB_SQI_ExistsXIPModeBytes Function

C

```
bool PLIB_SQI_ExistsXIPModeBytes(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the XIPModeBytes feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_XIPModeBytesSet](#)
- [PLIB_SQI_XIPModeBytesGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The XIPModeBytes feature is supported on the device
- false - The XIPModeBytes feature is not supported on the device

Remarks

None.

5.6.23.5.10.58 PLIB_SQI_ExistsXIPModeCode Function

C

```
bool PLIB_SQI_ExistsXIPModeCode(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the XIPModeCode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_XIPModeCodeSet](#)
- [PLIB_SQI_XIPModeCodeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The XIPModeCode feature is supported on the device
- false - The XIPModeCode feature is not supported on the device

Remarks

None.

5.6.23.5.10.59 PLIB_SQI_ExistsXIPNumberOfAddressBytes Function

C

```
bool PLIB_SQI_ExistsXIPNumberOfAddressBytes(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the XIPNumberOfAddressBytes feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_XIPAddressBytesSet](#)
- [PLIB_SQI_XIPAddressBytesGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The XIPNumberOfAddressBytes feature is supported on the device
- false - The XIPNumberOfAddressBytes feature is not supported on the device

Remarks

None.

5.6.23.5.10.60 PLIB_SQI_ExistsXIPNumberOfDummyBytes Function**C**

```
bool PLIB_SQI_ExistsXIPNumberOfDummyBytes(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the XIPNumberOfDummyBytes feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_XIPDummyBytesSet](#)
- [PLIB_SQI_XIPDummyBytesGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The XIPNumberOfDummyBytes feature is supported on the device
- false - The XIPNumberOfDummyBytes feature is not supported on the device

Remarks

None.

5.6.23.5.10.61 PLIB_SQI_ExistsXIPReadOpCode Function**C**

```
bool PLIB_SQI_ExistsXIPReadOpCode(  
    SQI_MODULE_ID index  
);
```

Description

This function identifies whether the XIPReadOpCode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- [PLIB_SQI_XIPReadOpcodeSet](#)
- [PLIB_SQI_XIPReadOpcodeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The XIPReadOpCode feature is supported on the device
- false - The XIPReadOpCode feature is not supported on the device

Remarks

None.

5.6.23.6 Files

Files

Name	Description
plib_sqi.h	SQI PLIB Interface Header for common definitions

Description

5.6.23.6.1 plib_sqi.h

SQI Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the SQI PLIB.

Functions

	Name	Description
≡	PLIB_SQI_BurstEnable	This bit sets the burst enable (BURSTEN) function for higher throughput. This function is artifact of the system bus architecture.
≡	PLIB_SQI_ByteCountGet	Returns the current transmit/receive count.
≡	PLIB_SQI_ByteCountSet	Sets the transmit/receive count.
≡	PLIB_SQI_ChipSelectDeassertDisable	Clears the chip select de-assert.
≡	PLIB_SQI_ChipSelectDeassertEnable	Sets the chip select de-assert.
≡	PLIB_SQI_ChipSelectGet	Returns the chip select that is currently active.
≡	PLIB_SQI_ChipSelectSet	Activates the chip select.
≡	PLIB_SQI_ClockDisable	Disables the SQI transfer clock.
≡	PLIB_SQI_ClockDividerGet	Returns the SQI clock divider value.
≡	PLIB_SQI_ClockDividerSet	Sets the SQI clock (that drives the SQI protocol) divider value. Divides the base clock to generate the SQI clock.
≡	PLIB_SQI_ClockEnable	Enables the SQI transfer clock.
≡	PLIB_SQI_ClockIsStable	Returns SQI transfer clock state.
≡	PLIB_SQI_ConfigWordGet	Get the SQI configuration word.
≡	PLIB_SQI_ConfigWordSet	Sets SQI Configuration Word.
≡	PLIB_SQI_ControlBufferThresholdGet	Returns the transmit buffer space in bytes.
≡	PLIB_SQI_ControlBufferThresholdSet	Sets the control buffer threshold value.
≡	PLIB_SQI_ControlWordGet	Get the SQI control word.
≡	PLIB_SQI_ControlWordSet	Sets SQI Control Word.
≡	PLIB_SQI_CSOutputEnableSelect	Selects the output enables on SQI chip select pins.

	PLIB_SQI_DataFormatGet	Returns the data format to Least Significant Bit First or LITTLE-ENDIAN.
	PLIB_SQI_DataFormatSet	Sets the data format to Least Significant Bit First or LITTLE-ENDIAN.
	PLIB_SQI_DataLineStatus	Returns the logical status of the SQI data lines.
	PLIB_SQI_DataModeGet	Returns the SQI data mode of operation (SPI Mode0/Mode3).
	PLIB_SQI_DataModeSet	Sets the SQI data mode of operation (SPI Mode0/Mode3/Serial Flash).
	PLIB_SQI_DataOutputEnableSelect	Selects the output enables on SQI data outputs.
	PLIB_SQI_Disable	Disables the SQI module.
	PLIB_SQI_DMABDBaseAddressGet	Returns the address of the base buffer descriptor.
	PLIB_SQI_DMABDBaseAddressSet	Sets the address of the base buffer descriptor.
	PLIB_SQI_DMABDControlWordGet	Returns Current Buffer Descriptor Control Word Information.
	PLIB_SQI_DMABDCurrentAddressGet	Returns the address of the current buffer descriptor in process.
	PLIB_SQI_DMABDFetchStart	Start the DMA buffer descriptor fetch process.
	PLIB_SQI_DMABDIsBusy	Returns true if DMA Buffer Descriptor is busy.
	PLIB_SQI_DMABDPollCounterSet	Sets the poll counter value.
	PLIB_SQI_DMABDPollDisable	Disables the buffer descriptor polling.
	PLIB_SQI_DMABDPollEnable	Enables the buffer descriptor polling.
	PLIB_SQI_DMABDPollIsEnabled	Returns true if DAM buffer descriptor poll is eanbled and false if it's disabled.
	PLIB_SQI_DMABDReceiveBufferCountGet	Returns the receive buffer space in bytes.
	PLIB_SQI_DMABDReceiveBufferLengthGet	Returns the receive length in bytes.
	PLIB_SQI_DMABDReceiveStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMABDStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMABDTransmitBufferCountGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_DMABDTransmitBufferLengthGet	Returns the transmit length in bytes.
	PLIB_SQI_DMABDTransmitStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMADisable	Disables the built-in DMA logic.
	PLIB_SQI_DMAEnable	Enables the built-in DMA logic.
	PLIB_SQI_DMAHasStarted	Returns true if DMA process has started if not returns false.
	PLIB_SQI_DMAIsEnabled	Returns true if DAM is eanbled and false if it's disabled.
	PLIB_SQI_Enable	Enables the SQI module.
	PLIB_SQI_ExistsBDBaseAddress	Identifies whether the BDBaseAddress feature exists on the SQI module
	PLIB_SQI_ExistsBDControlWord	Identifies whether the BDControlWord feature exists on the SQI module
	PLIB_SQI_ExistsBDCurrentAddress	Identifies whether the BDCurrentAddress feature exists on the SQI module
	PLIB_SQI_ExistsBDPollCount	Identifies whether the BDPollCount feature exists on the SQI module
	PLIB_SQI_ExistsBDPollingEnable	Identifies whether the BDPollingEnable feature exists on the SQI module
	PLIB_SQI_ExistsBDProcessState	Identifies whether the BDProcessState feature exists on the SQI module
	PLIB_SQI_ExistsBDRxBufCount	Identifies whether the BDRxBufCount feature exists on the SQI module
	PLIB_SQI_ExistsBDRxLength	Identifies whether the BDRxLength feature exists on the SQI module
	PLIB_SQI_ExistsBDRxState	Identifies whether the BDRxState feature exists on the SQI module

	<u>PLIB_SQI_ExistsBDTxBufCount</u>	Identifies whether the BDTxBufCount feature exists on the SQI module
	<u>PLIB_SQI_ExistsBDTxLength</u>	Identifies whether the BDTxLength feature exists on the SQI module
	<u>PLIB_SQI_ExistsBDTxState</u>	Identifies whether the BDTxState feature exists on the SQI module
	<u>PLIB_SQI_ExistsBurstControl</u>	Identifies whether the BurstControl feature exists on the SQI module
	<u>PLIB_SQI_ExistsChipSelect</u>	Identifies whether the ChipSelect feature exists on the SQI module
	<u>PLIB_SQI_ExistsClockControl</u>	Identifies whether the ClockControl feature exists on the SQI module
	<u>PLIB_SQI_ExistsClockDivider</u>	Identifies whether the ClockDivider feature exists on the SQI module
	<u>PLIB_SQI_ExistsClockReady</u>	Identifies whether the ClockReady feature exists on the SQI module
	<u>PLIB_SQI_ExistsConBufThreshold</u>	Identifies whether the ConBufThreshold feature exists on the SQI module
	<u>PLIB_SQI_ExistsConfigWord</u>	Identifies whether the ConfigWord feature exists on the SQI module
	<u>PLIB_SQI_ExistsControlWord</u>	Identifies whether the ControlWord feature exists on the SQI module
	<u>PLIB_SQI_ExistsCSDeassert</u>	Identifies whether the CSDeassert feature exists on the SQI module
	<u>PLIB_SQI_ExistsCSOutputEnable</u>	Identifies whether the CSOutputEnable feature exists on the SQI module
	<u>PLIB_SQI_ExistsDataFormat</u>	Identifies whether the DataFormat feature exists on the SQI module
	<u>PLIB_SQI_ExistsDataModeControl</u>	Identifies whether the DataModeControl feature exists on the SQI module
	<u>PLIB_SQI_ExistsDataOutputEnable</u>	Identifies whether the DataOutputEnable feature exists on the SQI module
	<u>PLIB_SQI_ExistsDataPinStatus</u>	Identifies whether the DataPinStatus feature exists on the SQI module
	<u>PLIB_SQI_ExistsDmaEnable</u>	Identifies whether the DMAEnable feature exists on the SQI module
	<u>PLIB_SQI_ExistsDMAEngineBusy</u>	Identifies whether the DMAEngineBusy feature exists on the SQI module
	<u>PLIB_SQI_ExistsDMAProcessInProgress</u>	Identifies whether the DMAProcessInProgress feature exists on the SQI module
	<u>PLIB_SQI_ExistsEnableControl</u>	Identifies whether the EnableControl feature exists on the SQI module
	<u>PLIB_SQI_ExistsHoldPinControl</u>	Identifies whether the HoldPinControl feature exists on the SQI module
	<u>PLIB_SQI_ExistsInterruptControl</u>	Identifies whether the InterruptControl feature exists on the SQI module
	<u>PLIB_SQI_ExistsInterruptSignalControl</u>	Identifies whether the InterruptSignalControl feature exists on the SQI module
	<u>PLIB_SQI_ExistsInterruptStatus</u>	Identifies whether the InterruptStatus feature exists on the SQI module
	<u>PLIB_SQI_ExistsLaneMode</u>	Identifies whether the LaneMode feature exists on the SQI module
	<u>PLIB_SQI_ExistsReceiveLatch</u>	Identifies whether the ReceiveLatch feature exists on the SQI module
	<u>PLIB_SQI_ExistsRxBufferCount</u>	Identifies whether the RxBufferCount feature exists on the SQI module
	<u>PLIB_SQI_ExistsRxBufIntThreshold</u>	Identifies whether the RxBufIntThreshold feature exists on the SQI module
	<u>PLIB_SQI_ExistsRxBufThreshold</u>	Identifies whether the RxBufThreshold feature exists on the SQI module
	<u>PLIB_SQI_ExistsRxData</u>	Identifies whether the RxData feature exists on the SQI module
	<u>PLIB_SQI_ExistsRxUnderRun</u>	Identifies whether the RxUnderRun feature exists on the SQI module

	PLIB_SQI_ExistsSoftReset	Identifies whether the SoftReset feature exists on the SQI module
	PLIB_SQI_ExistsStartDMA	Identifies whether the StartDMA feature exists on the SQI module
	PLIB_SQI_ExistsTransferCommand	Identifies whether the TransferCommand feature exists on the SQI module
	PLIB_SQI_ExistsTransferCount	Identifies whether the TransferCount feature exists on the SQI module
	PLIB_SQI_ExistsTransferModeControl	Identifies whether the TransferModeControl feature exists on the SQI module
	PLIB_SQI_ExistsTxBufferFree	Identifies whether the TxBufferFree feature exists on the SQI module
	PLIB_SQI_ExistsTxBufIntThreshold	Identifies whether the TxBufIntThreshold feature exists on the SQI module
	PLIB_SQI_ExistsTxBufThreshold	Identifies whether the TxBufThreshold feature exists on the SQI module
	PLIB_SQI_ExistsTxData	Identifies whether the TxData feature exists on the SQI module
	PLIB_SQI_ExistsTxOverFlow	Identifies whether the TxOverFlow feature exists on the SQI module
	PLIB_SQI_ExistsWPPinControl	Identifies whether the WPPinControl feature exists on the SQI module
	PLIB_SQI_ExistsXIPChipSelect	Identifies whether the XIPChipSelect feature exists on the SQI module
	PLIB_SQI_ExistsXIPControlWord1	Identifies whether the XIPControlWord1 feature exists on the SQI module
	PLIB_SQI_ExistsXIPControlWord2	Identifies whether the XIPControlWord2 feature exists on the SQI module
	PLIB_SQI_ExistsXIPLaneMode	Identifies whether the XIPLaneMode feature exists on the SQI module
	PLIB_SQI_ExistsXIPModeBytes	Identifies whether the XIPModeBytes feature exists on the SQI module
	PLIB_SQI_ExistsXIPModeCode	Identifies whether the XIPModeCode feature exists on the SQI module
	PLIB_SQI_ExistsXIPNumberOfAddressBytes	Identifies whether the XIPNumberOfAddressBytes feature exists on the SQI module
	PLIB_SQI_ExistsXIPNumberOfDummyBytes	Identifies whether the XIPNumberOfDummyBytes feature exists on the SQI module
	PLIB_SQI_ExistsXIPReadOpCode	Identifies whether the XIPReadOpCode feature exists on the SQI module
	PLIB_SQI_HoldClear	This bit clears the hold function to be disabled on SQID3 in single and dual lane modes.
	PLIB_SQI_HoldGet	This bit gets the status of HOLD function on SQID3 pin.
	PLIB_SQI_HoldSet	This bit sets the hold function to be enabled on SQID3 in single or dual lane modes.
	PLIB_SQI_InterruptDisable	Disables the interrupt source.
	PLIB_SQI_InterruptEnable	Enables the interrupt source passed.
	PLIB_SQI_InterruptFlagGet	Return SQI Interrupt flag status.
	PLIB_SQI_InterruptIsEnabled	Returns the interrupt state (enabled/disabled).
	PLIB_SQI_InterruptSignalDisable	Disables the interrupt signal source.
	PLIB_SQI_InterruptSignalEnable	Enables the interrupt signal source passed.
	PLIB_SQI_InterruptSignalIsEnabled	Returns the interrupt signal state (enabled/disabled).
	PLIB_SQI_LaneModeGet	Returns the lane mode (single/dual/quad).
	PLIB_SQI_LaneModeSet	Sets the data lane mode (single/dual/quad).

	<u>PLIB_SQI_NumberOfReceiveBufferReads</u>	Returns the number of receive buffer reads.
	<u>PLIB_SQI_ReceiveBufferIsUnderrun</u>	Returns the status of receive buffer.
	<u>PLIB_SQI_ReceiveData</u>	Reads the data from the receive buffer.
	<u>PLIB_SQI_ReceiveLatchDisable</u>	Disables the receive latch so receive data is discarded when in transmit mode.
	<u>PLIB_SQI_ReceiveLatchEnable</u>	Enable the receive latch so receive data is latched during transmit mode.
	<u>PLIB_SQI_ReceiveLatchGet</u>	Returns the receive latch status in transmit mode.
	<u>PLIB_SQI_RxBufferThresholdGet</u>	Returns the receive command threshold.
	<u>PLIB_SQI_RxBufferThresholdIntGet</u>	Sets the receive buffer threshold interrupt.
	<u>PLIB_SQI_RxBufferThresholdIntSet</u>	Sets the receive buffer threshold for interrupt.
	<u>PLIB_SQI_RxBufferThresholdSet</u>	Sets the receive command threshold.
	<u>PLIB_SQI_SoftReset</u>	This bit issues a soft reset to the SQI module.
	<u>PLIB_SQI_TransferDirectionGet</u>	Returns the transfer command.
	<u>PLIB_SQI_TransferDirectionSet</u>	Sets the transfer command.
	<u>PLIB_SQI_TransferModeGet</u>	Returns the SQI transfer mode (PIO/DMA/XIP) of operation.
	<u>PLIB_SQI_TransferModeSet</u>	Sets the SQI transfer mode (PIO/DMA/XIP) of operation.
	<u>PLIB_SQI_TransmitBufferFreeSpaceGet</u>	Returns the number of transmit buffer words available.
	<u>PLIB_SQI_TransmitBufferHasOverflowed</u>	Returns the current status of the transmit buffer.
	<u>PLIB_SQI_TransmitData</u>	Writes data into the SQI transmit buffer.
	<u>PLIB_SQI_TxBufferThresholdGet</u>	Returns the transmit command threshold value.
	<u>PLIB_SQI_TxBufferThresholdIntGet</u>	Returns the value to trigger the transmit buffer threshold interrupt.
	<u>PLIB_SQI_TxBufferThresholdIntSet</u>	Sets the value to trigger the transmit buffer threshold interrupt.
	<u>PLIB_SQI_TxBufferThresholdSet</u>	Sets the transmit command threshold.
	<u>PLIB_SQI_WriteProtectClear</u>	This bit clears the Write-Protect function to be disabled on SQID2 in single or dual lane modes.
	<u>PLIB_SQI_WriteProtectGet</u>	This bit gets the Write-Protect state of Write_Protect Feature on SQID2.
	<u>PLIB_SQI_WriteProtectSet</u>	This bit sets the Write-Protect function to be enabled on SQID2 in single or dual Lane modes only.
	<u>PLIB_SQI_XIPAddressBytesGet</u>	Returns the number of address bytes.
	<u>PLIB_SQI_XIPAddressBytesSet</u>	Sets the number of address bytes.
	<u>PLIB_SQI_XIPChipSelectGet</u>	Returns the current active chip select.
	<u>PLIB_SQI_XIPChipSelectSet</u>	Activates the chip select in XIP mode.
	<u>PLIB_SQI_XIPControlWord1Get</u>	Get the XIP mode control word 1.
	<u>PLIB_SQI_XIPControlWord1Set</u>	Sets the XIP mode control word 1.
	<u>PLIB_SQI_XIPControlWord2Get</u>	Get the XIP mode control word 2.
	<u>PLIB_SQI_XIPControlWord2Set</u>	Sets the XIP mode control word 2.
	<u>PLIB_SQI_XIPDummyBytesGet</u>	Sets the number of dummy bytes.
	<u>PLIB_SQI_XIPDummyBytesSet</u>	Sets the number of dummy bytes.
	<u>PLIB_SQI_XIPLaneModeGet</u>	Returns the lane (single/dual/quad) mode in XIP mode.
	<u>PLIB_SQI_XIPLaneModeSet</u>	Selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.
	<u>PLIB_SQI_XIPModeBytesGet</u>	Returns the number of bytes used for mode code command.
	<u>PLIB_SQI_XIPModeBytesSet</u>	Allocates the bytes for mode code command.
	<u>PLIB_SQI_XIPModeCodeGet</u>	Returns the mode code op-code.
	<u>PLIB_SQI_XIPModeCodeSet</u>	Sets the mode code command.

	PLIB_SQI_XIPReadOpcodeGet	Returns the read opcode in XIP mode.
	PLIB_SQI_XIPReadOpcodeSet	Sets the read opcode in XIP mode.

File Name

plib_sqi.h

Company

Microchip Technology Inc.

5.6.24 Timer Peripheral Library

5.6.24.1 Introduction

Timer Peripheral Library
for
Microchip Microcontrollers



This library provides a low-level abstraction of the Timer module on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

Overview of Timers

The timer function is one of the basic features of a microcontroller. Timers are useful for generating accurate time-based periodic interrupts for software application or real-time operating systems. Other uses include counting external pulses or accurate timing measurement of external events using the timer's gate functions or producing precise time delays. Some timers can also be used to control timing of other peripherals.

5.6.24.2 Release Notes

MPLAB Harmony Version: v0.70b Timer Peripheral Library Version : 0.70b Release Date: 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.24.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED AS IS MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.24.4 Using the Library

This topic describes the basic architecture of the Timer Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_tmr.h](#)

The interface to the Timer Peripheral Library is defined in the "[plib_tmr.h](#)" header file. Any C language source (.c) file that uses the Timer Peripheral Library must include "[plib_tmr.h](#)".

Library File:

The Timer Peripheral Library is part of the processor-specific peripheral library archive (.a) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

5.6.24.4.1 Hardware Abstraction Model

This library provides the low-level abstraction of the Timer module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all the functionality of the available timers on the device. Refer to the specific device data sheet or related family reference manual to determine the set of functions that are supported for each Timer module on your device.

Description

Microchip timers can be classified as:

- Overflow-based
- Period match-based

These timers are essentially counters of a specific size (8-, 16-, or 32-bits wide) that increment based on the clock cycle and the

timer prescaler. An application can monitor these counters to determine how much time has elapsed.

The prescaler determines the timer granularity (resolution). It is a mechanism for generating the clock for the timer by the CPU clock. Every CPU has a clock source and the frequency of this source decides the rate at which instructions are executed by the processor. Most of the timers have the option of providing input clock prescalers, while some have the optional postscalers. The prescaler is used to divide the clock frequency of the Timer module and produce a clock for the timer.

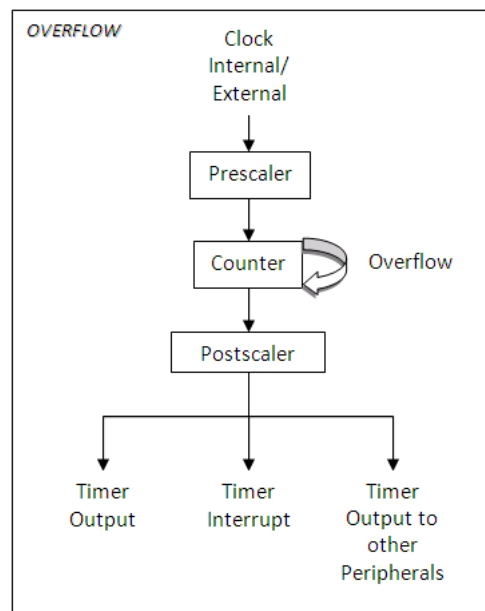
Each timer can be configured with a different source (internal or external) meaning, the source of the timer's input clock is selectable. Both internal and external sources are available, depending on the part in use, with some timers providing a clock synchronization mechanisms with reference to external source.

On some timers the clock edge, either the rising or the falling edge, on which the increment occurs, is selectable.

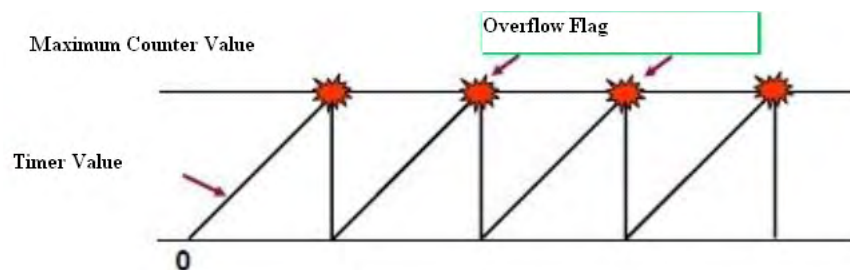
Overflow-based Timers

Overflow-based timers provide timer outputs or generate interrupts after a rollover from the maximum possible timer count.

Overflow Timer Abstraction Diagram



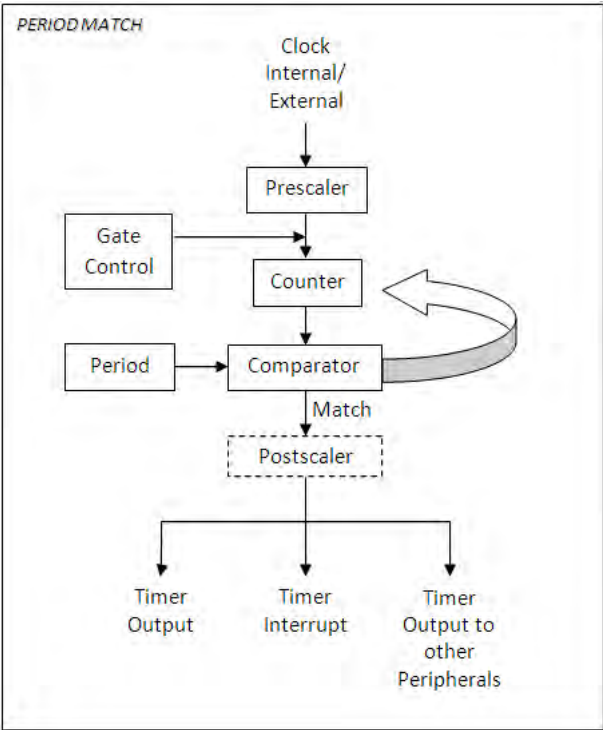
An overflow interrupt is triggered whenever the timer register overflows (i.e., reaches its maximum value based on its size as depicted by the following diagram).



Period Match-based Timers

Period match-based timers provide the timer outputs or generate interrupts upon a match between the counter's count value against the preprogrammed period.

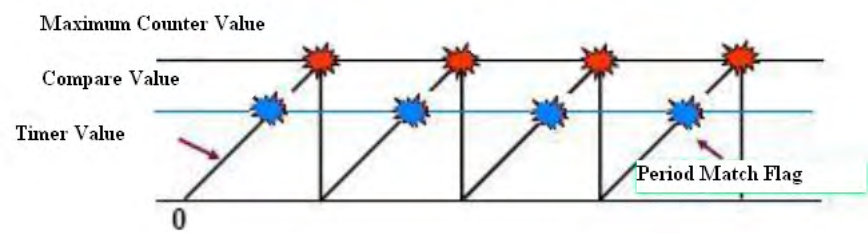
Period Match timer Abstraction Diagram



Gated Time Accumulation/Gated Control Mode

The Gated Time Accumulation/Gate control mode allows the internal timer to increment based upon the duration of the high time applied to the timer input pin. In the Gated Time Accumulation mode, the timer clock source is derived from the internal system clock. When the input clock pin state is high, the timer will count up until a period match has occurred, or the input pin state is changed to a low state. A pin state transition from high-to-low will trigger an interrupt.

A period/compare match interrupt is issued by a timer whenever the value of the timer becomes equal to a certain predefined value as depicted by the following diagram,



5.6.24.4.2 Library Overview

The library interface routines are divided into various subsections, each of the sub-sections addresses one of the blocks or the overall operation of the Timer module.

Library Interface Section	Description

5.6.24.4.3 How the Library Works

Timer modules can operate in the following modes:

- Synchronous Clock Counter
- Synchronous External Clock Counter
- Asynchronous External Clock Counter
- Gated Timer

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

5.6.24.4.3.1 Synchronous Internal Clock Counter

This section provides information and examples for setting up Synchronous Internal Clock Counter operation.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes for your device.

Description

Synchronous Clock Counter operation provides the following capabilities:

- Elapsed time measurements
- Time delays
- Periodic timer interrupts

Setup

Use the following steps for synchronous internal clock counter setup:

1. Some timer modules support multiple resolutions, which can be selected using [PLIB_TMR_Mode8BitEnable](#), [PLIB_TMR_Mode16BitEnable](#), or [PLIB_TMR_Mode32BitEnable](#).
2. Set the start value of the timer using [PLIB_TMR_Counter8BitSet](#) and [PLIB_TMR_Counter16BitSet](#) and [PLIB_TMR_Counter32BitSet](#).
3. The period register value can be set using [PLIB_TMR_Period8BitSet](#), [PLIB_TMR_Period16BitSet](#) or [PLIB_TMR_Period32BitSet](#). Use the appropriate function based on the resolution of the timer.
4. Clear the timer register using [PLIB_TMR_Counter8BitClear](#), [PLIB_TMR_Counter16BitClear](#), or [PLIB_TMR_Counter32BitClear](#).
5. In this mode, the input clock source for the timer is the internal clock, and is selected using [PLIB_TMR_ClockSourceSelect](#), depending on which clock is used as the internal clock in the timer module of the device.
6. The prescaler for the clock can be selected using [PLIB_TMR_PrescaleSelect](#). Pass in the desired prescaler into the function from the enumeration [TMR_PRESCALE](#).
7. Timer count is incremented on the rising timer clock edge. On some microcontrollers, this is can be changed using [PLIB_TMR_ClockSourceEdgeSelect](#). with the appropriate parameter for the falling edge (i.e., increment on high-to-low).
8. On some microcontrollers, the timer output can go through a postscaler. The postscaler can be configured using [PLIB_TMR_PostscaleSelect](#).
9. Start the timer once the setup is complete using [PLIB_TMR_Start](#). The timer operation can be stopped using [PLIB_TMR_Stop](#).

Example: Synchronous Counter (Internal Clock) Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);
```

```
// Set the prescaler, and set the clock source as internal
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_TMRX_VALUE_1);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0xFFFF);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

Timer interrupt is generated if enabled, when the timer count matches the period value if available, or when the counter overflows.

Example: Changing Counter Value During Timer Operation

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.
uint16_t timer_value;

// Clear the timer
timer_value = PLIB_TMR_Counter16BitGet(MY_TIMER_INSTANCE);

if(timer_value > 0x0fff0)
{
    PLIB_TMR_Counter16BitSet(MY_TIMER_INSTANCE, 0x000a);
}
```

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes for your device.

5.6.24.4.3.2 Synchronous External Clock Counter

This section provides information and examples for Synchronous External Clock Counter operation.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes for your device.

Description

Synchronous external clock counter operation provides the following capabilities:

- Counting periodic or non-periodic pulses
- Use external clock as time base for timers

Setup

Use the following steps for synchronous external clock counter setup:

1. Some timer modules support multiple resolutions, which can be selected using [PLIB_TMR_Mode8BitEnable](#), [PLIB_TMR_Mode16BitEnable](#), or [PLIB_TMR_Mode32BitEnable](#).
2. The period register value can be set using [PLIB_TMR_Period8BitSet](#), [PLIB_TMR_Period16BitSet](#), or [PLIB_TMR_Period32BitSet](#). Use the appropriate function based on the resolution of the timer.
3. Set the start value of the timer using [PLIB_TMR_Counter8BitSet](#), [PLIB_TMR_Counter16BitSet](#), or [PLIB_TMR_Counter32BitSet](#).
4. Some timer modules, can select an external clock as its input source. Use [PLIB_TMR_ClockSourceSelect](#) to select the external clock source.
5. The clock source is synchronized with the internal clock for synchronous operation. Use [PLIB_TMR_ClockSourceExternalSyncEnable](#) to synchronize the external clock source.

6. The prescaler for the clock can be selected using [PLIB_TMR_PrescaleSelect](#). Pass in the desired prescaler into the function from the enumeration [TMR_PRESCALE](#).
7. Timer count is incremented on the rising timer clock edge. On some microcontrollers, this is can be changed using [PLIB_TMR_ClockSourceEdgeSelect](#), with the appropriate parameter for the falling edge (i.e., increment on high-to-low).
8. On some microcontrollers, the timer output can go through a postscaler. The postscaler can be configured using [PLIB_TMR_PostscaleSelect](#).
9. Clear the timer register using [PLIB_TMR_Counter8BitClear](#), [PLIB_TMR_Counter16BitClear](#), or [PLIB_TMR_Counter32BitClear](#).
10. Start the timer once the setup is complete using [PLIB_TMR_Start](#). The timer operation can be stopped using [PLIB_TMR_Stop](#).

Example: Synchronous Counter (External Clock) Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and enable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_256);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
PLIB_TMR_ClockSourceExternalSyncEnable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0xFFFF);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

A timer interrupt is generated if enabled, when the timer count matches the period value if available, or when the counter overflows.

Example: Changing Counter Value During Timer Operation

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.
uint16_t timer_value;

// read the current timer value
timer_value = PLIB_TMR_Counter16BitGet(MY_TIMER_INSTANCE);

if(timer_value > 0x0fff0)
{
    PLIB_TMR_Counter16BitSet(MY_TIMER_INSTANCE, 0x000a);
}
```

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes for your device.

5.6.24.4.3.3 Asynchronous Counter

This section provides information and examples for setting up Asynchronous Counter operation.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes for your device.

Description

Asynchronous Counter operation provides the following capabilities:

- The timer can operate in Sleep mode and can generate an interrupt on a period match
- The processor can wake up from Sleep on the timer interrupt
- The timer can be clocked from the Secondary Oscillator for real-time clock applications

Setup

Use the following steps for Asynchronous Counter setup:

1. Some timer modules support multiple resolutions, which can be selected using [PLIB_TMR_Mode8BitEnable](#), [PLIB_TMR_Mode16BitEnable](#), or [PLIB_TMR_Mode32BitEnable](#).
2. The period register value can be set using [PLIB_TMR_Period8BitSet](#), [PLIB_TMR_Period16BitSet](#), or [PLIB_TMR_Period32BitSet](#). Use the appropriate function based on the resolution of the timer.
3. Set the start value of the timer using [PLIB_TMR_Counter8BitSet](#), [PLIB_TMR_Counter16BitSet](#), and [PLIB_TMR_Counter32BitSet](#).
4. Some timer modules can select an external clock as its input source. Use [PLIB_TMR_ClockSourceSelect](#) to select the external clock source.
5. For Asynchronous mode operation, the clock source is not synchronized with the internal clock. Use [PLIB_TMR_ClockSourceExternalSyncDisable](#) to disable the synchronization of the external clock source.
6. The prescaler for the clock can be selected using [PLIB_TMR_PrescaleSelect](#). Pass in the desired prescaler into the function from the enumeration [TMR_PRESCALE](#).
7. Timer count is incremented on the rising timer clock edge. On some microcontrollers, this can be changed using [PLIB_TMR_ClockSourceEdgeSelect](#), with the appropriate parameter for the falling edge (i.e., increment on high-to-low).
8. On some microcontrollers, the timer output can go through a postscaler. The postscaler can be configured using [PLIB_TMR_PostscaleSelect](#).
9. Clear the timer register using [PLIB_TMR_Counter8BitClear](#), [PLIB_TMR_Counter16BitClear](#), or [PLIB_TMR_Counter32BitClear](#).
10. Start the timer once the setup is complete using [PLIB_TMR_Start](#). The timer operation can be stopped using [PLIB_TMR_Stop](#).

Example: Asynchronous Counter Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and disable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_64);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0x7F);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

A timer interrupt is generated if enabled, when the timer count matches the period value if available, or when the counter overflows.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes for your device.

5.6.24.4.3.4 Gated Timer

This section provides information and examples for setting up Gated Timer operation.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes for your device.

Description

Gated Timer operation can be used to allow measurement of an external signal.

Setup

Use the following steps for Gated Timer setup:

1. Some timer modules support multiple resolutions, which can be selected using [PLIB_TMR_Mode8BitEnable](#), [PLIB_TMR_Mode16BitEnable](#), or [PLIB_TMR_Mode32BitEnable](#).
2. The period register value can be set using [PLIB_TMR_Period8BitSet](#), [PLIB_TMR_Period16BitSet](#), or [PLIB_TMR_Period32BitSet](#). Use the appropriate function based on the resolution of the timer.
3. Set the start value of the timer using [PLIB_TMR_Counter8BitSet](#), [PLIB_TMR_Counter16BitSet](#), and [PLIB_TMR_Counter32BitSet](#).
4. In this mode, the input clock source for the timer is the internal clock, and is selected using [PLIB_TMR_ClockSourceSelect](#), depending on which clock is used as the internal clock in the timer module of the device.
5. The prescaler for the clock can be selected using [PLIB_TMR_PrescaleSelect](#). Pass in the desired pre scaler into the function from the enumeration [TMR_PRESCALE](#).
6. Timer count is incremented on the rising timer clock edge. On some microcontrollers, this is can be changed using [PLIB_TMR_ClockSourceEdgeSelect](#), with the appropriate parameter for the falling edge (i.e., increment on high-to-low).
7. On some microcontrollers, the timer output can go through a postscaler. The postscaler can be configured using [PLIB_TMR_PostscaleSelect](#).
8. Clear the timer register using [PLIB_TMR_Counter8BitClear](#), [PLIB_TMR_Counter16BitClear](#), or [PLIB_TMR_Counter32BitClear](#).
9. Gated operation can be enabled using the function [PLIB_TMR_GateEnable](#).
10. On some devices, additional gate functionality can be controlled:
 - Gate Polarity: To allow the timer to count when the gate signal is high, use [PLIB_TMR_GatePolaritySelect](#) with the parameter [TMR_GATE_POLARITY_ACTIVE_HIGH](#). To allow the timer to count when the gate signal is low, use [PLIB_TMR_GatePolaritySelect](#) with the parameter [TMR_GATE_POLARITY_ACTIVE_LOW](#).
 - Gate Toggle mode: To allow the timer to measure the full cycle length of gate signal as opposed to the duration of a single pulse level, use [PLIB_TMR_GateToggleModeEnable](#). To disable this mode, use [PLIB_TMR_GateToggleModeDisable](#).
 - Gate Single Pulse mode: To allow the timer to capture a single pulse gate event, use [PLIB_TMR_GateSinglePulseModeEnable](#) and [PLIB_TMR_GateSinglePulseModeDisable](#). The single pulse acquisition can be started using [PLIB_TMR_GateSinglePulseAcquisitionStart](#), and its status can be read using [PLIB_TMR_GateSinglePulseAcquisitionHasCompleted](#).
 - Additionally, the current state of the gate can be obtained using [PLIB_TMR_GateCurrentStateGet](#)
 - The gate source can be selected using [PLIB_TMR_GateSourceSelect](#), using one of the possible values from [TMR_GATE_SOURCE](#)
11. Start the timer once the setup is complete using [PLIB_TMR_Start](#). The timer operation can be stopped using [PLIB_TMR_Stop](#).

Example 1: Gated Timer Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
```

```

PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and disable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_64);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0x7F00);

// Enable the gate timer
PLIB_TMR_GateEnable(MY_TIMER_INSTANCE);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);

```

Example 2: Gated Timer Setup

```

// Stop the timer
uint8_t pulse_width;
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and disable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_64);
PLIB_TMR_ClockSourceExternalInputClockEnable(MY_TIMER_INSTANCE);
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);
// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0x7F00);

// Enable the gate timer
PLIB_TMR_GateEnable(MY_TIMER_INSTANCE);
// Gate source selection
PLIB_TMR_GateSourceSelect(MY_TIMER_INSTANCE, TMR_GATE_SOURCE_T1_GATE);
// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
// Single pulse acquisition start
PLIB_TMR_GateSinglePulseAcquisitionStart(MY_TIMER_INSTANCE);
// Counting enabled on the rising edge
PLIB_TMR_GatePolaritySelect(MY_TIMER_INSTANCE, TMR_GATE_POLARITY_ACTIVE_HIGH);
// Gate pulse mode enabled
PLIB_TMR_GateSinglePulseModeEnable(MY_TIMER_INSTANCE);

```

A timer interrupt is generated if enabled, on the falling edge of the gate signal. Refer to the specific device data sheet to determine whether this feature is available for your device.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes for your device.

5.6.24.4.3.5 Other Features

This section provides information on additional features.

Description

Each of the following modes can additionally support some or all of the features listed. Please refer to the specific device data sheet or family reference manual for details.

Operation During Sleep Mode

When the Timer module is configured to operate in Asynchronous mode (with an external clock source), it will continue to increment each timer clock (or prescale multiple of clocks) during Sleep mode.

On some devices, Timer operation during Sleep mode can be disabled using [PLIB_TMR_OperateInSleepDisable](#), and can be enabled using [PLIB_TMR_OperateInSleepEnable](#).

Operation During Idle Mode

A Timer module can continue to operate in Idle mode by calling [PLIB_TMR_StopInIdleDisable](#), or can stop operation in Idle mode by calling [PLIB_TMR_StopInIdleEnable](#).

Event Trigger Reset

If the PWM modules are configured to use a Timer module, a special event trigger can be generated in Compare mode. The trigger can reset the timer modules. If supported by the device, the special event trigger reset can be enabled using [PLIB_TMR_TriggerEventResetEnable](#), or disabled using [PLIB_TMR_TriggerEventResetDisable](#).

To be able to use this feature, the Timer module must be configured in one of the synchronous modes.

Timer Mode Control

- [PLIB_TMR_Mode8BitEnable](#) - For PIC18 timers, This interface disables 16-bit mode and enables 8-bit mode
- [PLIB_TMR_Mode16BitEnable](#):
 - For PIC18 timers, This interface disables 8-bit mode and enables 16-bit mode
 - For PIC24/PIC32 timers, this interface enables 16-bit mode and disables 32-bit mode
- [PLIB_TMR_Mode32BitEnable](#) - For PIC24/PIC32 timers, this interface enables 32-bit mode and disables 16-bit mode

Timer Usage Mode

Some timers operate in either Period Match mode or Overflow mode. The mode state can be retrieved from the Timer Peripheral Library using [IsPeriodMatchBased](#). This function returns 'true' Period Match mode and 'false' for Overflow mode.

Timer Prescaler Information

Prescaler divisor information associated with the respective prescaler value selected through [TMR_PRESCALE](#) can be retrieved through [PLIB_TMR_PrescaleDivisorGet](#).

The true prescaler value can be obtained by using [PLIB_TMR_PrescaleGet](#).

Timer Postscaler Information

Postscaler divisor information associated with the respective postscaler value is selected through the enumeration [TMR_POSTSCALE](#), and can be retrieved through [PLIB_TMR_PostscaleDivisorGet](#).

The true postscaler value can be obtained by using [PLIB_TMR_PostscaleGet](#).

Timer Source Edge Information

The TMR source edge information can be retrieved using [PLIB_TMR_ClockSourceEdgeGet](#).

5.6.24.4.4 Configuring the Library

The library is configured for the supported Timer modules when the processor is chosen in the MPLAB IDE.

5.6.24.5 Library Interface

Data Types and Constants

	Name	Description
	<u>TMR_ASSIGNMENT</u>	Data type defining the Timer assignment to modules.
	<u>TMR_CLOCK_SOURCE</u>	Data type defining the different clock sources.
	<u>TMR_CLOCK_SOURCE_EDGE</u>	Data type defining the different source edges.
	<u>TMR_GATE_POLARITY</u>	Data type defining the different gate polarities.
	<u>TMR_GATE_SOURCE</u>	Data type defining the different gate source selections.
	<u>TMR_MODULE_ID</u>	Identifies the supported Timer modules.
	<u>TMR_POSTSCALE</u>	Defines the list of possible postscaler values.
	<u>TMR_PRESCALE</u>	Defines the list of possible prescaler values.

Clock Source Control Functions

	Name	Description
	<u>PLIB_TMR_ClockSourceEdgeGet</u>	Gets the source edge information.
	<u>PLIB_TMR_ClockSourceEdgeSelect</u>	Selects the input clock source edge.
	<u>PLIB_TMR_ClockSourceExternalSyncDisable</u>	Disables the clock synchronization of the external input.
	<u>PLIB_TMR_ClockSourceExternalSyncEnable</u>	Enables the clock synchronization of the external input.
	<u>PLIB_TMR_ClockSourceSelect</u>	Selects the clock source.

Counter Control Functions

	Name	Description
	<u>PLIB_TMR_Counter16BitClear</u>	Clears the 16-bit timer value.
	<u>PLIB_TMR_Counter16BitGet</u>	Gets the 16-bit timer value.
	<u>PLIB_TMR_Counter16BitSet</u>	Sets the 16-bit timer value.
	<u>PLIB_TMR_Counter32BitClear</u>	Clears the 32-bit timer value.
	<u>PLIB_TMR_Counter32BitGet</u>	Gets the 32-bit timer value.
	<u>PLIB_TMR_Counter32BitSet</u>	Sets the 32-bit timer value.
	<u>PLIB_TMR_Counter8BitClear</u>	Clears the 8-bit timer value.
	<u>PLIB_TMR_Counter8BitGet</u>	Gets the 8-bit timer value.
	<u>PLIB_TMR_Counter8BitSet</u>	Sets the 8-bit timer value.
	<u>PLIB_TMR_CounterAsyncWriteDisable</u>	Disables the writes to the counter register until the pending write operation completes.
	<u>PLIB_TMR_CounterAsyncWriteEnable</u>	Enables back-to-back writes with legacy asynchronous timer functionality.
	<u>PLIB_TMR_CounterAsyncWriteInProgress</u>	Returns the status of the counter write in Asynchronous mode.

Feature Existence Functions

	Name	Description
	<u>PLIB_TMR_ExistsClockSource</u>	Identifies whether the ClockSource feature exists on the Timer module.
	<u>PLIB_TMR_ExistsClockSourceEdge</u>	Identifies whether the ClockSourceEdge feature exists on the Timer module.
	<u>PLIB_TMR_ExistsClockSourceSync</u>	Identifies whether the ClockSourceSync feature exists on the Timer module.

	PLIB_TMR_ExistsCounter16Bit	Identifies whether the Counter16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounter32Bit	Identifies whether the Counter32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounter8Bit	Identifies whether the Counter8bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteControl	Identifies whether the CounterAsyncWriteControl feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteInProgress	Identifies whether the CounterAsyncWriteInProgress feature exists on the Timer module.
	PLIB_TMR_ExistsCountMode	Identifies whether the CountMode feature exists on the Timer module.
	PLIB_TMR_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Timer module.
	PLIB_TMR_ExistsGateCurrentState	Identifies whether the GateCurrentState feature exists on the Timer module.
	PLIB_TMR_ExistsGatedTimeAccumulation	Identifies whether the GatedTimeAccumulation feature exists on the Timer module.
	PLIB_TMR_ExistsGatePolarity	Identifies whether the GatePolarity feature exists on the Timer module.
	PLIB_TMR_ExistsGateSinglePulseAcquisition	Identifies whether the GateSinglePulseAcquisition feature exists on the Timer module.
	PLIB_TMR_ExistsGateSinglePulseMode	Identifies whether the GateSinglePulseMode feature exists on the Timer module.
	PLIB_TMR_ExistsGateSource	Identifies whether the GateSource feature exists on the Timer module.
	PLIB_TMR_ExistsGateToggleMode	Identifies whether the GateToggleMode feature exists on the Timer module.
	PLIB_TMR_ExistsMode16Bit	Identifies whether the Mode16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsMode32Bit	Identifies whether the Mode32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsMode8Bit	Identifies whether the Mode8Bit feature exists on the Timer module.
	PLIB_TMR_ExistsOperationInSleep	Identifies whether the OperationInSleep feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod16Bit	Identifies whether the Period16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod32Bit	Identifies whether the Period32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod8Bit	Identifies whether the Period8Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPostscale	Identifies whether the Postscale feature exists on the Timer module.
	PLIB_TMR_ExistsPrescale	Identifies whether the Prescale feature exists on the Timer module.
	PLIB_TMR_ExistsPrescalerAssignment	Identifies whether the PrescalerControl feature exists on the Timer module.
	PLIB_TMR_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the Timer module.
	PLIB_TMR_ExistsSystemClockStatus	Identifies whether the SystemClockStatus feature exists on the Timer module.

	PLIB_TMR_ExistsTimerAssignment	Identifies whether the TimerAssignment feature exists on the Timer module.
	PLIB_TMR_ExistsTimerOperationMode	Identifies whether the TimerOperationMode feature exists on the Timer module.
	PLIB_TMR_ExistsTimerOscillator	Identifies whether the TimerOscillator feature exists on the Timer module.
	PLIB_TMR_ExistsTriggerEventReset	Identifies whether the TriggerEventReset feature exists on the Timer module.

Gate Control Functions

	Name	Description
	PLIB_TMR_GateCurrentStateGet	Returns the current level of the timer gate.
	PLIB_TMR_GateDisable	Enables counting regardless of the corresponding timer gate function.
	PLIB_TMR_GateEnable	Enables counting controlled by the corresponding gate function.
	PLIB_TMR_GatePolaritySelect	Selects the timer gate polarity.
	PLIB_TMR_GateSinglePulseAcquisitionHasCompleted	Returns the gate single pulse acquisition status.
	PLIB_TMR_GateSinglePulseAcquisitionStart	Starts the gate single pulse acquisition.
	PLIB_TMR_GateSinglePulseModeDisable	Disables Single Pulse mode.
	PLIB_TMR_GateSinglePulseModeEnable	Enables Single pulse mode and the controlling corresponding timer gate.
	PLIB_TMR_GateSourceSelect	Selects the timer gate source.
	PLIB_TMR_GateToggleModeDisable	Disables the Gate Toggle mode.
	PLIB_TMR_GateToggleModeEnable	Enables the Gate Toggle mode.

General Setup Functions

	Name	Description
	PLIB_TMR_AssignmentSelect	Assigns the specified Timer(s) to the selected modules.
	PLIB_TMR_ContinuousCountModeEnable	Enables Continuous Count mode
	PLIB_TMR_Mode16BitEnable	Enables the Timer module for 16-bit operation and disables all other modes.
	PLIB_TMR_Mode32BitEnable	Enables 32-bit operation on the Timer module combination.
	PLIB_TMR_Mode8BitEnable	Enables the Timer module in 8-bit operation mode and disables 16-bit mode.
	PLIB_TMR_Start	Starts/enables the indexed timer.
	PLIB_TMR_Stop	Stops/disables the indexed timer.
	PLIB_TMR_TimerOscillatorDisable	Disables the oscillator associated with the Timer module.
	PLIB_TMR_TimerOscillatorEnable	Enables the oscillator associated with the Timer module.
	PLIB_TMR_TriggerEventResetDisable	Disables the special event trigger reset.
	PLIB_TMR_TriggerEventResetEnable	Enables the special event trigger reset.
	PLIB_TMR_SingleShotModeEnable	Enables Single-shot mode.

Miscellaneous Functions

	Name	Description
	PLIB_TMR_IsPeriodMatchBased	Gets the operating mode state of the Timer module based on Period Match or Overflow mode.
	PLIB_TMR_SystemClockFromTimerIsActive	Gets the system clock status.

Period Control Functions

	Name	Description
≡	PLIB_TMR_Period16BitGet	Gets the 16-bit period value.
≡	PLIB_TMR_Period16BitSet	Sets the 16-bit period value.
≡	PLIB_TMR_Period32BitGet	Gets the 32-bit period value.
≡	PLIB_TMR_Period32BitSet	Sets the 32-bit period value.
≡	PLIB_TMR_Period8BitGet	Gets the 8-bit period value.
≡	PLIB_TMR_Period8BitSet	Sets the 8-bit period value.

Post-processing Functions

	Name	Description
≡	PLIB_TMR_PostscaleDivisorGet	Gets the postscaler divisor information.
≡	PLIB_TMR_PostscaleGet	Gets the postscaler divisor information.
≡	PLIB_TMR_PostscaleSelect	Selects the output clock postscaler.

Power Control Functions

	Name	Description
≡	PLIB_TMR_OperateInSleepDisable	Disables Timer module operation when the device is in Sleep mode.
≡	PLIB_TMR_OperateInSleepEnable	Enables Timer module operation when the device is in Sleep mode.
≡	PLIB_TMR_StopInIdleDisable	Continue module operation when the device enters Idle mode.
≡	PLIB_TMR_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.

Preprocessing Functions

	Name	Description
≡	PLIB_TMR_PrescaleDivisorGet	Gets the prescaler divisor information.
≡	PLIB_TMR_PrescaleGet	Gets the prescaler information.
≡	PLIB_TMR_PrescalerDisable	Disables the prescaler assignment to the indexed Timer module.
≡	PLIB_TMR_PrescalerEnable	Enables the prescaler assignment to the indexed Timer module.
≡	PLIB_TMR_PrescaleSelect	Selects the clock prescaler.

Description

This section lists and describes the functions, data types, and constants available in the Timer Peripheral Library.

5.6.24.5.1 General Setup Functions**5.6.24.5.1.1 PLIB_TMR_AssignmentSelect Function****C**

```
void PLIB_TMR_AssignmentSelect(
    TMR_MODULE_ID index,
    TMR_ASSIGNMENT tmrNums
);
```

Description

This function assigns the specified Timer(s) as the clock source for the selected modules.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tmrNums	One of the possible values of TMR_ASSIGNMENT

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsTimerAssignment](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_AssignmentSelect(MY_TMR_INSTANCE, TMR_ASSIGNMENT_T1_TO_CCP_ALL);
```

5.6.24.5.1.2 PLIB_TMR_ContinuousCountModeEnable Function**C**

```
void PLIB_TMR_ContinuousCountModeEnable(  
    TMR_MODULE_ID index  
);
```

Description

This function enables Continuous Count mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCountMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_ContinuousCountModeEnable(MY_TMR_INSTANCE);
```

5.6.24.5.1.3 PLIB_TMR_Mode16BitEnable Function**C**

```
void PLIB_TMR_Mode16BitEnable(  
    TMR_MODULE_ID index  
);
```

Description

This function enables the Timer module for 16-bit operation and disables all other modes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Calling this function disables the operation of the Timer module 8-bit mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsMode16Bit](#) in your application to determine whether this feature is available.

For PIC18 Timers, this function disables 8-bit mode and enables 16-bit mode. For PIC24/PIC32 TMRs, this function enables 16-bit mode and disables 32-bit mode.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Mode16BitEnable(MY_TMR_INSTANCE);
```

5.6.24.5.1.4 PLIB_TMR_Mode32BitEnable Function**C**

```
void PLIB_TMR_Mode32BitEnable(
    TMR_MODULE_ID index
);
```

Description

This function enables the Timer module and the index +1 Timer module to act as one 32-bit timer module and disables all other modes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsMode32Bit](#) in your application to determine whether this feature is available.

For PIC24/PIC32 Timers, this function enables 32-bit mode and disables 16-bit mode.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Mode32BitEnable(MY_TMR_INSTANCE);
```

5.6.24.5.1.5 PLIB_TMR_Mode8BitEnable Function

C

```
void PLIB_TMR_Mode8BitEnable(  
    TMR_MODULE_ID index  
);
```

Description

This function enables the Timer module in 8-bit operation mode and disables 16-bit mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Calling this function disables the operation of Timer module in 16-bit mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsMode8Bit](#) in your application to determine whether this feature is available.

For PIC18 Timers, this function disables 16-bit mode and enables 8-bit mode.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_Mode8BitEnable(MY_TMR_INSTANCE);
```

5.6.24.5.1.6 PLIB_TMR_Start Function

C

```
void PLIB_TMR_Start(  
    TMR_MODULE_ID index  
);
```

Description

This function starts/enables the indexed timer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_TMR_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Start(MY_TMR_INSTANCE);
```

5.6.24.5.1.7 PLIB_TMR_Stop Function

C

```
void PLIB_TMR_Stop(
    TMR_MODULE_ID index
);
```

Description

This function stops/disables the indexed timer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Stop(MY_TMR_INSTANCE);
```

5.6.24.5.1.8 PLIB_TMR_TimerOscillatorDisable Function

C

```
void PLIB_TMR_TimerOscillatorDisable(
    TMR_MODULE_ID index
);
```

Description

This function disables the oscillator associated with the Timer module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsTimerOscillator](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_TimerOscillatorDisable(MY_TMR_INSTANCE);
```

5.6.24.5.1.9 PLIB_TMR_TimerOscillatorEnable Function**C**

```
void PLIB_TMR_TimerOscillatorEnable(
    TMR_MODULE_ID index
);
```

Description

This function enables the oscillator associated with the Timer module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsTimerOscillator](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_TimerOscillatorEnable(MY_TMR_INSTANCE);
```

5.6.24.5.1.10 PLIB_TMR_TriggerEventResetDisable Function**C**

```
void PLIB_TMR_TriggerEventResetDisable(
    TMR_MODULE_ID index
);
```

Description

This function disables the special event trigger reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsTriggerEventReset](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_TriggerEventResetDisable(MY_TMR_INSTANCE);
```

5.6.24.5.1.11 PLIB_TMR_TriggerEventResetEnable Function**C**

```
void PLIB_TMR_TriggerEventResetEnable(  
    TMR_MODULE_ID index  
)
```

Description

This function enables the special event trigger reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsTriggerEventReset](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_TriggerEventResetEnable(MY_TMR_INSTANCE);
```

5.6.24.5.1.12 PLIB_TMR_SingleShotModeEnable Function**C**

```
void PLIB_TMR_SingleShotModeEnable(  
    TMR_MODULE_ID index  
)
```

Description

This function enables Single-shot mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCountMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_SingleShotModeEnable(MY_TMR_INSTANCE);
```

5.6.24.5.2 Power Control Functions

5.6.24.5.2.1 PLIB_TMR_OperateInSleepDisable Function

C

```
void PLIB_TMR_OperateInSleepDisable(  
    TMR_MODULE_ID index  
);
```

Description

This function disables Timer module operation when the device is in Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsOperationInSleep](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_OperateInSleepDisable(MY_TMR_INSTANCE);
```

5.6.24.5.2.2 PLIB_TMR_OperateInSleepEnable Function

C

```
void PLIB_TMR_OperateInSleepEnable(  
    TMR_MODULE_ID index  
);
```


Description

This function enables Timer module operation when the device is in Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsOperationInSleep](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_OperateInSleepEnable(MY_TMR_INSTANCE);
```

5.6.24.5.2.3 PLIB_TMR_StopInIdleDisable Function**C**

```
void PLIB_TMR_StopInIdleDisable(
    TMR_MODULE_ID index
);
```

Description

This function continues module operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsStopInIdleControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_StopInIdleDisable(MY_TMR_INSTANCE);
```

5.6.24.5.2.4 PLIB_TMR_StopInIdleEnable Function**C**

```
void PLIB_TMR_StopInIdleEnable(
```

```
TMR_MODULE_ID index  
);
```

Description

This function discontinues module operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsStopInIdleControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the // application developer.  
PLIB_TMR_StopInIdleEnable(MY_TMR_INSTANCE);
```

5.6.24.5.3 Clock Source Control Functions

5.6.24.5.3.1 PLIB_TMR_ClockSourceEdgeGet Function

C

```
TMR_CLOCK_SOURCE_EDGE PLIB_TMR_ClockSourceEdgeGet(  
    TMR_MODULE_ID index  
);
```

Description

This function gets the input clock source edge information.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

One of the possible values of [TMR_CLOCK_SOURCE_EDGE](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsClockSourceEdge](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.
```

```
TMR_CLOCK_SOURCE_EDGE source;  
source = PLIB_TMR_ClockSourceEdgeGet(MY_TMR_INSTANCE);
```

5.6.24.5.3.2 PLIB_TMR_ClockSourceEdgeSelect Function

C

```
void PLIB_TMR_ClockSourceEdgeSelect(  
    TMR_MODULE_ID index,  
    TMR_CLOCK_SOURCE_EDGE source  
);
```

Description

This function selects the input clock source edge, which could either be high-to-low (falling edge) or low-to-high (rising edge).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values of TMR_CLOCK_SOURCE_EDGE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsClockSourceEdge](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_ClockSourceEdgeSelect(MY_TMR_INSTANCE,  
    TMR_CLOCK_SOURCE_EDGE_INCREMENT_ON_LOW_TO_HIGH);
```

5.6.24.5.3.3 PLIB_TMR_ClockSourceExternalSyncDisable Function

C

```
void PLIB_TMR_ClockSourceExternalSyncDisable(  
    TMR_MODULE_ID index  
);
```

Description

This function disables the clock synchronization of the external input.

Preconditions

The timer module must be configured to use the external clock using the function [PLIB_TMR_ClockSourceSelect](#) with the clock source as `TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN`. If the function [PLIB_TMR_ClockSourceSelect](#) configures the clock with some other enumeration, this function will have no effect.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsClockSourceSync](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TMR_INSTANCE);
```

5.6.24.5.3.4 PLIB_TMR_ClockSourceExternalSyncEnable Function**C**

```
void PLIB_TMR_ClockSourceExternalSyncEnable(  
    TMR_MODULE_ID index  
);
```

Description

This function enables the clock synchronization of the external input.

Preconditions

The timer module must be configured to use the external clock using the function [PLIB_TMR_ClockSourceSelect](#) with the clock source as TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN. If the function [PLIB_TMR_ClockSourceSelect](#) configures the clock with some other enumeration, this function will have no effect.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsClockSourceSync](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_ClockSourceExternalSyncEnable(MY_TMR_INSTANCE);
```

5.6.24.5.3.5 PLIB_TMR_ClockSourceSelect Function**C**

```
void PLIB_TMR_ClockSourceSelect(  
    TMR_MODULE_ID index,  
    TMR_CLOCK_SOURCE source  
);
```

Description

This function selects the clock source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values of TMR_CLOCK_SOURCE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGateSource](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_ClockSourceSelect(MY_TMR_INSTANCE, TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
```

5.6.24.5.4 Gate Control Functions

5.6.24.5.4.1 PLIB_TMR_GateCurrentStateGet Function

C

```
bool PLIB_TMR_GateCurrentStateGet(
    TMR_MODULE_ID index
);
```

Description

This function returns the current level of the timer gate. The gate value is set for the entire duration of the clock count whether it is a single pulse or a full clock cycle.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Acquisition has started
- false - Acquisition has completed or has not been started

Remarks

This feature is not affected by [PLIB_TMR_GateEnable](#) or [PLIB_TMR_GateDisable](#).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGateCurrentState](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
bool gatestate = PLIB_TMR_GateCurrentStateGet(MY_TMR_INSTANCE);
```

5.6.24.5.4.2 PLIB_TMR_GateDisable Function

C

```
void PLIB_TMR_GateDisable(  
    TMR_MODULE_ID index  
);
```

Description

This function enables counting regardless of the gate function.

Preconditions

The timer must be enabled for this function to take effect.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGatedTimeAccumulation](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_GateDisable(MY_TMR_INSTANCE);
```

5.6.24.5.4.3 PLIB_TMR_GateEnable Function

C

```
void PLIB_TMR_GateEnable(  
    TMR_MODULE_ID index  
);
```

Description

This function enables counting controlled by the gate function.

Preconditions

The timer must be enabled, for this function to take effect.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGatedTimeAccumulation](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
```

```
// application developer.  
PLIB_TMR_GateEnable(MY_TMR_INSTANCE);
```

5.6.24.5.4.4 PLIB_TMR_GatePolaritySelect Function

C

```
void PLIB_TMR_GatePolaritySelect(  
    TMR_MODULE_ID index,  
    TMR_GATE_POLARITY polarity  
);
```

Description

This function selects the timer gate polarity.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
polarity	One of the possible values of TMR_GATE_POLARITY

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGatePolarity](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_GatePolaritySelect(MY_TMR_INSTANCE, TMR_GATE_POLARITY_ACTIVE_HIGH);
```

5.6.24.5.4.5 PLIB_TMR_GateSinglePulseAcquisitionHasCompleted Function

C

```
bool PLIB_TMR_GateSinglePulseAcquisitionHasCompleted(  
    TMR_MODULE_ID index  
);
```

Description

This function returns the gate single pulse acquisition status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Acquisition is waiting for an edge
- false - Acquisition has completed or has not been started

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGateSinglePulseAcquisition](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
bool isComplete;  
isComplete = PLIB_TMR_GateSinglePulseAcquisitionHasCompleted(MY_TMR_INSTANCE);
```

5.6.24.5.4.6 PLIB_TMR_GateSinglePulseAcquisitionStart Function**C**

```
void PLIB_TMR_GateSinglePulseAcquisitionStart(  
    TMR_MODULE_ID index  
);
```

Description

This function starts the gate single pulse acquisition feature.

Preconditions

[PLIB_TMR_GateSinglePulseModeEnable](#) should be called before this function.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGateSinglePulseAcquisition](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_GateSinglePulseAcquisitionStart(MY_TMR_INSTANCE);
```

5.6.24.5.4.7 PLIB_TMR_GateSinglePulseModeDisable Function**C**

```
void PLIB_TMR_GateSinglePulseModeDisable(  
    TMR_MODULE_ID index  
);
```

Description

This function disables Single Pulse mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGateSinglePulseMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_GateSinglePulseModeDisable(MY_TMR_INSTANCE);
```

5.6.24.5.4.8 PLIB_TMR_GateSinglePulseModeEnable Function**C**

```
void PLIB_TMR_GateSinglePulseModeEnable(  
    TMR_MODULE_ID index  
);
```

Description

This function enables Single pulse mode and the controlling corresponding timer gate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGateSinglePulseMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_GateSinglePulseModeEnable(MY_TMR_INSTANCE);
```

5.6.24.5.4.9 PLIB_TMR_GateSourceSelect Function**C**

```
void PLIB_TMR_GateSourceSelect(  
    TMR_MODULE_ID index,  
    TMR_GATE_SOURCE source  
);
```

Description

This function selects the timer gate source.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values of TMR_GATE_SOURCE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGateSource](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_GateSourceSelect(MY_TMR_INSTANCE, TMR_GATE_SOURCE_CMP1_SYNC_OUTPUT);
```

5.6.24.5.4.10 PLIB_TMR_GateToggleModeDisable Function**C**

```
void PLIB_TMR_GateToggleModeDisable(  
    TMR_MODULE_ID index  
);
```

Description

This function disables Gate Toggle mode and the toggle flip flop is cleared.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGateToggleMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_GateToggleModeDisable(MY_TMR_INSTANCE);
```

5.6.24.5.4.11 PLIB_TMR_GateToggleModeEnable Function**C**

```
void PLIB_TMR_GateToggleModeEnable(  
    TMR_MODULE_ID index  
);
```

Description

This function enables Gate Toggle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsGateToggleMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_GateToggleModeEnable(MY_TMR_INSTANCE);
```

5.6.24.5.5 Preprocessing Functions

5.6.24.5.5.1 PLIB_TMR_PrescaleDivisorGet Function

C

```
uint16_t PLIB_TMR_PrescaleDivisorGet(
    TMR_MODULE_ID index,
    TMR_PRESCALE prescale
);
```

Description

This function gets the prescaler divisor information.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
prescale	One of the possible values of TMR_PRESCALE

Returns

prescale divisor value

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPrescale](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint16_t div = PLIB_TMR_PrescaleDivisorGet (MY_TMR_INSTANCE, TMR_PRESCALE_VALUE_1);
```

5.6.24.5.5.2 PLIB_TMR_PrescaleGet Function

C

```
TMR_PRESCALE PLIB_TMR_PrescaleGet(  
    TMR_MODULE_ID index  
);
```

Description

This function gets the prescaler information.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

One of the possible values of [TMR_PRESCALE](#)

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPrescale](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
TMR_PRESCALE prescale;  
prescale = PLIB_TMR_PrescaleGet (MY_TMR_INSTANCE);
```

5.6.24.5.5.3 PLIB_TMR_PrescalerDisable Function

C

```
void PLIB_TMR_PrescalerDisable(  
    TMR_MODULE_ID index  
);
```

Description

This function disables the prescaler assignment to the indexed Timer module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPrescalerAssignment](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_PrescalerDisable(MY_TMR_INSTANCE);
```

5.6.24.5.5.4 PLIB_TMR_PrescalerEnable Function**C**

```
void PLIB_TMR_PrescalerEnable(
    TMR_MODULE_ID index
);
```

Description

This function enables the prescaler assignment to the indexed Timer module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPrescalerAssignment](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_PrescalerEnable(MY_TMR_INSTANCE);
```

5.6.24.5.5.5 PLIB_TMR_PrescaleSelect Function**C**

```
void PLIB_TMR_PrescaleSelect(
    TMR_MODULE_ID index,
    TMR_PRESCALE prescale
);
```

Description

This function selects the clock prescaler.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
prescale	One of the possible values of TMR_PRESCALE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPrescale](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_PrescaleSelect(MY_TMR_INSTANCE, TMR_PRESCALE_VALUE_8);
```

5.6.24.5.6 Period Control Functions

5.6.24.5.6.1 PLIB_TMR_Period16BitGet Function

C

```
uint16_t PLIB_TMR_Period16BitGet(  
    TMR_MODULE_ID index  
);
```

Description

This function gets the 16-bit period value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

period - 16-bit period value

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPeriod16Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
uint16_t period = PLIB_TMR_Period16BitGet(MY_TMR_INSTANCE);
```

5.6.24.5.6.2 PLIB_TMR_Period16BitSet Function

C

```
void PLIB_TMR_Period16BitSet(  
    TMR_MODULE_ID index,  
    uint16_t period  
);
```

Description

This function sets the 16-bit period value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
period	16-bit period register value

Returns

None.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPeriod16Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
// where, MY_PERIOD_VALUE is the 16-bit value which needs to be stored in the
// period register.
PLIB_TMR_Period16BitSet(MY_TMR_INSTANCE, MY_PERIOD_VALUE);
```

5.6.24.5.6.3 PLIB_TMR_Period32BitGet Function**C**

```
uint32_t PLIB_TMR_Period32BitGet(
    TMR_MODULE_ID index
);
```

Description

This function gets the 32-bit period value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

period - 32-bit period value

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPeriod32Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
uint32_t period = PLIB_TMR_Period32BitGet(MY_TMR_INSTANCE);
```

5.6.24.5.6.4 PLIB_TMR_Period32BitSet Function**C**

```
void PLIB_TMR_Period32BitSet(  
    TMR_MODULE_ID index,  
    uint32_t period  
);
```

Description

This function sets the 32-bit period value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
period	32-bit period register value

Returns

None.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPeriod32Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
// where, MY_PERIOD_VALUE is the 32-bit value which needs to be stored in the  
// period register.  
PLIB_TMR_Period32BitSet(MY_TMR_INSTANCE, MY_PERIOD_VALUE);
```

5.6.24.5.6.5 PLIB_TMR_Period8BitGet Function**C**

```
uint8_t PLIB_TMR_Period8BitGet(  
    TMR_MODULE_ID index  
);
```

Description

This function gets the 8-bit period value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

period - 8-bit period value

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPeriod8Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint8_t period = PLIB_TMR_Period8BitGet(MY_TMR_INSTANCE);
```

5.6.24.5.6.6 PLIB_TMR_Period8BitSet Function**C**

```
void PLIB_TMR_Period8BitSet(
    TMR_MODULE_ID index,
    uint8_t period
);
```

Description

This function sets the 8-bit period value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
period	8-bit period value

Returns

None.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPeriod8Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
// where, MY_PERIOD_VALUE is the 8-bit value which needs to be stored in the
// period register.
```

```
PLIB_TMR_Period8BitSet(MY_TMR_INSTANCE, MY_PERIOD_VALUE);
```

5.6.24.5.7 Counter Control Functions

5.6.24.5.7.1 PLIB_TMR_Counter16BitClear Function

C

```
void PLIB_TMR_Counter16BitClear(  
    TMR_MODULE_ID index  
);
```

Description

This function clears the 16-bit timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounter16Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_Counter16BitClear(MY_TMR_INSTANCE);
```

5.6.24.5.7.2 PLIB_TMR_Counter16BitGet Function

C

```
uint16_t PLIB_TMR_Counter16BitGet(  
    TMR_MODULE_ID index  
);
```

Description

This function gets the 16-bit timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

16-bit timer value.

Remarks

PLIB_TMR_Counter16BitGet does not prevent the timer from incrementing during the same instruction cycle.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounter16Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
uint16_t timerValue = PLIB_TMR_Counter16BitGet(MY_TMR_INSTANCE);
```

5.6.24.5.7.3 PLIB_TMR_Counter16BitSet Function**C**

```
void PLIB_TMR_Counter16BitSet(  
    TMR_MODULE_ID index,  
    uint16_t value  
);
```

Description

This function sets the 16-bit timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	16-bit value to be set

Returns

None.

Remarks

When the timer register is written to, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounter16Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_Counter16BitSet(MY_TMR_INSTANCE, 0x100);
```

5.6.24.5.7.4 PLIB_TMR_Counter32BitClear Function**C**

```
void PLIB_TMR_Counter32BitClear(  
    TMR_MODULE_ID index  
);
```

Description

This function clears the 32-bit timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounter32Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter32BitClear(MY_TMR_INSTANCE);
```

5.6.24.5.7.5 PLIB_TMR_Counter32BitGet Function**C**

```
uint32_t PLIB_TMR_Counter32BitGet(
    TMR_MODULE_ID index
);
```

Description

This function gets the 32-bit timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

32 bit timer value.

Remarks

PLIB_TMR_Counter32BitGet does not prevent the timer from incrementing during the same instruction cycle.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounter32Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint32_t timerValue = PLIB_TMR_Counter32BitGet(MY_TMR_INSTANCE);
```

5.6.24.5.7.6 PLIB_TMR_Counter32BitSet Function

C

```
void PLIB_TMR_Counter32BitSet(  
    TMR_MODULE_ID index,  
    uint32_t value  
);
```

Description

This function sets the 32-bit timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	32-bit value to be set

Returns

None.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounter32Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_Counter32BitSet(MY_TMR_INSTANCE, 0x1000000);
```

5.6.24.5.7.7 PLIB_TMR_Counter8BitClear Function

C

```
void PLIB_TMR_Counter8BitClear(  
    TMR_MODULE_ID index  
);
```

Description

This function clears the 8-bit timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounter8Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter8BitClear(MY_TMR_INSTANCE);
```

5.6.24.5.7.8 PLIB_TMR_Counter8BitGet Function**C**

```
uint8_t PLIB_TMR_Counter8BitGet(
    TMR_MODULE_ID index
);
```

Description

This function gets the 8-bit timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

8-bit timer value.

Remarks

PLIB_TMR_Counter8BitGet does not prevent the timer from incrementing during the same instruction cycle.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounter8Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint8_t timerValue = PLIB_TMR_Counter8BitGet(MY_TMR_INSTANCE);
```

5.6.24.5.7.9 PLIB_TMR_Counter8BitSet Function**C**

```
void PLIB_TMR_Counter8BitSet(
    TMR_MODULE_ID index,
    uint8_t value
);
```

Description

This function sets the 8-bit timer value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	8-bit value to be set

Returns

None.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounter8Bit](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter8BitSet(MY_TMR_INSTANCE, 0x10);
```

5.6.24.5.7.10 PLIB_TMR_CounterAsyncWriteDisable Function**C**

```
void PLIB_TMR_CounterAsyncWriteDisable(
    TMR_MODULE_ID index
);
```

Description

This function disables the writes to the counter register until the pending write operation completes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounterAsyncWriteControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_CounterAsyncWriteDisable(MY_TMR_INSTANCE);
```

5.6.24.5.7.11 PLIB_TMR_CounterAsyncWriteEnable Function**C**

```
void PLIB_TMR_CounterAsyncWriteEnable(
    TMR_MODULE_ID index
);
```

Description

This function enables back-to-back writes with legacy asynchronous timer functionality .

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounterAsyncWriteControl](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_CounterAsyncWriteEnable(MY_TMR_INSTANCE);
```

5.6.24.5.7.12 PLIB_TMR_CounterAsyncWriteInProgress Function**C**

```
bool PLIB_TMR_CounterAsyncWriteInProgress(  
    TMR_MODULE_ID index  
);
```

Description

This function returns the status of the counter write in Asynchronous mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Write is in progress
- false - Write is complete

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsCounterAsyncWriteInProgress](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
bool inProgress = PLIB_TMR_CounterAsyncWriteInProgress(MY_TMR_INSTANCE);
```

5.6.24.5.8 Post-processing Functions

5.6.24.5.8.1 PLIB_TMR_PostscaleDivisorGet Function

C

```
uint16_t PLIB_TMR_PostscaleDivisorGet(  
    TMR_MODULE_ID index,  
    TMR_POSTSCALE postscale  
);
```

Description

This function gets the postscaler divisor information.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
postscale	One of the possible values of TMR_POSTSCALE

Returns

Postscaler divisor value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPostscale](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
uint16_t div = PLIB_TMR_PostscaleDivisorGet (MY_TMR_INSTANCE, TMR_POSTSCALE_1_TO_16);
```

5.6.24.5.8.2 PLIB_TMR_PostscaleGet Function

C

```
TMR_POSTSCALE PLIB_TMR_PostscaleGet(  
    TMR_MODULE_ID index  
);
```

Description

This function gets the postscaler divisor information.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

One of the possible values of [TMR_POSTSCALE](#).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPostscale](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
TMR_POSTSCALE postscale;  
postscale = PLIB_TMR_PostscaleGet (MY_TMR_INSTANCE);
```

5.6.24.5.8.3 PLIB_TMR_PostscaleSelect Function**C**

```
void PLIB_TMR_PostscaleSelect(  
    TMR_MODULE_ID index,  
    TMR_POSTSCALE postscale  
);
```

Description

This function selects the output clock postscaler.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
postscale	One of the possible values of TMR_POSTSCALE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsPostscale](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the  
// application developer.  
PLIB_TMR_PostscaleSelect(MY_TMR_INSTANCE, TMR_POSTSCALE_1_TO_1);
```

5.6.24.5.9 Miscellaneous Functions**5.6.24.5.9.1 PLIB_TMR_IsPeriodMatchBased Function****C**

```
bool PLIB_TMR_IsPeriodMatchBased(  
    TMR_MODULE_ID index  
);
```

Description

This function gets the operating mode state of the Timer module based on Period Match or Overflow mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Operation in Period Match mode
- false - Operation in Overflow mode

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsTimerOperationMode](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
bool status = PLIB_TMR_IsPeriodMatchBased(MY_TMR_INSTANCE);
```

5.6.24.5.9.2 PLIB_TMR_SystemClockFromTimerIsActive Function**C**

```
bool PLIB_TMR_SystemClockFromTimerIsActive(
    TMR_MODULE_ID index
);
```

Description

This function gets the system clock derivation status. When the function returns true, the system and peripheral clocks are sourced from the timer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - System/Device Clock is derived from the timer
- false - System/Device Clock is from another source

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_TMR_ExistsSystemClockStatus](#) in your application to determine whether this feature is available.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint8_t status = PLIB_TMR_SystemClockFromTimerIsActive(MY_TMR_INSTANCE);
```

5.6.24.5.10 Data Types and Constants

5.6.24.5.10.1 TMR_ASSIGNMENT Enumeration

C

```
typedef enum {
    TMR_ASSIGNMENT_T1_TO_CCP_ALL,
    TMR_ASSIGNMENT_T3_TO_CCP2_T1_TO_CCP1,
    TMR_ASSIGNMENT_T3_TO_CCP_ALL
} TMR_ASSIGNMENT;
```

Description

TMR assignment to modules definition

This data type defines the Timer assignment to modules.

Members

Members	Description
TMR_ASSIGNMENT_T1_TO_CCP_ALL	Timer1 is the capture/compare clock source for the CCP modules
TMR_ASSIGNMENT_T3_TO_CCP2_T1_TO_CCP1	Timer3 is the capture/compare clock source for CCP2 & Timer1 is the capture/compare clock source for CCP1
TMR_ASSIGNMENT_T3_TO_CCP_ALL	Timer3 is the capture/compare clock source for the CCP modules

Remarks

None.

5.6.24.5.10.2 TMR_CLOCK_SOURCE Enumeration

C

```
typedef enum {
    TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK,
    TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN
} TMR_CLOCK_SOURCE;
```

Description

Clock sources selection enumeration

This data type defines the different clock sources.

Members

Members	Description
TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK	Clock source is the internal clock = FOSC/n (where n, is processor specific)
TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN	Clock source is the external input pin

Remarks

None.

5.6.24.5.10.3 TMR_CLOCK_SOURCE_EDGE Enumeration

C

```
typedef enum {
    TMR_CLOCK_SOURCE_EDGE_INCREMENT_ON_LOW_TO_HIGH,
    TMR_CLOCK_SOURCE_EDGE_INCREMENT_ON_HIGH_TO_LOW
} TMR_CLOCK_SOURCE_EDGE;
```

Description

Source edge selection enumeration

This data type defines the different source edges.

Members

Members	Description
TMR_CLOCK_SOURCE_EDGE_INCREMENT_ON_LOW_TO_HIGH	Source edge - increment on low to high edge
TMR_CLOCK_SOURCE_EDGE_INCREMENT_ON_HIGH_TO_LOW	Source edge - increment on high to low edge

Remarks

None.

5.6.24.5.10.4 TMR_GATE_POLARITY Enumeration

C

```
typedef enum {
    TMR_GATE_POLARITY_ACTIVE_HIGH,
    TMR_GATE_POLARITY_ACTIVE_LOW
} TMR_GATE_POLARITY;
```

Description

Gate polarity selection enumeration

This data type defines the different gate polarities.

Members

Members	Description
TMR_GATE_POLARITY_ACTIVE_HIGH	Gate Polarity - Active High
TMR_GATE_POLARITY_ACTIVE_LOW	Gate Polarity - Active Low

Remarks

None.

5.6.24.5.10.5 TMR_GATE_SOURCE Enumeration

C

```
typedef enum {
    TMR_GATE_SOURCE_T1_GATE,
    TMR_GATE_SOURCE_T0_OVERFLOW,
    TMR_GATE_SOURCE_CMP1_SYNC_OUTPUT,
    TMR_GATE_SOURCE_CMP2_SYNC_OUTPUT
} TMR_GATE_SOURCE;
```

Description

Gate Source selection enumeration

This data type defines the different gate source selections.

Members

Members	Description
TMR_GATE_SOURCE_T1_GATE	Timer1 Gate pin
TMR_GATE_SOURCE_T0_OVERFLOW	Timer0 overflow output
TMR_GATE_SOURCE_CMP1_SYNC_OUTPUT	Comparator 1 optionally synchronized output
TMR_GATE_SOURCE_CMP2_SYNC_OUTPUT	Comparator 2 optionally synchronized output

Remarks

None.

5.6.24.5.10.6 TMR_MODULE_ID Enumeration

C

```
typedef enum {
    TMR_ID_0,
    TMR_ID_1,
    TMR_ID_2,
    TMR_ID_3,
    TMR_ID_4,
    TMR_ID_5,
    TMR_ID_6,
    TMR_ID_7,
    TMR_ID_8,
    TMR_ID_9,
    TMR_NUMBER_OF_MODULES
} TMR_MODULE_ID;
```

Description

TMR Module ID

This enumeration identifies the available Timer modules.

Members

Members	Description
TMR_ID_0	Timer 0
TMR_ID_1	Timer 1
TMR_ID_2	Timer 2
TMR_ID_3	Timer 3
TMR_ID_4	Timer 4
TMR_ID_5	Timer 5
TMR_ID_6	Timer 6
TMR_ID_7	Timer 7
TMR_ID_8	Timer 8
TMR_ID_9	Timer 9
TMR_NUMBER_OF_MODULES	Max number of timers

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

5.6.24.5.10.7 TMR_POSTSCALE Enumeration

C

```
typedef enum {
    TMR_POSTSCALE_1_TO_1,
    TMR_POSTSCALE_1_TO_2,
    TMR_POSTSCALE_1_TO_3,
    TMR_POSTSCALE_1_TO_4,
```

```

TMR_POSTSCALE_1_TO_5,
TMR_POSTSCALE_1_TO_6,
TMR_POSTSCALE_1_TO_7,
TMR_POSTSCALE_1_TO_8,
TMR_POSTSCALE_1_TO_9,
TMR_POSTSCALE_1_TO_10,
TMR_POSTSCALE_1_TO_11,
TMR_POSTSCALE_1_TO_12,
TMR_POSTSCALE_1_TO_13,
TMR_POSTSCALE_1_TO_14,
TMR_POSTSCALE_1_TO_15,
TMR_POSTSCALE_1_TO_16
} TMR_POSTSCALE;

```

Description

Postscaler Values

This macro defines the list of possible postscaler values.

Members

Members	Description
TMR_POSTSCALE_1_TO_1	Timer Postscale Value 1:1
TMR_POSTSCALE_1_TO_2	Timer Postscale Value 1:2
TMR_POSTSCALE_1_TO_3	Timer Postscale Value 1:3
TMR_POSTSCALE_1_TO_4	Timer Postscale Value 1:4
TMR_POSTSCALE_1_TO_5	Timer Postscale Value 1:5
TMR_POSTSCALE_1_TO_6	Timer Postscale Value 1:6
TMR_POSTSCALE_1_TO_7	Timer Postscale Value 1:7
TMR_POSTSCALE_1_TO_8	Timer Postscale Value 1:8
TMR_POSTSCALE_1_TO_9	Timer Postscale Value 1:9
TMR_POSTSCALE_1_TO_10	Timer Postscale Value 1:10
TMR_POSTSCALE_1_TO_11	Timer Postscale Value 1:11
TMR_POSTSCALE_1_TO_12	Timer Postscale Value 1:12
TMR_POSTSCALE_1_TO_13	Timer Postscale Value 1:13
TMR_POSTSCALE_1_TO_14	Timer Postscale Value 1:14
TMR_POSTSCALE_1_TO_15	Timer Postscale Value 1:15
TMR_POSTSCALE_1_TO_16	Timer0 Prescaler Value 1:16

5.6.24.5.10.8 TMR_PRESCALE Enumeration

C

```

typedef enum {
    TMR_PRESCALE_1_TO_1,
    TMR_PRESCALE_1_TO_8,
    TMR_PRESCALE_1_TO_64,
    TMR_PRESCALE_1_TO_256,
    TMR_PRESCALE_VALUE_1,
    TMR_PRESCALE_VALUE_2,
    TMR_PRESCALE_VALUE_4,
    TMR_PRESCALE_VALUE_8,
    TMR_PRESCALE_VALUE_16,
    TMR_PRESCALE_VALUE_32,
    TMR_PRESCALE_VALUE_64,
    TMR_PRESCALE_VALUE_256
} TMR_PRESCALE;

```

Description

Prescaler Values

This macro defines the list of possible prescaler values.

Members

Members	Description
TMR_PRESCALE_1_TO_1	Timer Prescaler Value 1:1
TMR_PRESCALE_1_TO_8	Timer Prescaler Value 1:8
TMR_PRESCALE_1_TO_64	Timer Prescaler Value 1:64
TMR_PRESCALE_1_TO_256	Timer Prescaler Value 1:256
TMR_PRESCALE_VALUE_1	Timer Prescaler Value 1:1
TMR_PRESCALE_VALUE_2	Timer Prescaler Value 1:2
TMR_PRESCALE_VALUE_4	Timer Prescaler Value 1:4
TMR_PRESCALE_VALUE_8	Timer Prescaler Value 1:8
TMR_PRESCALE_VALUE_16	Timer Prescaler Value 1:16
TMR_PRESCALE_VALUE_32	Timer Prescaler Value 1:32
TMR_PRESCALE_VALUE_64	Timer Prescaler Value 1:64
TMR_PRESCALE_VALUE_256	Timer Prescaler Value 1:256

5.6.24.5.11 Feature Existence Functions

5.6.24.5.11.1 PLIB_TMR_ExistsClockSource Function

C

```
bool PLIB_TMR_ExistsClockSource(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the ClockSource feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_ClockSourceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ClockSource feature is supported on the device
- false - The ClockSource feature is not supported on the device

Remarks

None.

5.6.24.5.11.2 PLIB_TMR_ExistsClockSourceEdge Function

C

```
bool PLIB_TMR_ExistsClockSourceEdge(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the ClockSourceEdge feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_ClockSourceEdgeSelect](#)
- [PLIB_TMR_ClockSourceEdgeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ClockSourceEdge feature is supported on the device
- false - The ClockSourceEdge feature is not supported on the device

Remarks

None.

5.6.24.5.11.3 PLIB_TMR_ExistsClockSourceSync Function

C

```
bool PLIB_TMR_ExistsClockSourceSync(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the ClockSourceSync feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_ClockSourceExternalSyncEnable](#)
- [PLIB_TMR_ClockSourceExternalSyncDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ClockSourceSync feature is supported on the device
- false - The ClockSourceSync feature is not supported on the device

Remarks

None.

5.6.24.5.11.4 PLIB_TMR_ExistsCounter16Bit Function**C**

```
bool PLIB_TMR_ExistsCounter16Bit(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Counter16Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_Counter16BitSet](#)
- [PLIB_TMR_Counter16BitGet](#)
- [PLIB_TMR_Counter16BitClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Counter16Bit feature is supported on the device
- false - The Counter16Bit feature is not supported on the device

Remarks

None.

5.6.24.5.11.5 PLIB_TMR_ExistsCounter32Bit Function**C**

```
bool PLIB_TMR_ExistsCounter32Bit(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Counter32Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_Counter32BitSet](#)
- [PLIB_TMR_Counter32BitGet](#)
- [PLIB_TMR_Counter32BitClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Counter32Bit feature is supported on the device
- false - The Counter32Bit feature is not supported on the device

Remarks

None.

5.6.24.5.11.6 PLIB_TMR_ExistsCounter8Bit Function**C**

```
bool PLIB_TMR_ExistsCounter8Bit(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Counter8bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_Counter8BitSet](#)
- [PLIB_TMR_Counter8BitGet](#)
- [PLIB_TMR_Counter8BitClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Counter8bit feature is supported on the device
- false - The Counter8bit feature is not supported on the device

Remarks

None.

5.6.24.5.11.7 PLIB_TMR_ExistsCounterAsyncWriteControl Function**C**

```
bool PLIB_TMR_ExistsCounterAsyncWriteControl(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the CounterAsyncWriteControl feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_CounterAsyncWriteEnable](#)

- [PLIB_TMR_CounterAsyncWriteDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CounterAsyncWriteControl feature is supported on the device
- false - The CounterAsyncWriteControl feature is not supported on the device

Remarks

None.

5.6.24.5.11.8 PLIB_TMR_ExistsCounterAsyncWriteInProgress Function

C

```
bool PLIB_TMR_ExistsCounterAsyncWriteInProgress(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the CounterAsyncWriteInProgress feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_CounterAsyncWriteInProgress](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CounterAsyncWriteInProgress feature is supported on the device
- false - The CounterAsyncWriteInProgress feature is not supported on the device

Remarks

None.

5.6.24.5.11.9 PLIB_TMR_ExistsCountMode Function

C

```
bool PLIB_TMR_ExistsCountMode(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the CountMode feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_ContinuousCountModeEnable](#)

- [PLIB_TMR_SingleShotModeEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The CountMode feature is supported on the device
- false - The CountMode feature is not supported on the device

Remarks

None.

5.6.24.5.11.10 PLIB_TMR_ExistsEnableControl Function**C**

```
bool PLIB_TMR_ExistsEnableControl(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_Start](#)
- [PLIB_TMR_Stop](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Remarks

None.

5.6.24.5.11.11 PLIB_TMR_ExistsGateCurrentState Function**C**

```
bool PLIB_TMR_ExistsGateCurrentState(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the GateCurrentState feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_GateCurrentStateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The GateCurrentState feature is supported on the device
- false - The GateCurrentState feature is not supported on the device

Remarks

None.

5.6.24.5.11.12 PLIB_TMR_ExistsGatedTimeAccumulation Function**C**

```
bool PLIB_TMR_ExistsGatedTimeAccumulation(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the GatedTimeAccumulation feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_GateEnable](#)
- [PLIB_TMR_GateDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The GatedTimeAccumulation feature is supported on the device
- false - The GatedTimeAccumulation feature is not supported on the device

Remarks

None.

5.6.24.5.11.13 PLIB_TMR_ExistsGatePolarity Function**C**

```
bool PLIB_TMR_ExistsGatePolarity(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the GatePolarity feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_GatePolaritySelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The GatePolarity feature is supported on the device
- false - The GatePolarity feature is not supported on the device

Remarks

None.

5.6.24.5.11.14 PLIB_TMR_ExistsGateSinglePulseAcquisition Function**C**

```
bool PLIB_TMR_ExistsGateSinglePulseAcquisition(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the GateSinglePulseAcquisition feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_GateSinglePulseAcquisitionStart](#)
- [PLIB_TMR_GateSinglePulseAcquisitionHasCompleted](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The GateSinglePulseAcquisition feature is supported on the device
- false - The GateSinglePulseAcquisition feature is not supported on the device

Remarks

None.

5.6.24.5.11.15 PLIB_TMR_ExistsGateSinglePulseMode Function**C**

```
bool PLIB_TMR_ExistsGateSinglePulseMode(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the GateSinglePulseMode feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_GateSinglePulseModeEnable](#)
- [PLIB_TMR_GateSinglePulseModeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The GateSinglePulseMode feature is supported on the device
- false - The GateSinglePulseMode feature is not supported on the device

Remarks

None.

5.6.24.5.11.16 PLIB_TMR_ExistsGateSource Function**C**

```
bool PLIB_TMR_ExistsGateSource(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the GateSource feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_GateSourceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The GateSource feature is supported on the device
- false - The GateSource feature is not supported on the device

Remarks

None.

5.6.24.5.11.17 PLIB_TMR_ExistsGateToggleMode Function**C**

```
bool PLIB_TMR_ExistsGateToggleMode(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the GateToggleMode feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_GateToggleModeEnable](#)
- [PLIB_TMR_GateToggleModeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The GateToggleMode feature is supported on the device
- false - The GateToggleMode feature is not supported on the device

Remarks

None.

5.6.24.5.11.18 PLIB_TMR_ExistsMode16Bit Function

C

```
bool PLIB_TMR_ExistsMode16Bit(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Mode16Bit feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_Mode16BitEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Mode16Bit feature is supported on the device
- false - The Mode16Bit feature is not supported on the device

Remarks

None.

5.6.24.5.11.19 PLIB_TMR_ExistsMode32Bit Function

C

```
bool PLIB_TMR_ExistsMode32Bit(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Mode32Bit feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_Mode32BitEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Mode32Bit feature is supported on the device
- false - The Mode32Bit feature is not supported on the device

Remarks

None.

5.6.24.5.11.20 PLIB_TMR_ExistsMode8Bit Function**C**

```
bool PLIB_TMR_ExistsMode8Bit(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Mode8Bit feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_Mode8BitEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Mode8Bit feature is supported on the device
- false - The Mode8Bit feature is not supported on the device

Remarks

None.

5.6.24.5.11.21 PLIB_TMR_ExistsOperationInSleep Function**C**

```
bool PLIB_TMR_ExistsOperationInSleep(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the OperationInSleep feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_OperateInSleepEnable](#)

- [PLIB_TMR_OperateInSleepDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OperationInSleep feature is supported on the device
- false - The OperationInSleep feature is not supported on the device

Remarks

None.

5.6.24.5.11.22 PLIB_TMR_ExistsPeriod16Bit Function**C**

```
bool PLIB_TMR_ExistsPeriod16Bit(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Period16Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_Period16BitSet](#)
- [PLIB_TMR_Period16BitGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Period16Bit feature is supported on the device
- false - The Period16Bit feature is not supported on the device

Remarks

None.

5.6.24.5.11.23 PLIB_TMR_ExistsPeriod32Bit Function**C**

```
bool PLIB_TMR_ExistsPeriod32Bit(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Period32Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_Period32BitSet](#)
- [PLIB_TMR_Period32BitGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Period32Bit feature is supported on the device
- false - The Period32Bit feature is not supported on the device

Remarks

None.

5.6.24.5.11.24 PLIB_TMR_ExistsPeriod8Bit Function**C**

```
bool PLIB_TMR_ExistsPeriod8Bit(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Period8Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_Period8BitSet](#)
- [PLIB_TMR_Period8BitGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Period8Bit feature is supported on the device
- false - The Period8Bit feature is not supported on the device

Remarks

None.

5.6.24.5.11.25 PLIB_TMR_ExistsPostscale Function**C**

```
bool PLIB_TMR_ExistsPostscale(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Postscale feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_PostscaleSelect](#)
- [PLIB_TMR_PostscaleGet](#)
- [PLIB_TMR_PostscaleDivisorGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Postscale feature is supported on the device
- false - The Postscale feature is not supported on the device

Remarks

None.

5.6.24.5.11.26 PLIB_TMR_ExistsPrescale Function**C**

```
bool PLIB_TMR_ExistsPrescale(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the Prescale feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_PrescaleSelect](#)
- [PLIB_TMR_PrescaleGet](#)
- [PLIB_TMR_PrescaleDivisorGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Prescale feature is supported on the device
- false - The Prescale feature is not supported on the device

Remarks

None.

5.6.24.5.11.27 PLIB_TMR_ExistsPrescalerAssignment Function

C

```
bool PLIB_TMR_ExistsPrescalerAssignment(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the PrescalerControl feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_PrescalerEnable](#)
- [PLIB_TMR_PrescalerDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PrescalerControl feature is supported on the device
- false - The PrescalerControl feature is not supported on the device

Remarks

None.

5.6.24.5.11.28 PLIB_TMR_ExistsStopInIdleControl Function

C

```
bool PLIB_TMR_ExistsStopInIdleControl(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_StopInIdleEnable](#)
- [PLIB_TMR_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Remarks

None.

5.6.24.5.11.29 PLIB_TMR_ExistsSystemClockStatus Function**C**

```
bool PLIB_TMR_ExistsSystemClockStatus(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the SystemClockStatus feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_SystemClockFromTimerIsActive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SystemClockStatus feature is supported on the device
- false - The SystemClockStatus feature is not supported on the device

Remarks

None.

5.6.24.5.11.30 PLIB_TMR_ExistsTimerAssignment Function**C**

```
bool PLIB_TMR_ExistsTimerAssignment(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the TimerAssignment feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_AssignmentSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TimerAssignment feature is supported on the device
- false - The TimerAssignment feature is not supported on the device

Remarks

None.

5.6.24.5.11.31 PLIB_TMR_ExistsTimerOperationMode Function**C**

```
bool PLIB_TMR_ExistsTimerOperationMode(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the TimerOperationMode feature is available on the Timer module. When this function returns true, this function is supported on the device:

- [PLIB_TMR_IsPeriodMatchBased](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TimerOperationMode feature is supported on the device
- false - The TimerOperationMode feature is not supported on the device

Remarks

None.

5.6.24.5.11.32 PLIB_TMR_ExistsTimerOscillator Function**C**

```
bool PLIB_TMR_ExistsTimerOscillator(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the TimerOscillator feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_TimerOscillatorEnable](#)
- [PLIB_TMR_TimerOscillatorDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TimerOscillator feature is supported on the device

- false - The TimerOscillator feature is not supported on the device

Remarks

None.

5.6.24.5.11.33 PLIB_TMR_ExistsTriggerEventReset Function

C

```
bool PLIB_TMR_ExistsTriggerEventReset(  
    TMR_MODULE_ID index  
);
```

Description

This function identifies whether the TriggerEventReset feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- [PLIB_TMR_TriggerEventResetEnable](#)
- [PLIB_TMR_TriggerEventResetDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TriggerEventReset feature is supported on the device
- false - The TriggerEventReset feature is not supported on the device

Remarks

None.

5.6.24.6 Files

Files

Name	Description
plib_tmr.h	Timer Peripheral Library interface header.

Description

5.6.24.6.1 plib_tmr.h

Timer Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Timer Peripheral Library.

Functions

	Name	Description
	PLIB_TMR_AssignmentSelect	Assigns the specified Timer(s) to the selected modules.
	PLIB_TMR_ClockSourceEdgeGet	Gets the source edge information.
	PLIB_TMR_ClockSourceEdgeSelect	Selects the input clock source edge.
	PLIB_TMR_ClockSourceExternalSyncDisable	Disables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceExternalSyncEnable	Enables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceSelect	Selects the clock source.
	PLIB_TMR_ContinuousCountModeEnable	Enables Continuous Count mode
	PLIB_TMR_Counter16BitClear	Clears the 16-bit timer value.
	PLIB_TMR_Counter16BitGet	Gets the 16-bit timer value.
	PLIB_TMR_Counter16BitSet	Sets the 16-bit timer value.
	PLIB_TMR_Counter32BitClear	Clears the 32-bit timer value.
	PLIB_TMR_Counter32BitGet	Gets the 32-bit timer value.
	PLIB_TMR_Counter32BitSet	Sets the 32-bit timer value.
	PLIB_TMR_Counter8BitClear	Clears the 8-bit timer value.
	PLIB_TMR_Counter8BitGet	Gets the 8-bit timer value.
	PLIB_TMR_Counter8BitSet	Sets the 8-bit timer value.
	PLIB_TMR_CounterAsyncWriteDisable	Disables the writes to the counter register until the pending write operation completes.
	PLIB_TMR_CounterAsyncWriteEnable	Enables back-to-back writes with legacy asynchronous timer functionality.
	PLIB_TMR_CounterAsyncWriteInProgress	Returns the status of the counter write in Asynchronous mode.
	PLIB_TMR_ExistsClockSource	Identifies whether the ClockSource feature exists on the Timer module.
	PLIB_TMR_ExistsClockSourceEdge	Identifies whether the ClockSourceEdge feature exists on the Timer module.
	PLIB_TMR_ExistsClockSourceSync	Identifies whether the ClockSourceSync feature exists on the Timer module.
	PLIB_TMR_ExistsCounter16Bit	Identifies whether the Counter16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounter32Bit	Identifies whether the Counter32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounter8Bit	Identifies whether the Counter8bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteControl	Identifies whether the CounterAsyncWriteControl feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteInProgress	Identifies whether the CounterAsyncWriteInProgress feature exists on the Timer module.
	PLIB_TMR_ExistsCountMode	Identifies whether the CountMode feature exists on the Timer module.
	PLIB_TMR_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Timer module.
	PLIB_TMR_ExistsGateCurrentState	Identifies whether the GateCurrentState feature exists on the Timer module.
	PLIB_TMR_ExistsGatedTimeAccumulation	Identifies whether the GatedTimeAccumulation feature exists on the Timer module.

	<u>PLIB_TMR_ExistsGatePolarity</u>	Identifies whether the GatePolarity feature exists on the Timer module.
	<u>PLIB_TMR_ExistsGateSinglePulseAcquisition</u>	Identifies whether the GateSinglePulseAcquisition feature exists on the Timer module.
	<u>PLIB_TMR_ExistsGateSinglePulseMode</u>	Identifies whether the GateSinglePulseMode feature exists on the Timer module.
	<u>PLIB_TMR_ExistsGateSource</u>	Identifies whether the GateSource feature exists on the Timer module.
	<u>PLIB_TMR_ExistsGateToggleMode</u>	Identifies whether the GateToggleMode feature exists on the Timer module.
	<u>PLIB_TMR_ExistsMode16Bit</u>	Identifies whether the Mode16Bit feature exists on the Timer module.
	<u>PLIB_TMR_ExistsMode32Bit</u>	Identifies whether the Mode32Bit feature exists on the Timer module.
	<u>PLIB_TMR_ExistsMode8Bit</u>	Identifies whether the Mode8Bit feature exists on the Timer module.
	<u>PLIB_TMR_ExistsOperationInSleep</u>	Identifies whether the OperationInSleep feature exists on the Timer module.
	<u>PLIB_TMR_ExistsPeriod16Bit</u>	Identifies whether the Period16Bit feature exists on the Timer module.
	<u>PLIB_TMR_ExistsPeriod32Bit</u>	Identifies whether the Period32Bit feature exists on the Timer module.
	<u>PLIB_TMR_ExistsPeriod8Bit</u>	Identifies whether the Period8Bit feature exists on the Timer module.
	<u>PLIB_TMR_ExistsPostscale</u>	Identifies whether the Postscale feature exists on the Timer module.
	<u>PLIB_TMR_ExistsPrescale</u>	Identifies whether the Prescale feature exists on the Timer module.
	<u>PLIB_TMR_ExistsPrescalerAssignment</u>	Identifies whether the PrescalerControl feature exists on the Timer module.
	<u>PLIB_TMR_ExistsStopInIdleControl</u>	Identifies whether the StopInIdle feature exists on the Timer module.
	<u>PLIB_TMR_ExistsSystemClockStatus</u>	Identifies whether the SystemClockStatus feature exists on the Timer module.
	<u>PLIB_TMR_ExistsTimerAssignment</u>	Identifies whether the TimerAssignment feature exists on the Timer module.
	<u>PLIB_TMR_ExistsTimerOperationMode</u>	Identifies whether the TimerOperationMode feature exists on the Timer module.
	<u>PLIB_TMR_ExistsTimerOscillator</u>	Identifies whether the TimerOscillator feature exists on the Timer module.
	<u>PLIB_TMR_ExistsTriggerEventReset</u>	Identifies whether the TriggerEventReset feature exists on the Timer module.
	<u>PLIB_TMR_GateCurrentStateGet</u>	Returns the current level of the timer gate.
	<u>PLIB_TMR_GateDisable</u>	Enables counting regardless of the corresponding timer gate function.
	<u>PLIB_TMR_GateEnable</u>	Enables counting controlled by the corresponding gate function.
	<u>PLIB_TMR_GatePolaritySelect</u>	Selects the timer gate polarity.
	<u>PLIB_TMR_GateSinglePulseAcquisitionHasCompleted</u>	Returns the gate single pulse acquisition status.
	<u>PLIB_TMR_GateSinglePulseAcquisitionStart</u>	Starts the gate single pulse acquisition.
	<u>PLIB_TMR_GateSinglePulseModeDisable</u>	Disables Single Pulse mode.

	PLIB_TMR_GateSinglePulseModeEnable	Enables Single pulse mode and the controlling corresponding timer gate.
	PLIB_TMR_GateSourceSelect	Selects the timer gate source.
	PLIB_TMR_GateToggleModeDisable	Disables the Gate Toggle mode.
	PLIB_TMR_GateToggleModeEnable	Enables the Gate Toggle mode.
	PLIB_TMR_IsPeriodMatchBased	Gets the operating mode state of the Timer module based on Period Match or Overflow mode.
	PLIB_TMR_Mode16BitEnable	Enables the Timer module for 16-bit operation and disables all other modes.
	PLIB_TMR_Mode32BitEnable	Enables 32-bit operation on the Timer module combination.
	PLIB_TMR_Mode8BitEnable	Enables the Timer module in 8-bit operation mode and disables 16-bit mode.
	PLIB_TMR_OperateInSleepDisable	Disables Timer module operation when the device is in Sleep mode.
	PLIB_TMR_OperateInSleepEnable	Enables Timer module operation when the device is in Sleep mode.
	PLIB_TMR_Period16BitGet	Gets the 16-bit period value.
	PLIB_TMR_Period16BitSet	Sets the 16-bit period value.
	PLIB_TMR_Period32BitGet	Gets the 32-bit period value.
	PLIB_TMR_Period32BitSet	Sets the 32-bit period value.
	PLIB_TMR_Period8BitGet	Gets the 8-bit period value.
	PLIB_TMR_Period8BitSet	Sets the 8-bit period value.
	PLIB_TMR_PostscaleDivisorGet	Gets the postscale divisor information.
	PLIB_TMR_PostscaleGet	Gets the postscale divisor information.
	PLIB_TMR_PostscaleSelect	Selects the output clock postscale.
	PLIB_TMR_PrescaleDivisorGet	Gets the prescaler divisor information.
	PLIB_TMR_PrescaleGet	Gets the prescaler information.
	PLIB_TMR_PrescalerDisable	Disables the prescaler assignment to the indexed Timer module.
	PLIB_TMR_PrescalerEnable	Enables the prescaler assignment to the indexed Timer module.
	PLIB_TMR_PrescaleSelect	Selects the clock prescaler.
	PLIB_TMR_SingleShotModeEnable	Enables Single-shot mode.
	PLIB_TMR_Start	Starts/enables the indexed timer.
	PLIB_TMR_Stop	Stops/disables the indexed timer.
	PLIB_TMR_StopInIdleDisable	Continue module operation when the device enters Idle mode.
	PLIB_TMR_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.
	PLIB_TMR_SystemClockFromTimerIsActive	Gets the system clock status.
	PLIB_TMR_TimerOscillatorDisable	Disables the oscillator associated with the Timer module.
	PLIB_TMR_TimerOscillatorEnable	Enables the oscillator associated with the Timer module.
	PLIB_TMR_TriggerEventResetDisable	Disables the special event trigger reset.
	PLIB_TMR_TriggerEventResetEnable	Enables the special event trigger reset.

File Name

plib_tmr.h

Company

Microchip Technology Inc.

5.6.25 USART Peripheral Library

5.6.25.1 Introduction

USART Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the USART Peripheral Library on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description**Overview**

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) module is the key component of the serial communications sub-system of a computer. The USART takes the data and transmits the individual bits in a sequential fashion. While receiving, the USART reassembles the bits into complete bytes. The data can be **8-bit** or **9-bit**.

There are two primary forms of serial transmission: Asynchronous and Synchronous.

Asynchronous

Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters (i.e., baud rate) in advance and special bits are added to each word, which are used to synchronize the sending and receiving units.

When a word is given to the USART for asynchronous transmissions, a bit called the "Start" bit is added to the beginning of each word that is to be transmitted. The Start bit is used to alert the receiver that a word of data is about to be sent. After the Start bit, the individual bits of the word of data are sent, with the Least Significant bit (LSb) being sent first. When the entire data word has been sent, the transmitter may add a Parity bit that the transmitter generates. The Parity bit may be used by the receiver to perform simple error checking. Then, at least one Stop bit is sent by the transmitter.

The Start bit is always a space and the Stop bits are always marks. Each transmitted bit persists for a period of $1/(\text{Baud Rate})$. An on-chip dedicated Baud Rate Generator (BRG) is used to derive standard baud rate frequencies.

When the receiver has received all of the bits in the data word, it may check for the Parity bits (both sender and receiver must agree on whether a Parity bit is to be used), and then the receiver looks for a Stop bit. If the Stop bit does not appear when it is supposed to, the USART module considers the entire word to be unintelligible and will report a framing error to the host processor when the data word is read.

Note: Please refer to the the specific device data sheet or the related family reference manual section for details.

Synchronous

Synchronous serial transmission requires that the sender and receiver share a clock with one another, or that the sender provide a strobe or other timing signal so that the receiver knows when to "read" the next bit of the data. This mode is not available on all microcontrollers.

It is typically used in systems with a single master and one or more slaves. The master device contains the necessary circuitry for baud rate generation and supplies the clock for all devices in the system.

There are two signal lines a bidirectional data line and a clock line. Since the data line is bidirectional, synchronous operation is only half-duplex.

Start and Stop bits are not used in synchronous transmissions.

5.6.25.2 Release Notes

MPLAB Harmony Version: v0.70b **USART Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.25.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.25.4 Using the Library

This topic describes the basic architecture of the USART Peripheral Library and provides information and examples on how to use it.

Interface Header File: peripheral.h

The interface to the USART Peripheral library is defined in the "plib_usart.h" header file. Any C language source (.c) file that uses the USART Peripheral library should include "peripheral.h".

Library File:

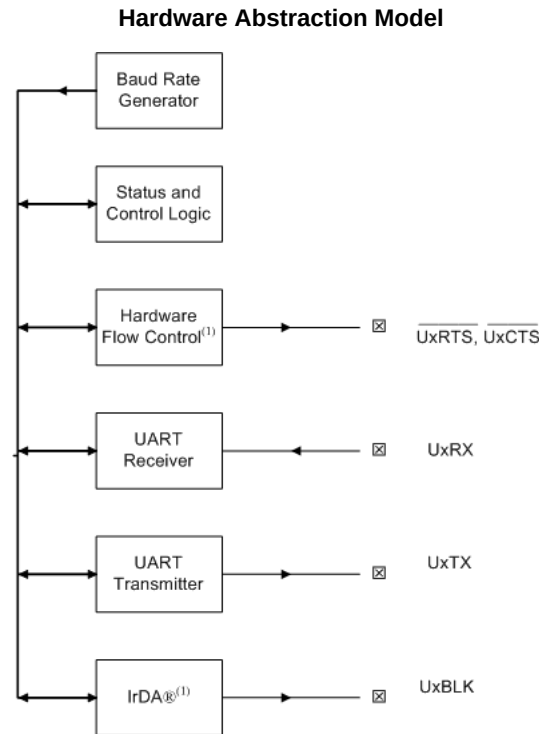
The USART Peripheral library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the library interacts with the framework.

5.6.25.4.1 Hardware Abstraction Model

This section describes the hardware abstraction model.

Description



⁽¹⁾ - Not available in all microcontrollers

Hardware Abstraction Model

The USART Peripheral Library provides interface routines to interact with the blocks shown in the diagram.

The **Baud Rate Controller** controls the timing in Asynchronous mode, the desired baud rate can be programmed in the baud rate controller or, the baud rate controller can be configured to determine the baud rate of the transmitting device automatically.

The **Transmitter** and **Receiver** are capable of handling 8-bit or 9-bit data, which can be programmed in the control logic. The 9th bit is used to transfer the address or the parity. Enable 9-bit mode if the hardware is to be used for address or parity detection.

The **Status and Control logic**, provide the ability to control different ways the transmitter, receiver, and the baud rate controller can function. It also can provides status about the transmitter, receiver, or the baud rate controller.

The **Hardware Flow Control** uses CTS and the RTS lines to perform handshaking. Flow control is the ability to stop, and then restart the flow without any loss of bytes. Flow control is needed for modems and other hardware to allow a jump in instantaneous flow rates.

Note: This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

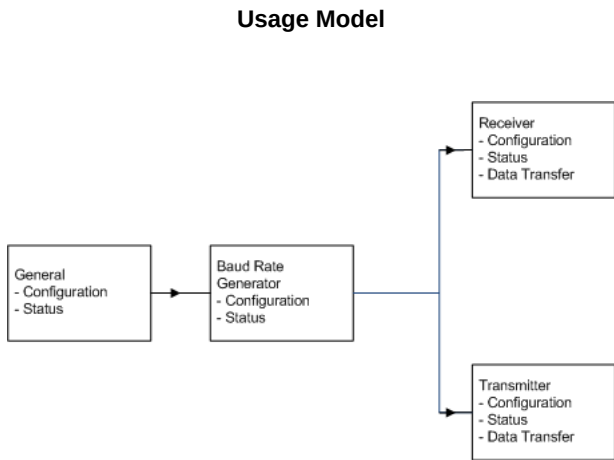
The USART module provides two types of infrared USART support: one is the **IrDA** clock output to support the external IrDA encoder and decoder, and the other is the full implementation of the IrDA encoder and decoder.

Note: This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

5.6.25.4.2 Library Usage Model

This section describes the library usage model.

Description



Each of the blocks in the diagram correspond to the library interface section.

Library Interface Section	Description

5.6.25.4.3 How the Library Works

This section provides information on using the peripheral library with the USART module. The usage model for different scenarios is also described.

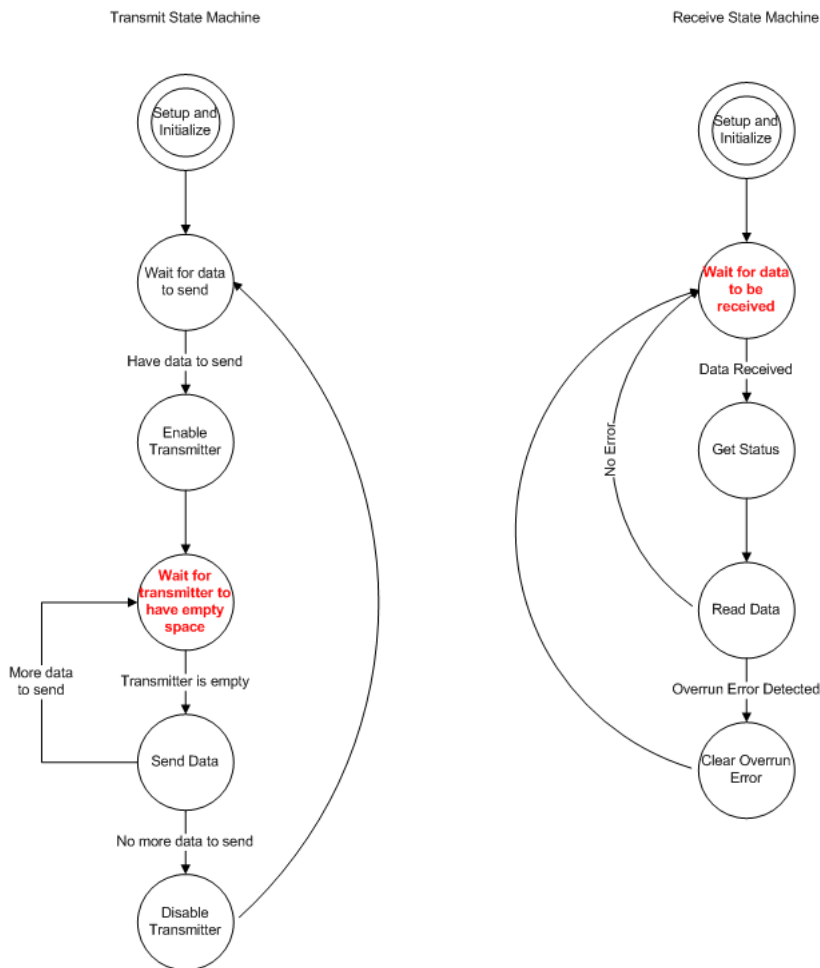
5.6.25.4.3.1 State Machine

This section describes the transmission and reception state machine.

Description

The following state machine is for the transmission and reception during normal operation. This state machine provides a general idea of the usage model of the USART Peripheral Library. Refer to the usage model for more detailed steps for the scenario that is being used.

Transmit and Receive State machine

**Notes:**

1. The wait states shown in red in the diagram can either be achieved using polled functions or using interrupts.
2. Refer to the specific USART mode in use for the setup and initialization steps.
3. Refer to the following table for the functions that can be used at each state.

State	Associated Function
Setup and initialization	Refer to the specific USART mode for detailed setup instructions.
Wait for data to send	Once the USART module has been appropriately set up and initialized, the state machine waits for the application to issue one of the send data functions.
Wait for data to be received	When the application is using Polled mode, the application can use the function PLIB_USART_ReceiverDataIsAvailable . When the application is using Interrupt mode, the application should wait for a receive interrupt.
Enable transmitter	Use the function PLIB_USART_TransmitterEnable to enable the transmitter

Wait for transmitter to have empty space	When the application is using polled mode, the application can use the function PLIB_USART_TransmitterBufferIsFull . When the application is using Interrupt mode, the application should wait for a transmit interrupt.
Send data	Use the function PLIB_USART_TransmitterByteSend to send data. If an address needs to be transmitted or if a byte needs to be transmitted with a Parity bit use the function PLIB_USART_Transmitter9BitsSend .
Disable transmitter	Use the function PLIB_USART_TransmitterDisable to disable the transmitter.
Get status	To get the status of the hardware module, use the set of functions, PLIB_USART_ReceiverFramingErrorHasOccurred , PLIB_USART_ReceiverParityErrorHasOccurred , and PLIB_USART_ReceiverOverrunHasOccurred .
Read data	Use the function PLIB_USART_ReceiverByteReceive to read data.
Clear overrun error	Clear the overrun error using the function PLIB_USART_ReceiverOverrunErrorClear .

5.6.25.4.3.2 Asynchronous USART Mode

This section provides processes for setting up asynchronous transmission and reception.

Description

Asynchronous Transmission

Setup

1. Set the desired baud rate using either [PLIB_USART_BaudRateHighSet](#) or [PLIB_USART_BaudRateSet](#).
2. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
3. *If available on the device*, enable the asynchronous serial port using [PLIB_USART_SyncModeSelect](#).
4. Set the number of data bits (8 or 9) and other line control modes using [PLIB_USART_LineControlModeSelect](#).
5. *Optional (if available on the device)*: If desired, enable the transmit interrupt.
 - Ensure that the interrupts are enabled for the system and the peripherals
 - Select the interrupt priority
 - Set up the transmit FIFO interrupt mode using [PLIB_USART_TransmitterInterruptModeSelect](#). Refer to [USART_TRANSMIT_INTR_MODE](#) for a list of possible values.
6. Enable the USART module using the function [PLIB_USART_Enable](#)

Example: Asynchronous Transmission Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
//being used by the system.
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the asynchronous serial port.
// enable asynchronous mode if the part supports it
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);

/* Select 8 data bits, No parity and one stop bit */
PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE, USART_8N1);
```

```
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.
```

Enable Transmission

1. Enable the transmission using [PLIB_USART_TransmitterEnable](#).
2. Ensure that the transmit buffer is not full, before attempting to write into the transmit buffer. Either poll using [PLIB_USART_TransmitterBufferIsFull](#) or wait for the transmit interrupt.
3. Start the transmission using [PLIB_USART_TransmitterByteSend](#) (if transmitting 8-bit data) or [PLIB_USART_Transmitter9BitsSend](#) (if transmitting an address or if transmitting a byte with parity).
4. After the transmission is complete, disable the transmitter.

Example: Asynchronous Transmission

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// TODO - Either wait for Transmit buffer to be not full or wait for
//an transmit interrupt.

// Transmit the byte when Transmit buffer is empty.
// where, my_byte is the 8 bit data to be transmitted
PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, my_byte);
```

Asynchronous Transmission of Break Characters

A break characters consists of a **Start Bit** followed by twelve bits of "0" and a **Stop Bit**.

Setup and Transmission

1. Configure the desired mode, using the previous steps described.
2. Enable the transmitter using [PLIB_USART_TransmitterEnable](#).
3. Transmit the break character using [PLIB_USART_TransmitterBreakSend](#) function.
4. Check if the break transmission is complete using [PLIB_USART_TransmitterBreakSendIsComplete](#).
5. Transmission of the next bytes can start.

Example: USART Break and Sync Transmission

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
//being used by the system.
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the asynchronous serial port.
// enable asynchronous mode if the part supports it
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);

/* Select 8 data bits, No parity and one stop bit */
PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE, USART_8N1);

// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.
```

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// Start the transmission
// Transmit the break character
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);

// Transmit the byte
while(PLIB_USART_TransmitterBreakSendIsComplete(MY_USART_INSTANCE));

PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, 0x55);
```

Refer to the following steps for setting up asynchronous receptions and how to receive data.

Asynchronous Reception

Setup

1. Set the desired baud rate using [PLIB_USART_BaudRateHighSet](#) or [PLIB_USART_BaudRateSet](#).
2. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
3. *If available on the device*, enable the asynchronous serial port using [PLIB_USART_SyncModeSelect](#).
4. Set the number of data bits (8 or 9) and other line control modes using [PLIB_USART_LineControlModeSelect](#).
5. *Optional (if available on the device)*: If desired, enable the receive interrupt.
 - Ensure that the interrupts are enabled for the system and the peripherals
 - Select the interrupt priority
 - Set up the receive FIFO interrupt mode using [PLIB_USART_ReceiverInterruptModeSelect](#). Refer to [USART_RECEIVE_INTR_MODE](#) for a list of possible values.
6. *Optional*: If inverted receive polarity is desired, use [PLIB_USART_ReceiverIdleStateLowEnable](#).
7. *If available on the device*, enable the reception using [PLIB_USART_ReceiverEnable](#).
8. Enable the USART module using [PLIB_USART_Enable](#).

Example: Asynchronous Reception Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently being
// used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable asynchronous mode if the part supports it
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Start the reception
// Enable the continuous reception
PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed, or poll using [PLIB_USART_ReceiverDataIsAvailable](#).

2. Read the status of the device using [PLIB_USART_ReceiverOverrunHasOccurred](#), [PLIB_USART_ReceiverParityErrorHasOccurred](#), and [PLIB_USART_ReceiverFramingErrorOccurred](#).
3. Read the data using [PLIB_USART_ReceiverByteReceive](#).
4. If an overrun error occurred, clear the error using [PLIB_USART_ReceiverOverrunErrorClear](#).

Example: Asynchronous Reception

```
// TODO - Either wait for Receive buffer to have data or wait for
// a receive interrupt.
// read the data
my_byte = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
// where, my_byte is the 8 bit data received

// If overrun error clear the error.
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Asynchronous Reception with Address Detection Mode

A typical multi-processor communication protocol will differentiate between data bytes and address/control bytes. A common method is to use a 9th data bit to identify whether a data byte is address or data information. If the 9th bit is set, the data is processed as address or control information. If the 9th bit is cleared, the received data word is processed as data associated with the previous address/control byte.

The protocol operates as follows:

- The master device transmits a data word with 9th bit set. The data word contains the address of a slave device.
- All slave devices in the communication chain receive the address word and check the slave address.
- The slave device that was addressed will receive and process subsequent data bytes sent by the master device. All other slave devices will discard subsequent data bytes until a new address word(9th bit set) is received.

This mode will typically be used in RS-485 systems.

Setup

1. Set the desired baud rate using [PLIB_USART_BaudRateHighSet](#) or [PLIB_USART_BaudRateSet](#).
2. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
3. *If available on the device*, enable the asynchronous serial port using [PLIB_USART_SyncModeSelect](#).
4. Set the number of data bits (8 or 9) and other line control modes using [PLIB_USART_LineControlModeSelect](#).
5. *Optional (if available on the device)*: If desired, enable the receive interrupt.
 - Ensure that the interrupts are enabled for the system and the peripherals. Also select the interrupt priority.
 - Set up the receive FIFO interrupt mode by using the function [PLIB_USART_ReceiverInterruptModeSelect](#). Refer to [USART_RECEIVE_INTR_MODE](#) for a list of possible values.
 - For Address Detect mode, configure [PLIB_USART_ReceiverInterruptModeSelect](#) with the receive FIFO interrupt mode as [USART_RECEIVE_FIFO_ONE_CHAR](#).
6. *Optional*: If inverted receive polarity is desired, use [PLIB_USART_ReceiverIdleStateLowEnable](#).
7. *If available on the device*, enable the reception using [PLIB_USART_ReceiverEnable](#).
8. Enable address detection using [PLIB_USART_ReceiverAddressDetectEnable](#).
9. Enable the USART module using [PLIB_USART_Enable](#).

Note: This feature is not available on all devices. On devices where automatic address detection is available, configure automatic address detection using [PLIB_USART_AutomaticAddressDetectModeSetup](#). Skip steps 2-6 in the following Reception section during the reception phase.

Example: Asynchronous Reception Setup (with address detect enable)

```

// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
// being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Set the reception to 9 bits to enable address byte.
PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE, USART_9N1);

// Enable address detection.
PLIB_USART_ReceiverAddressDetectEnable(MY_USART_INSTANCE);

// Start the reception
// Enable the continuous reception
PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);

```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed or poll using [PLIB_USART_ReceiverDataIsAvailable](#).
2. Read the address using [PLIB_USART_ReceiverByteReceive](#).
3. The application determines if this is the device address. If the device is addressed, disable address detect using [PLIB_USART_ReceiverAddressDetectDisable](#) to allow subsequent data bytes to be received; otherwise, discard the received word.
4. Read the status of the device using [PLIB_USART_ReceiverOverrunHasOccurred](#), [PLIB_USART_ReceiverParityErrorHasOccurred](#), and [PLIB_USART_ReceiverFramingErrorHasOccurred](#).
5. If an overrun error occurred, clear the error using [PLIB_USART_ReceiverOverrunErrorClear](#).
6. If the device has been addressed, disable the address detection using [PLIB_USART_ReceiverAddressDetectDisable](#), which will allow all received data into the receive buffer and generate interrupts if desired.
7. *Optional (if available on the device):* If a long packet is expected, the receive interrupt mode could be changed to buffer more than one data word between interrupts. Use [PLIB_USART_ReceiverInterruptModeSelect](#) with `USART_RECEIVE_FIFO_HALF_FULL` or `USART_RECEIVE_FIFO_3B4FULL`.
8. When the last data byte has been received, enable the address detect using [PLIB_USART_ReceiverAddressDetectEnable](#), so that only address bytes will be received.
9. *Optional (if available on the device):* Ensure that the receive interrupt is configured to be triggered after each received word using [PLIB_USART_ReceiverInterruptModeSelect](#) with `USART_RECEIVE_FIFO_ONE_CHAR`.

Example: Asynchronous Reception (with Address Detect)

```

// Either wait for Receive buffer to have data or wait for a receive interrupt.
int8_t my_address;

// Receive the address
if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    my_address = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
    // where, my_address is the address received
}

```

```

// If overrun error clear the error.
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}

if(my_address == MY_DEVICE_ADDRESS)
{
    // The device has been addressed.
    PLIB_USART_ReceiverAddressDetectDisable(MY_USART_INSTANCE);
}

// TODO - Receive the data bytes for the packet. Either wait for RX buffer
//to have data or wait for a receive interrupt.

// When all the bytes for the packet are received.

// Enable address detection mode
PLIB_USART_ReceiverAddressDetectEnable(MY_USART_INSTANCE);

```

5.6.25.4.3.3 Synchronous Master Mode

This section provides processes for setting up synchronous master transmission and reception.

Note: This feature is not available on all devices. Please refer to the specific device data sheet or related family reference manual to determine which features are supported for your device.

Description

Synchronous Master Transmission

Setup

1. Set the desired baud rate using [PLIB_USART_BaudRateSet](#) or [PLIB_USART_BaudRateHighSet](#).
2. Enable the synchronous master serial port using [PLIB_USART_SyncModeSelect](#), [PLIB_USART_SyncClockSourceSelect](#), and [PLIB_USART_Enable](#).
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. Disable receive using [PLIB_USART_ReceiverDisable](#).
5. *Optional:* If desired set the number of data bits to 9 (to [send](#) address byte or to enable parity check) using [PLIB_USART_Transmitter9BitModeEnable](#).
6. *Optional:* If desired, enable the transmit interrupt, ensuring that the interrupts are enabled for the system and the peripherals.

Example: Synchronous Master Transmission Setup

```

// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently being used by
// the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous master serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
// enable master clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_MASTER);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO Call the appropriate function.

```


Transmission

1. Enable transmission using [PLIB_USART_TransmitterEnable](#).
2. Ensure that the transmit buffer is not full before attempting to write onto the transmit buffer. Either poll using [PLIB_USART_TransmitterBufferIsFull](#) or wait for the transmit interrupt.

Note: If using interrupts, make sure that interrupts are also enabled for the system and the peripherals.

3. Start the transmission using [PLIB_USART_TransmitterByteSend](#) (if transmitting 8 - bit data) or [PLIB_USART_Transmitter9BitsSend](#) (if transmitting address or if transmitting a byte with parity).
4. After the transmission is complete, disable the transmitter.

Example: Synchronous Master Transmission

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// TODO Either wait for Transmit buffer to be not full or wait for an transmit interrupt.

// Transmit the byte when Transmit buffer is empty.
// where, my_byte is the 8 bit data to be transmitted
PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, my_byte);
```

Synchronous Master Reception

Setup

1. Set the desired baud rate using [PLIB_USART_BaudRateSet](#) or [PLIB_USART_BaudRateHighSet](#).
2. Enable the synchronous master serial port using [PLIB_USART_SyncModeSelect](#), [PLIB_USART_SyncClockSourceSelect](#), and [PLIB_USART_Enable](#).
3. Set the appropriate IO direction and corresponding to RX and TX I/O pins.
4. *Optional:* If desired, enable the receive interrupt, ensuring that the interrupts are enabled for the system and the peripherals.
5. Start the reception using [PLIB_USART_ReceiverDisable](#).

Example: Synchronous Master Reception Setup

```
uint32_t baudRate = 9600;
// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently being used by
the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

PLIB_USART_HandshakeModeSelect(MY_USART_INSTANCE,
                               USART_HANDSHAKE_MODE_SIMPLEX);
// Enable the synchronous master serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_SYNC_MODE);
// enable master clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_MASTER);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Enable receive mode
PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed, or poll using [PLIB_USART_ReceiverDataIsAvailable](#).

Note: If using interrupts, ensure that interrupts are also enabled for the system and the peripherals.

2. Read the status of the device using [PLIB_USART_ReceiverOverrunHasOccurred](#), [PLIB_USART_ReceiverParityErrorHasOccurred](#) and [PLIB_USART_ReceiverFramingErrorHasOccurred](#).
3. Read the data using [PLIB_USART_ReceiverByteReceive](#).
4. If an overrun error occurred, clear the error using the function [PLIB_USART_ReceiverOverrunErrorClear](#).

Example: Synchronous Master Reception

```
// Read the byte
// where, my_byte is the 8 bit data to be transmitted
my_byte = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

5.6.25.4.3.4 Synchronous Slave Mode

This section provides processes for setting up synchronous slave transmission and reception.

Note: This feature is not available on all devices. Please refer to the specific device data sheet or related family reference manual to determine which features are supported for your device.

Description

Synchronous Slave Transmission

Setup

1. Set the desired baud rate using either [PLIB_USART_BaudRateHighSet](#) or [PLIB_USART_BaudRateSet](#).
2. Enable the synchronous slave serial port using [PLIB_USART_SyncModeSelect](#), [PLIB_USART_ClockSourceSlaveSelect](#), and [PLIB_USART_Enable](#).
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. Disable receive mode using [PLIB_USART_ReceiverDisable](#).
5. *Optional:* If desired, enable the transmit interrupt, ensuring that the interrupts are enabled for the system and the peripherals.

Example: Synchronous Slave Transmission Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware
// peripheral MY_CLOCK_FREQUENCY - is hardware clock frequency which is
// currently being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_SYNC_MODE);
// enable slave clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_SLAVE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Disable receive mode
PLIB_USART_ReceiverDisable(MY_USART_INSTANCE);

// TODO - If using interrupts Enable the transmit interrupt
// TODO - Call the appropriate function for the transmit enable interrupt,
```

// TODO - Also, make sure that the interrupts are enabled for system and the peripherals.

Transmission

1. Enable transmission using [PLIB_USART_TransmitterEnable](#).
2. Ensure that the transmit buffer is not full before attempting to write into the transmit buffer. Either poll using [PLIB_USART_TransmitterBufferIsFull](#) or wait for the transmit interrupt.

Note: If using interrupts, ensure that interrupts are also enabled for the system and the peripherals.

3. Start the transmission using [PLIB_USART_TransmitterByteSend](#) (if transmitting 8-bit data) or [PLIB_USART_Transmitter9BitsSend](#) (if transmitting address if transmitting a byte with parity)
4. After the transmission is complete, disable the transmitter.

Example: Synchronous Slave Transmission

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// TODO - Either wait for Transmit buffer to be not full or wait
//for an transmit interrupt.

// Transmit the byte when Transmit buffer is empty.
PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, my_word);
// where, my_word is the 9 bit data to be transmitted
```

Synchronous Slave Reception

Setup

1. Set the desired baud rate using [PLIB_USART_BaudRateHighSet](#) or [PLIB_USART_BaudRateSet](#).
2. Enable the synchronous master serial port using [PLIB_USART_SyncModeSelect](#), [PLIB_USART_SyncClockSourceSelect](#), and [PLIB_USART_Enable](#).
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. *Optional:* If desired, enable the receive interrupt, ensuring that the interrupts are enabled for the system and the peripherals.
5. Start the reception using [PLIB_USART_ReceiverEnable](#).

Example: Synchronous Slave Reception Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
//being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_SYNC_MODE);
// enable master clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_SLAVE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Start the reception
// Enable the continuous reception
PLIB_USART_ReceiverDisable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed, or poll using [PLIB_USART_ReceiverDataIsAvailable](#).

Note: If using interrupts make sure that interrupts are also enabled for the system and the peripherals.

2. Read the status of the device using [PLIB_USART_ReceiverOverrunHasOccurred](#), [PLIB_USART_ReceiverParityErrorHasOccurred](#), and [PLIB_USART_ReceiverFramingErrorHasOccurred](#).
3. Read the data using [PLIB_USART_ReceiverByteReceive](#).
4. If an overrun error occurred, clear the error using [PLIB_USART_ReceiverOverrunErrorClear](#).

Example: Synchronous Slave Reception

```
// Transmit the data
my_byte = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
// where, my_byte is the 8 bit data received

// If overrun error clear the error.
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

5.6.25.4.3.5 Other Features

This section describes additional features of the USART Peripheral Library.

Note: The features described in this section are not available on all devices. Please refer to the specific device data sheet or the related family reference manual to determine which features are supported by your device.

Description

IrDA Support

Support for IrDA is available in some microcontrollers.

1. Enable IrDA support using [PLIB_USART_IrDAEnable](#). IrDA support can be disabled using [PLIB_USART_IrDADisable](#).
2. *Optional:* If using the external IrDA encoder and decoder, use the function [PLIB_USART_OperationModeSelect](#) with [USART_ENABLE_TX_RX_BCLK_USED](#) to output the 16x baud clock.
3. *Optional:* IrDA Transmit Polarity can be inverted using [PLIB_USART_TransmitterIdleIsLowEnable](#).
4. *Optional:* IrDA Receive Polarity can be inverted using [PLIB_USART_ReceiverIdleStateLowEnable](#).
5. Enable the USART using [PLIB_USART_Enable](#).

Loopback Support

This is the mode in which the transmission is internally connected to reception.

1. Set the desired baud rate using [PLIB_USART_BaudRateSet](#) or [PLIB_USART_BaudRateHighSet](#).
2. Set up the line control parameters (data bits, stop bits, parity, flow control) using [PLIB_USART_LineControlModeSelect](#). Refer to [USART_LINECONTROL_MODE](#) for a list of possible values.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. *Optional:* If desired, enable the transmit interrupt, also make sure that the interrupts are enabled for the system and the peripherals.
5. *Optional:* Also select the interrupt priority.
6. *Optional:* Also, set up the transmit FIFO interrupt mode using [PLIB_USART_TransmitterInterruptModeSelect](#). Refer to [USART_TRANSMIT_INTR_MODE](#) for a list of possible values.
7. Enable the USART module using [PLIB_USART_Enable](#) and enable the transmission using [PLIB_USART_TransmitterEnable](#).

8. Enable loopback using [PLIB_USART_LoopbackEnable](#). Loopback can be disabled at any time using [PLIB_USART_LoopbackDisable](#).
9. Continue with transmission and/or reception using the same steps as for asynchronous transmission/reception.

Auto Baud Support

1. Enable the auto baud detection using the function [PLIB_USART_BaudRateAutoDetectEnable](#), which requires a reception of the Sync character (0x55).
2. *Optional:* Poll [PLIB_USART_BaudRateAutoDetectIsComplete](#) to check if the baud rate is detected.
3. *Optional:* Call [PLIB_USART_BaudRateGet](#) to get the baud rate that was detected.
4. Clear the receive interrupt.
5. On some microcontrollers, reading the data using [PLIB_USART_ReceiverByteReceive](#) may be required.

Flow Control Support

The USART can use Simplex or Flow Control modes:

- Use [PLIB_USART_HandshakeModeSelect](#) to enable flow control (USART_HANDSHAKE_MODE_FLOW_CONTROL) or use simplex mode (USART_HANDSHAKE_MODE_SIMPLEX) selected from [USART_HANDSHAKE_MODE](#).
- To configure the USART line control features, use [PLIB_USART_LineControlModeSelect](#) selected from [USART_LINECONTROL_MODE](#).
- Additionally, the USART pins may need to be configured using [PLIB_USART_OperationModeSelect](#) and the appropriate values from [USART_OPERATION_MODE](#).

Operation During Sleep Mode

When the device enters Sleep mode, all clock sources supplied to the USART modules are shut down. If the device enters Sleep mode in the middle of a USART transmission or reception operation, the operation is aborted and the USART pins are driven to default state.

A Start bit, when detected, can wake up the device.

- The application needs to enable the wake-up on start bit using [PLIB_USART_WakeOnStartEnable](#) just before entering Sleep mode, to wake-up the device
- For the device to wake-up from sleep, reception of the Sync character or Wake-up Signal character is required
- *Optional:* If using interrupts, a receive interrupt is generated

Operation During Idle Mode

When the device enters Idle mode, the system clock sources remain functional and the CPU stops code execution.

- USART operation during Idle mode can be stopped using [PLIB_USART_StopInIdleEnable](#). When the device enters Idle mode, the USART module operation is similar to that as in Sleep mode (Refer to Operation in Sleep Mode).
- By default, the USART module continues to operate in Idle mode and provide full functionality

USART Operation With DMA

The DMA module can be used to transfer data between the USART and the CPU, without CPU assistance.

1. Set the desired baud rate using the function [PLIB_USART_BaudRateSet](#) or [PLIB_USART_BaudRateHighSet](#).
2. Set up the line control parameters (data bits, stop bits, parity, flow control) using the function [PLIB_USART_LineControlModeSelect](#). Refer to [USART_LINECONTROL_MODE](#) for a list of possible values.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. Enable the USART module using [PLIB_USART_Enable](#) and enable the transmission using [PLIB_USART_TransmitterEnable](#).
5. Enable the Transmit, Receive, and Error interrupts, ensuring that the interrupts are enabled for the system and the peripherals.

6. Select the interrupt priority.
7. Set up the Transmit FIFO Interrupt mode and Receive FIFO Interrupt mode using the `PLIB_USART_TransmitterFIFOLevelSelect` and `PLIB_USART_ReceiverFIFOLevelSelect` functions. To generate interrupts for every character received, use `USART_RECEIVE_FIFO_ONE_CHAR` from [USART_RECEIVE_INTR_MODE](#).
8. Configure the DMA to be used with the USART receiver and transmitter.

5.6.25.4.4 Configuring the Library

The library is configured for the supported USART modules when the processor is chosen in the MPLAB IDE.

5.6.25.5 Library Interface

Data Types and Constants

	Name	Description
	USART_BAUD_RATE_MODE	Lists the USART baud rate modes.
	USART_HANDSHAKE_MODE	Lists the USART handshake modes.
	USART_LINECONTROL_MODE	Data type defining the different configurations by which the USART data flow can be configured.
	USART_MODULE_ID	Enumeration: <code>USART_MODULE_ID</code> This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	USART_OPERATION_MODE	Data type defining the different configurations by which the USART can be enabled.
	USART_RECEIVE_INTR_MODE	Data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.
	USART_RECEIVE_MODES	Lists the USART receive modes.
	USART_SYNC_CLOCK_SOURCE	Lists the USART clock sources.
	USART_SYNC_MODES	Selection between asynchronous communication and synchronous communication.
	USART_TRANSMIT_INTR_MODE	Data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

Baud Rate Generator Functions

	Name	Description
	PLIB_USART_BaudRateAutoDetectEnable	Enables baud rate measurement on the next character, which requires reception of the Sync character.
	PLIB_USART_BaudRateAutoDetectIsComplete	Gets the state of the automatic baud detection.
	PLIB_USART_BaudRateAutoDetectOverflowHasOccurred	Identifies if the automatic baud timer has overflowed.
	PLIB_USART_BaudRateGet	Gets the baud rate current in use.
	PLIB_USART_BaudRateHighDisable	Disables the high baud rate selection.
	PLIB_USART_BaudRateHighEnable	Enables high baud rate selection.
	PLIB_USART_BaudRateHighSet	Sets the baud rate to the desired value.
	PLIB_USART_BaudRateModeSelect	Enables the 16-bit Baud Rate Generator (BRG).
	PLIB_USART_BaudRateSet	Sets the baud rate to the desired value.

Feature Existence Functions

	Name	Description
	<u>PLIB_USART_ExistsAlternateIO</u>	Identifies whether the AlternateIOEnableControl feature exists on the USART module.
	<u>PLIB_USART_ExistsBaudRate</u>	Identifies whether the BaudRate feature exists on the USART module.
	<u>PLIB_USART_ExistsBaudRateAutoDetect</u>	Identifies whether the BaudRateAutoDetect feature exists on the USART module.
	<u>PLIB_USART_ExistsBaudRateAutoDetectOverflow</u>	Identifies whether the BaudRateAutoDetectOverflow feature exists on the USART module.
	<u>PLIB_USART_ExistsBaudRateBitMode</u>	Identifies whether the BaudRateMode feature exists on the USART module.
	<u>PLIB_USART_ExistsBaudRateHigh</u>	Identifies whether the BaudRateHigh feature exists on the USART module.
	<u>PLIB_USART_ExistsEnable</u>	Identifies whether the EnableControl feature exists on the USART module.
	<u>PLIB_USART_ExistsHandshakeMode</u>	Identifies whether the HandShakeMode feature exists on the USART module.
	<u>PLIB_USART_ExistsIrDA</u>	Identifies whether the IrDAControl feature exists on the USART module.
	<u>PLIB_USART_ExistsLineControlMode</u>	Identifies whether the LineControlMode feature exists on the USART module.
	<u>PLIB_USART_ExistsLoopback</u>	Identifies whether the Loopback feature exists on the USART module.
	<u>PLIB_USART_ExistsOperationMode</u>	Identifies whether the OperationMode feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiver</u>	Identifies whether the Receiver feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverAddressAutoDetect</u>	Identifies whether the ReceiverAddressAutoDetect feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverAddressDetect</u>	Identifies whether the ReceiverAddressDetect feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverDataAvailableStatus</u>	Identifies whether the ReceiverDataAvailable feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverEnable</u>	Identifies whether the ReceiverEnableControl feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverFramingErrorStatus</u>	Identifies whether the ReceiverFramingError feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverIdleStateLowEnable</u>	Identifies whether the ReceiverPolarityInvert feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverIdleStatus</u>	Identifies whether the ReceiverIdle feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverInterruptMode</u>	Identifies whether the ReceiverInterruptMode feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverMode</u>	Identifies whether the ReceiverMode feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverOverrunStatus</u>	Identifies whether the ReceiverOverrunError feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverParityErrorStatus</u>	Identifies whether the ReceiverParityError feature exists on the USART module.

	PLIB_USART_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USART module.
	PLIB_USART_ExistsSyncClockPolarity	Identifies whether the SyncClockPolarity feature exists on the USART module.
	PLIB_USART_ExistsSyncClockSource	Identifies whether the SyncClockSource feature exists on the USART module.
	PLIB_USART_ExistsSyncMode	Identifies whether the SyncMode feature exists on the USART module.
	PLIB_USART_ExistsTransmitter	Identifies whether the Transmitter feature exists on the USART module.
	PLIB_USART_ExistsTransmitter9BitsSend	Identifies whether the Transmitter9Bits feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBreak	Identifies whether the TransmitterBreak feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBufferFullStatus	Identifies whether the TransmitterBufferFull feature exists on the USART module.
	PLIB_USART_ExistsTransmitterEmptyStatus	Identifies whether the TransmitterEmpty feature exists on the USART module.
	PLIB_USART_ExistsTransmitterEnable	Identifies whether the TransmitterEnableControl feature exists on the USART module.
	PLIB_USART_ExistsTransmitterIdleIsLow	Identifies whether the TransmitterIdleIsLow feature exists on the USART module.
	PLIB_USART_ExistsTransmitterIdleStatus	Identifies whether the TransmitterIdle feature exists on the USART module.
	PLIB_USART_ExistsTransmitterInterruptMode	Identifies whether the TransmitterInterruptMode feature exists on the USART module.
	PLIB_USART_ExistsWakeOnStart	Identifies whether the WakeOnStart feature exists on the USART module.

General Functions

	Name	Description
	PLIB_USART_AlternateIODisable	Disables the use of alternate I/O pins for the receiver and transmitter.
	PLIB_USART_AlternateIOEnable	Enables the use of alternate I/O pins for the receiver and transmitter.
	PLIB_USART_Disable	Disables the specific USART module.
	PLIB_USART_Enable	Enables the specific USART module.
	PLIB_USART_HandshakeModeSelect	Sets the data flow configuration.
	PLIB_USART_IrDADisable	Disables the IrDA encoder and decoder.
	PLIB_USART_IrDAEnable	Enables the IrDA encoder and decoder.
	PLIB_USART_LineControlModeSelect	Sets the data flow configuration.
	PLIB_USART_LoopbackDisable	Disables Loopback mode.
	PLIB_USART_LoopbackEnable	Enables Loopback mode.
	PLIB_USART_OperationModeSelect	Configures the operation mode of the USART module.
	PLIB_USART_StopInIdleDisable	Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).
	PLIB_USART_StopInIdleEnable	Discontinues operation when the device enters Idle mode.
	PLIB_USART_SyncClockPolarityIdleHighDisable	Sets the idle state of the clock to low.
	PLIB_USART_SyncClockPolarityIdleHighEnable	Sets the idle state of the clock to high.
	PLIB_USART_SyncClockSourceSelect	Sets the clock source.

	PLIB_USART_SyncModeSelect	Selects the USART mode to be asynchronous.
	PLIB_USART_WakeOnStartDisable	Disables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartEnable	Enables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartIsEnabled	Gets the state of the sync break event completion.

Receiver Functions

	Name	Description
	PLIB_USART_ReceiverAddressAutoDetectDisable	Disables the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressAutoDetectEnable	Setup the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressDetectDisable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressDetectEnable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressIsReceived	Checks and return if the data received is an address.
	PLIB_USART_ReceiverByteReceive	Data to be received in the Byte mode.
	PLIB_USART_ReceiverDataIsAvailable	Identifies if the receive data is available for the specified USART module.
	PLIB_USART_ReceiverDisable	Disables the USART receiver.
	PLIB_USART_ReceiverEnable	Enables the USART receiver.
	PLIB_USART_ReceiverFramingErrorHasOccurred	Gets the framing error status.
	PLIB_USART_ReceiverIdleStateLowDisable	Disables receive polarity inversion.
	PLIB_USART_ReceiverIdleStateLowEnable	Enables receive polarity inversion.
	PLIB_USART_ReceiverInterruptModeSelect	Sets the USART receiver FIFO level.
	PLIB_USART_ReceiverIsIdle	Identifies if the receiver is idle.
	PLIB_USART_ReceiverModeSelect	Disables Single Receive mode.
	PLIB_USART_ReceiverOverrunErrorClear	Clears a USART Overrun error.
	PLIB_USART_ReceiverOverrunHasOccurred	Identifies if there was a receiver overrun error.
	PLIB_USART_ReceiverParityErrorHasOccurred	Gets the parity error status.

Transmitter Functions

	Name	Description
	PLIB_USART_Transmitter9BitsSend	Data to be transmitted in the byte mode with the 9th bit.
	PLIB_USART_TransmitterBreakSend	Transmits the break character.
	PLIB_USART_TransmitterBreakSendIsComplete	Returns the status of the break transmission
	PLIB_USART_TransmitterBufferIsFull	Gets the transmit buffer full status.
	PLIB_USART_TransmitterByteSend	Data to be transmitted in the Byte mode.
	PLIB_USART_TransmitterDisable	Disables the specific USART module transmitter.
	PLIB_USART_TransmitterEnable	Enables the specific USART module transmitter.
	PLIB_USART_TransmitterIdleIsLowDisable	Disables the Transmit Idle Low state.
	PLIB_USART_TransmitterIdleIsLowEnable	Enables the Transmit Idle Low state.
	PLIB_USART_TransmitterInterruptModeSelect	Set the USART transmitter interrupt mode.
	PLIB_USART_TransmitterIsEmpty	Gets the transmit shift register empty status.
	PLIB_USART_TransmitterIsIdle	Gets the transmit buffer full status.

Description

This section describes the functions, data types, and constants available in the USART Peripheral Library.

5.6.25.5.1 General Functions

5.6.25.5.1.1 PLIB_USART_AlternateIODisable Function

C

```
void PLIB_USART_AlternateIODisable(  
    USART_MODULE_ID index  
);
```

Description

This function disables the USART module to use the alternate I/O pins for the receiver and transmitter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsAlternateIO](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1  
  
PLIB_USART_AlternateIODisable(MY_USART_INSTANCE);
```

5.6.25.5.1.2 PLIB_USART_AlternateIOEnable Function

C

```
void PLIB_USART_AlternateIOEnable(  
    USART_MODULE_ID index  
);
```

Description

This function enables the USART module to use the alternate I/O pins for the receiver and transmitter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_USART_ExistsAlternateIO](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_AlternateIOEnable(MY_USART_INSTANCE);
```

5.6.25.5.1.3 PLIB_USART_Disable Function**C**

```
void PLIB_USART_Disable(
    USART_MODULE_ID index
);
```

Description

This function disables the specific USART module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

The default state after any reset for a USART module is the disable state. If the USART is disabled, all of the related pins are in control of the general purpose I/O logic.

Disabling the USART module resets the buffers to empty states. Any data characters in the buffers are lost and the baud rate is reset. All error and status bits are also reset.

Disabling the USART while the USART is active, will abort all pending transmissions and receptions. Re-enabling the USART will restart the module in the same configuration.

When disabled, the USART power consumption is minimal.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_Disable(MY_USART_INSTANCE);
```

5.6.25.5.1.4 PLIB_USART_Enable Function**C**

```
void PLIB_USART_Enable(
    USART_MODULE_ID index
);
```

Description

This function enables the specific USART module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

By calling this function, the USART pins are controlled by the USART module.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_Enable(MY_USART_INSTANCE);
```

5.6.25.5.1.5 PLIB_USART_HandshakeModeSelect Function**C**

```
void PLIB_USART_HandshakeModeSelect(
    USART_MODULE_ID index,
    USART_HANDSHAKE_MODE handshakeConfig
);
```

Description

This function sets the USART data flow configuration based on the mask provided and the specified baud rate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	For possible data flow configurations, refer to USART_HANDSHAKE_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsHandshakeMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_HandshakeModeSelect(MY_USART_INSTANCE,
    USART_HANDSHAKE_MODE_SIMPLEX);
```

5.6.25.5.1.6 PLIB_USART_IrDADisable Function

C

```
void PLIB_USART_IrDADisable(  
    USART_MODULE_ID index  
);
```

Description

This function disables the IrDA encoder and decoder.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

By default, the IrDA Encoder and Decoder are disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsIrDA](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1  
  
PLIB_USART_IrDADisable(MY_USART_INSTANCE);
```

5.6.25.5.1.7 PLIB_USART_IrDAEnable Function

C

```
void PLIB_USART_IrDAEnable(  
    USART_MODULE_ID index  
);
```

Description

This function enables the IrDA encoder and decoder.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsIrDA](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_IrDAEnable(MY_USART_INSTANCE);
```

5.6.25.5.1.8 PLIB_USART_LineControlModeSelect Function**C**

```
void PLIB_USART_LineControlModeSelect(
    USART_MODULE_ID index,
    USART_LINECONTROL_MODE dataFlowConfig
);
```

Description

This function sets the USART data flow configuration based on the mask provided and the specified baud rate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	For possible data flow configurations, refer to USART_LINECONTROL_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsLineControlMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE,
    USART_8N1);
```

5.6.25.5.1.9 PLIB_USART_LoopbackDisable Function**C**

```
void PLIB_USART_LoopbackDisable(
    USART_MODULE_ID index
);
```

Description

This function disables Loopback mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

By default, Loopback mode is disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsLoopback](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_LoopbackDisable(MY_USART_INSTANCE);
```

5.6.25.5.1.10 PLIB_USART_LoopbackEnable Function**C**

```
void PLIB_USART_LoopbackEnable(
    USART_MODULE_ID index
);
```

Description

This function enables Loopback mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

By default, Loopback mode is disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsLoopback](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_LoopbackEnable(MY_USART_INSTANCE);
```

5.6.25.5.1.11 PLIB_USART_OperationModeSelect Function**C**

```
void PLIB_USART_OperationModeSelect(
    USART_MODULE_ID index,
    USART_OPERATION_MODE operationmode
);
```

Description

This function configures the operation mode of the USART (i.e., controls the pins used by the USART module). Refer to [USART_OPERATION_MODE](#) for the possible settings.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
operationmode	One of the possible values from USART_OPERATION_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsOperationMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_OperationModeSelect(MY_USART_INSTANCE, USART_ENABLE_TX_RX_BCLK_USED);
```

5.6.25.5.1.12 PLIB_USART_StopInIdleDisable Function**C**

```
void PLIB_USART_StopInIdleDisable(
    USART_MODULE_ID index
);
```

Description

This function disables the Stop in Idle mode. The USART module continues operation when the device is in Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

By default, the USART module will continue operation in idle mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_StopInIdleDisable(MY_USART_INSTANCE);
```

5.6.25.5.1.13 PLIB_USART_StopInIdleEnable Function**C**

```
void PLIB_USART_StopInIdleEnable(
```

```
    USART_MODULE_ID index
);
```

Description

This function enables the USART module to discontinue operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_StopInIdleEnable(MY_USART_INSTANCE);
```

5.6.25.5.1.14 PLIB_USART_SyncClockPolarityIdleHighDisable Function**C**

```
void PLIB_USART_SyncClockPolarityIdleHighDisable(
    USART_MODULE_ID index
);
```

Description

This function sets the clock idle in the synchronous mode to low.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Calling this function disables the clock polarity idle as high.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsSyncClockPolarity](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_SyncClockPolarityIdleHighDisable(MY_USART_INSTANCE);
```


5.6.25.5.1.15 PLIB_USART_SyncClockPolarityIdleHighEnable Function

C

```
void PLIB_USART_SyncClockPolarityIdleHighEnable(  
    USART_MODULE_ID index  
);
```

Description

This function sets the clock idle in the Synchronous mode to high.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Calling this function disables the clock polarity idle as low.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsSyncClockPolarity](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1  
  
PLIB_USART_SyncClockPolarityIdleHighEnable(MY_USART_INSTANCE);
```

5.6.25.5.1.16 PLIB_USART_SyncClockSourceSelect Function

C

```
void PLIB_USART_SyncClockSourceSelect(  
    USART_MODULE_ID index,  
    USART_SYNC_CLOCK_SOURCE source  
);
```

Description

This function sets the clock source to either master or slave, which is only valid in Synchronous mode. The clock is generated from the baud rate generator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Calling this function disables the use of the slave as the clock source.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsSyncClockSource](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_MASTER);
```

5.6.25.5.1.17 PLIB_USART_SyncModeSelect Function**C**

```
void PLIB_USART_SyncModeSelect(
    USART_MODULE_ID index,
    USART_SYNC_MODES mode
);
```

Description

This function enables the Asynchronous mode of the USART module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Calling this function disables the Synchronous mode of the USART module. By default, the module is in Asynchronous mode after a reset.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsSyncMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
```

5.6.25.5.1.18 PLIB_USART_WakeOnStartDisable Function**C**

```
void PLIB_USART_WakeOnStartDisable(
    USART_MODULE_ID index
);
```

Description

This function disables the wake-up on start bit detection feature during Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsWakeOnStart](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_WakeOnStartDisable(MY_USART_INSTANCE);
```

5.6.25.5.1.19 PLIB_USART_WakeOnStartEnable Function**C**

```
void PLIB_USART_WakeOnStartEnable(
    USART_MODULE_ID index
);
```

Description

This function enables the wake-up on start feature during Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsWakeOnStart](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_WakeOnStartEnable(MY_USART_INSTANCE);
```

5.6.25.5.1.20 PLIB_USART_WakeOnStartIsEnabled Function**C**

```
bool PLIB_USART_WakeOnStartIsEnabled(
    USART_MODULE_ID index
);
```

Description

This function returns the status of the sync break event, when called after enabling using [PLIB_USART_WakeOnStartEnable](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsWakeOnStart](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

//Call the interface just prior to entering the sleep mode.
PLIB_USART_WakeOnStartEnable(MY_USART_INSTANCE);

// Check the status if the Sync break event is over.
if(PLIB_USART_WakeOnStartIsEnabled(MY_USART_INSTANCE))
{
    // Do Something
}
```

5.6.25.5.2 Baud Rate Generator Functions

5.6.25.5.2.1 PLIB_USART_BaudRateAutoDetectEnable Function

C

```
void PLIB_USART_BaudRateAutoDetectEnable(
    USART_MODULE_ID index
);
```

Description

This function enables the baud rate measurement on the next character, which requires reception of the Sync character.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsBaudRateAutoDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1
```

```
PLIB_USART_BaudRateAutoDetectEnable(MY_USART_INSTANCE);  
  
// Wait till the baud rate is detected.  
while(!PLIB_USART_BaudRateAutoDetectIsComplete(MY_USART_INSTANCE));
```

5.6.25.5.2.2 PLIB_USART_BaudRateAutoDetectIsComplete Function

C

```
bool PLIB_USART_BaudRateAutoDetectIsComplete(  
    USART_MODULE_ID index  
);
```

Description

This function gets the state of the automatic baud detection and returns a '0' if the baud rate auto detection is complete.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Baud rate detection is not complete
- false - Baud rate detection is complete

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsBaudRateAutoDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1  
  
PLIB_USART_BaudRateAutoDetectEnable(MY_USART_INSTANCE);  
  
// Wait till the baud rate is detected.  
while(!PLIB_USART_BaudRateAutoDetectIsComplete(MY_USART_INSTANCE));
```

5.6.25.5.2.3 PLIB_USART_BaudRateAutoDetectOverflowHasOccurred Function

C

```
bool PLIB_USART_BaudRateAutoDetectOverflowHasOccurred(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether or not the automatic baud timer has overflowed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - An auto-baud overflow has occurred
- false - An auto-baud overflow has not occurred

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsBaudRateAutoDetectOverflow](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

status = PLIB_USART_BaudRateAutoDetectOverflowHasOccurred(MY_USART_INSTANCE);
```

5.6.25.5.2.4 PLIB_USART_BaudRateGet Function**C**

```
uint32_t PLIB_USART_BaudRateGet(
    USART_MODULE_ID index,
    int32_t clockFrequency
);
```

Description

This function gets the baud rate that is currently in use. The clock frequency needs to be passed to the function.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clockFrequency	Clock Frequency

Returns

BaudRate - Baud Rate Value

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsBaudRate](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRate ;
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
baudRate = PLIB_USART_BaudRateGet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY);
```

5.6.25.5.2.5 PLIB_USART_BaudRateHighDisable Function**C**

```
void PLIB_USART_BaudRateHighDisable(
    USART_MODULE_ID index
);
```

Description

This function disables the high baud rate selection.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsBaudRateHigh](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateHighDisable(MY_USART_INSTANCE);
```

5.6.25.5.2.6 PLIB_USART_BaudRateHighEnable Function**C**

```
void PLIB_USART_BaudRateHighEnable(
    USART_MODULE_ID index
);
```

Description

This function enables high baud rate selection.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsBaudRateHigh](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateHighEnable(MY_USART_INSTANCE);
```

5.6.25.5.2.7 PLIB_USART_BaudRateHighSet Function**C**

```
void PLIB_USART_BaudRateHighSet(
    USART_MODULE_ID index,
    uint32_t clockFrequency,
    uint32_t baudRate
);
```

```
);
```

Description

This function sets the baud rate to the desired value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baudRate	Baud Rate Value, it is the baud rate value
clockFrequency	Clock Frequency

Returns

None.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue verify that no receptions are in progress before changing the system clock.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsBaudRateHigh](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRateValue ;
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
baudRateValue = PLIB_USART_BaudRateGet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY);
```

5.6.25.5.2.8 PLIB_USART_BaudRateModeSelect Function**C**

```
void PLIB_USART_BaudRateModeSelect(
    USART_MODULE_ID index,
    USART_BAUD_RATE_MODE mode
);
```

Description

This function enables the 16-bit Baud Rate Generator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Baud rate mode

Returns

None.

Remarks

Calling this function disables the 8-bit Baud Rate Generator. The 16-bit mode is used to achieve a slow baud rate for fast oscillator frequencies.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsBaudRateBitMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateModeSelect(MY_USART_INSTANCE, USART_BAUD_RATE_16_BIT_MODE);
```

5.6.25.5.2.9 PLIB_USART_BaudRateSet Function**C**

```
void PLIB_USART_BaudRateSet(
    USART_MODULE_ID index,
    uint32_t clockFrequency,
    uint32_t baudRate
);
```

Description

This function sets the baud rate to the desired value.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baudRate	Baud Rate Value
clockFrequency	Clock Frequency

Returns

None.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue verify that no receptions are in progress before changing the system clock.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsBaudRate](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRateValue ;
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
baudRateValue = PLIB_USART_BaudRateGet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY);
```

5.6.25.5.3 Transmitter Functions

5.6.25.5.3.1 PLIB_USART_Transmitter9BitsSend Function

C

```
void PLIB_USART_Transmitter9BitsSend(  
    USART_MODULE_ID index,  
    int8_t data,  
    bool Bit9th  
);
```

Description

The data is transmitted in the byte mode for the specified USART module, with 9th bit.

Preconditions

The USART module should be configured to use the 9 data bits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be transmitted.
9thBit	9th bit of the data to be transmitted.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitter9BitsSend](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1  
  
uint8_t data = 'a';  
  
if(!PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE))  
{  
    PLIB_USART_Transmitter9BitsSend(MY_USART_INSTANCE, data, false);  
}
```

5.6.25.5.3.2 PLIB_USART_TransmitterBreakSend Function

C

```
void PLIB_USART_TransmitterBreakSend(  
    USART_MODULE_ID index  
);
```

Description

This function transmits the break character.

Preconditions

The application should wait for the transmitter to be idle, before calling this function.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

After the break has been transmitted, the application can start transmitting next bytes.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterBreak](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);

// wait for the break transmission to complete
while(!PLIB_USART_TransmitterBreakSendIsComplete(MY_USART_INSTANCE));
```

5.6.25.5.3.3 PLIB_USART_TransmitterBreakSendIsComplete Function**C**

```
bool PLIB_USART_TransmitterBreakSendIsComplete(
    USART_MODULE_ID index
);
```

Description

The function returns the status of the break transmission.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - Transmit Break on the next transmission
- false - Break Transmission completed or not started

Remarks

After the break has been transmitted, the application can start transmitting next bytes.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterBreak](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);

// wait for the break transmission to complete
```

```
while(!PLIB_USART_TransmitterBreakSendIsComplete(MY_USART_INSTANCE));
```

5.6.25.5.3.4 PLIB_USART_TransmitterBufferIsFull Function

C

```
bool PLIB_USART_TransmitterBufferIsFull(  
    USART_MODULE_ID index  
);
```

Description

Gets the transmit status of the specified USART module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The transmit buffer is full
- false - The transmit buffer is not full, at least one more character can be written.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterBufferFullStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1  
  
// Wait for the Transmit buffer to be empty.  
while(PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE));  
  
// Transmit the break character.  
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);
```

5.6.25.5.3.5 PLIB_USART_TransmitterByteSend Function

C

```
void PLIB_USART_TransmitterByteSend(  
    USART_MODULE_ID index,  
    int8_t data  
);
```

Description

The data is transmitted in the Byte mode for the specified USART module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be transmitted.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitter](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint8_t data = 'a';
if(!PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE))
{
    PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, data);
}
```

5.6.25.5.3.6 PLIB_USART_TransmitterDisable Function**C**

```
void PLIB_USART_TransmitterDisable(
    USART_MODULE_ID index
);
```

Description

This function disables the specific USART module transmitter.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

Disabling the transmitter during a transmission will cause the transmission to be aborted.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterDisable(MY_USART_INSTANCE);
```

5.6.25.5.3.7 PLIB_USART_TransmitterEnable Function**C**

```
void PLIB_USART_TransmitterEnable(
    USART_MODULE_ID index
);
```

Description

This function enables the specific USART module transmitter.

Preconditions

The USART module should be enabled using the function [PLIB_USART_Enable](#) before this function is called.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

The transmitter should not be enabled until the USART is enabled. The transmissions will not be enabled otherwise.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);
```

5.6.25.5.3.8 PLIB_USART_TransmitterIdleIsLowDisable Function**C**

```
void PLIB_USART_TransmitterIdleIsLowDisable(
    USART_MODULE_ID index
);
```

Description

This function disables the Transmit Idle Low state. In USART Synchronous mode, this function configures that the TX polarity the idle state is high.

When IrDA is enabled, this function sets the IrDA encoded Transmit Idle state to a '0'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterIdleIsLow](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterIdleIsLowDisable(MY_USART_INSTANCE);
```

5.6.25.5.3.9 PLIB_USART_TransmitterIdleIsLowEnable Function

C

```
void PLIB_USART_TransmitterIdleIsLowEnable(  
    USART_MODULE_ID index  
);
```

Description

This function enables the Transmit Idle Low state. In the USART Synchronous mode, this function configures that the TX polarity, the idle state is low.

When IrDA(c) is enabled, this function sets that IrDA encoded Transmit Idle state to a '1'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterIdleIsLow](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1  
  
PLIB_USART_TransmitterIdleIsLowEnable(MY_USART_INSTANCE);
```

5.6.25.5.3.10 PLIB_USART_TransmitterInterruptModeSelect Function

C

```
void PLIB_USART_TransmitterInterruptModeSelect(  
    USART_MODULE_ID index,  
    USART_TRANSMIT_INTR_MODE fifolevel  
);
```

Description

This function sets the condition in which the USART module should generate an interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptMode	Interrupt mode; for possible configurations, refer to USART_TRANSMIT_INTR_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterInterruptMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterInterruptModeSelect(MY_USART_INSTANCE,
    USART_TRANSMIT_FIFO_EMPTY );
```

5.6.25.5.3.11 PLIB_USART_TransmitterIsEmpty Function**C**

```
bool PLIB_USART_TransmitterIsEmpty(
    USART_MODULE_ID index
);
```

Description

This function gets the transmit shift register empty status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The transmit shift register is empty
- false - The transmit shift register is not empty

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterEmptyStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while( !PLIB_USART_TransmitterIsEmpty(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);
```

5.6.25.5.3.12 PLIB_USART_TransmitterIsIdle Function**C**

```
bool PLIB_USART_TransmitterIsIdle(
    USART_MODULE_ID index
);
```

Description

This function gets the transmit buffer full status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The transmitter is idle
- false - The transmitter is not idle

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsTransmitterIdleStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

status = PLIB_USART_TransmitterIsIdle(MY_USART_INSTANCE);
```

5.6.25.5.4 Receiver Functions

5.6.25.5.4.1 PLIB_USART_ReceiverAddressAutoDetectDisable Function

C

```
void PLIB_USART_ReceiverAddressAutoDetectDisable(
    USART_MODULE_ID index
);
```

Description

This function disables the automatic Address Detect mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverAddressAutoDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressAutoDetectDisable(MY_USART_INSTANCE);
```

5.6.25.5.4.2 PLIB_USART_ReceiverAddressAutoDetectEnable Function

C

```
void PLIB_USART_ReceiverAddressAutoDetectEnable(
    USART_MODULE_ID index,
    int8_t Mask
);
```

```
);
```

Description

This function configures the automatic Address Detect mode. Uses the mask as the address character for automatic address detection.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
Mask	Address character to be used, when enabled

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverAddressAutoDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressAutoDetectEnable(MY_USART_INSTANCE,
                                             MY_DEVICE_ADDRESS);
```

5.6.25.5.4.3 PLIB_USART_ReceiverAddressDetectDisable Function**C**

```
void PLIB_USART_ReceiverAddressDetectDisable(
    USART_MODULE_ID index
);
```

Description

This function disables the Address Detect mode. If it is enabled, and the device is addressed, disable the Address Detect mode to continue receiving bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

All bytes are received, and the 9th bit can be used as the parity bit. By default, the address detect is disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverAddressDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1
```

```
PLIB_USART_ReceiverAddressDetectDisable(MY_USART_INSTANCE);
```

5.6.25.5.4.4 PLIB_USART_ReceiverAddressDetectEnable Function

C

```
void PLIB_USART_ReceiverAddressDetectEnable(  
    USART_MODULE_ID index  
);
```

Description

This function enables the Address Detect mode. If it is enabled, and the device is addressed, disable the Address Detect mode to continue receiving bytes.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

If 9 data bits are not selected, this bit has no effect.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverAddressDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1  
  
PLIB_USART_ReceiverAddressDetectEnable(MY_USART_INSTANCE);
```

5.6.25.5.4.5 PLIB_USART_ReceiverAddressIsReceived Function

C

```
bool PLIB_USART_ReceiverAddressIsReceived(  
    USART_MODULE_ID index  
);
```

Description

Checks and return if the data received is an address. The address has the 9th bit set. If data received, has 9th bit set, function returns true, else returns false.

Preconditions

The USART module should be configured to use the 9 data bits.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - if the data received has the 9th bit set
- false - if the address received has the 9th bit cleared

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverAddressDetect](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

int8_t address;

if(PLIB_USART_ReceiverAddressIsReceived(MY_USART_INSTANCE))
{
    address = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}
```

5.6.25.5.4.6 PLIB_USART_ReceiverByteReceive Function**C**

```
int8_t PLIB_USART_ReceiverByteReceive(
    USART_MODULE_ID index
);
```

Description

The data to be received in Byte mode from the specified USART module. Call the functions [PLIB_USART_ReceiverFramingErrorHasOccurred](#), [PLIB_USART_ReceiverParityErrorHasOccurred](#) and [PLIB_USART_ReceiverOverrunHasOccurred](#) to get any error that occurred.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

data - Data to be received.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiver](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

bool isError;
uint8_t mydata;

if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}

isError = PLIB_USART_ReceiverFramingErrorHasOccurred(MY_USART_INSTANCE) |
          PLIB_USART_ReceiverParityErrorHasOccurred(MY_USART_INSTANCE) |
          PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE);

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

```
}
```

5.6.25.5.4.7 PLIB_USART_ReceiverDataIsAvailable Function

C

```
bool PLIB_USART_ReceiverDataIsAvailable(  
    USART_MODULE_ID index  
);
```

Description

This function identifies if the receive data is available for the specified USART module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The data is available
- false - The data is not available

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverDataAvailableStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1  
  
int8_t mydata;  
  
if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))  
{  
    mydata = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);  
}
```

5.6.25.5.4.8 PLIB_USART_ReceiverDisable Function

C

```
void PLIB_USART_ReceiverDisable(  
    USART_MODULE_ID index  
);
```

Description

This function disables the USART receiver.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverDisable(MY_USART_INSTANCE);
```

5.6.25.5.4.9 PLIB_USART_ReceiverEnable Function**C**

```
void PLIB_USART_ReceiverEnable(
    USART_MODULE_ID index
);
```

Description

This function enables the USART receiver.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

5.6.25.5.4.10 PLIB_USART_ReceiverFramingErrorHasOccurred Function**C**

```
bool PLIB_USART_ReceiverFramingErrorHasOccurred(
    USART_MODULE_ID index
);
```

Description

This function gets the framing error status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The framing error was detected on the current character
- false - The framing error was not detected on the current character

Remarks

Reading the error clears the error.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverFramingErrorStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

status = PLIB_USART_ReceiverFramingErrorHasOccurred(MY_USART_INSTANCE);
```

5.6.25.5.4.11 PLIB_USART_ReceiverIdleStateLowDisable Function**C**

```
void PLIB_USART_ReceiverIdleStateLowDisable(
    USART_MODULE_ID index
);
```

Description

This function disables receive polarity inversion. In the USART Synchronous mode, this function configures that the data is not inverted.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverIdleStateLowEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverIdleStateLowDisable(MY_USART_INSTANCE);
```

5.6.25.5.4.12 PLIB_USART_ReceiverIdleStateLowEnable Function**C**

```
void PLIB_USART_ReceiverIdleStateLowEnable(
    USART_MODULE_ID index
);
```

Description

This function enables receive polarity inversion. In the USART Synchronous mode, this function configures that the data is inverted.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverIdleStateLowEnable](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverIdleStateLowEnable(MY_USART_INSTANCE);
```

5.6.25.5.4.13 PLIB_USART_ReceiverInterruptModeSelect Function**C**

```
void PLIB_USART_ReceiverInterruptModeSelect(
    USART_MODULE_ID index,
    USART_RECEIVE_INTR_MODE interruptMode
);
```

Description

This function sets the USART receiver FIFO level.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
fifolevel	For possible configurations, refer to USART_RECEIVE_INTR_MODE

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverInterruptMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverInterruptModeSelect(MY_USART_INSTANCE,
    USART_RECEIVE_FIFO_ONE_CHAR );
```

5.6.25.5.4.14 PLIB_USART_ReceiverIsIdle Function**C**

```
bool PLIB_USART_ReceiverIsIdle(
```



```
    USART_MODULE_ID index
);
```

Description

This function identifies if the receiver is idle.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The receive buffer is idle
- false - The receive buffer is not idle

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverIdleStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

int8_t mydata;

if(PLIB_USART_ReceiverIsIdle(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}
```

5.6.25.5.4.15 PLIB_USART_ReceiverModeSelect Function**C**

```
void PLIB_USART_ReceiverModeSelect(
    USART_MODULE_ID index,
    USART_RECEIVE_MODES mode
);
```

Description

This function disables Single Receive mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This function has no effect during Asynchronous mode, or in the Synchronous Slave mode. By default, the single receive is disabled after a reset.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_USART_ExistsReceiverMode](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverModeSelect(MY_USART_INSTANCE, USART_RECEIVE_MODE_SINGLE);
```

5.6.25.5.4.16 PLIB_USART_ReceiverOverrunErrorClear Function**C**

```
void PLIB_USART_ReceiverOverrunErrorClear(
    USART_MODULE_ID index
);
```

Description

Clears an overrun error. Clearing the error, resets the receive buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

WARNING: Calling this API will clear all the previously received data.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverOverrunStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

5.6.25.5.4.17 PLIB_USART_ReceiverOverrunHasOccurred Function**C**

```
bool PLIB_USART_ReceiverOverrunHasOccurred(
    USART_MODULE_ID index
);
```

Description

This function identifies if there was a receiver overrun error.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The receive buffer has overflowed
- false - The receive buffer has not overflowed

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverOverrunStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

5.6.25.5.4.18 PLIB_USART_ReceiverParityErrorHasOccurred Function**C**

```
bool PLIB_USART_ReceiverParityErrorHasOccurred(
    USART_MODULE_ID index
);
```

Description

This function gets the parity error status.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - The parity error was detected on the current character
- false - The parity error was not detected on the current character

Remarks

Reading the error clears the error.

A Parity error is irrelevant in case of 9-bit mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USART_ExistsReceiverParityErrorStatus](#) in your application to determine whether this feature is available.

Example

```
#define MY_USART_INSTANCE USART_ID_1

status = PLIB_USART_ReceiverParityErrorHasOccurred(MY_USART_INSTANCE);
```

5.6.25.5.5 Data Types and Constants

5.6.25.5.5.1 USART_BAUD_RATE_MODE Enumeration

C

```
typedef enum {  
    USART_BAUD_RATE_8_BIT_MODE,  
    USART_BAUD_RATE_16_BIT_MODE  
} USART_BAUD_RATE_MODE;
```

Description

/* USART Baud rate mode

This enumeration lists the USART baud rate modes.

Members

Members	Description
USART_BAUD_RATE_8_BIT_MODE	Baud rate register is 8 bit
USART_BAUD_RATE_16_BIT_MODE	Baud rate register is 16 bit

Remarks

None.

5.6.25.5.5.2 USART_HANDSHAKE_MODE Enumeration

C

```
typedef enum {  
    USART_HANDSHAKE_MODE_FLOW_CONTROL,  
    USART_HANDSHAKE_MODE_SIMPLEX  
} USART_HANDSHAKE_MODE;
```

Description

/* USART Handshake modes

This enumeration lists the USART handshake modes.

Members

Members	Description
USART_HANDSHAKE_MODE_FLOW_CONTROL	Enables flow control
USART_HANDSHAKE_MODE_SIMPLEX	Enables simplex mode communication, no flow control

Remarks

None.

5.6.25.5.5.3 USART_LINECONTROL_MODE Enumeration

C

```
typedef enum {  
    USART_8N1,  
    USART_8E1,  
    USART_8O1,  
    USART_8N2,  
    USART_8E2,  
    USART_8O2,  
    USART_9N1,  
    USART_9N2  
} USART_LINECONTROL_MODE;
```

Description

Data Flow configuration

This data type defines the different configurations by which the USART can be configured for the data flow.

Members

Members	Description
USART_8N1	8 Data Bits, No Parity,one Stop Bit
USART_8E1	8 Data Bits, Even Parity, 1 stop bit
USART_8O1	8 Data Bits, odd Parity, 1 stop bit
USART_8N2	8 Data Bits, No Parity,two Stop Bits
USART_8E2	8 Data Bits, Even Parity, 2 stop bits
USART_8O2	8 Data Bits, odd Parity, 2 stop bits
USART_9N1	9 Data Bits, No Parity, 1 stop bit
USART_9N2	9 Data Bits, No Parity, 2 stop bits

Remarks

None.

5.6.25.5.5.4 USART_MODULE_ID Enumeration**C**

```
typedef enum {
    USART_ID_1,
    USART_ID_2,
    USART_ID_3,
    USART_ID_4,
    USART_ID_5,
    USART_ID_6,
    USART_NUMBER_OF_MODULES
} USART_MODULE_ID;
```

Description

Enumeration: USART_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers.

Refer to the data sheet to get the correct number of modules defined for desired microcontroller.

Members

Members	Description
USART_ID_1	USART 1
USART_ID_2	USART 2
USART_ID_3	USART 3
USART_ID_4	USART 4
USART_ID_5	USART 5
USART_ID_6	USART 6
USART_NUMBER_OF_MODULES	Total number of USART modules available

5.6.25.5.5.5 USART_OPERATION_MODE Enumeration

C

```
typedef enum {
    USART_ENABLE_TX_RX_BCLK_USED,
    USART_ENABLE_TX_RX_CTS_RTS_USED,
    USART_ENABLE_TX_RX_RTS_USED,
    USART_ENABLE_TX_RX_USED
} USART_OPERATION_MODE;
```

Description

Enable configuration

This data type defines the different configurations by which the USART can be enabled.

Members

Members	Description
USART_ENABLE_TX_RX_BCLK_USED	TX, RX and BCLK pins are used by USART module
USART_ENABLE_TX_RX_CTS_RTS_USED	TX, RX, CTS and RTS pins are used by USART module
USART_ENABLE_TX_RX_RTS_USED	TX, RX and RTS pins are used by USART module
USART_ENABLE_TX_RX_USED	TX and RX pins are used by USART module

Remarks

The actual definition of this enumeration is device-specific.

5.6.25.5.5.6 USART_RECEIVE_INTR_MODE Enumeration

C

```
typedef enum {
    USART_RECEIVE_FIFO_HALF_FULL,
    USART_RECEIVE_FIFO_3B4FULL,
    USART_RECEIVE_FIFO_ONE_CHAR
} USART_RECEIVE_INTR_MODE;
```

Description

Receive Interrupt mode configuration

This data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.

Members

Members	Description
USART_RECEIVE_FIFO_HALF_FULL	Interrupt when receive buffer is half full
USART_RECEIVE_FIFO_3B4FULL	Interrupt when receive buffer is 3/4 full
USART_RECEIVE_FIFO_ONE_CHAR	Interrupt when a character is received

Remarks

None.

5.6.25.5.5.7 USART_RECEIVE_MODES Enumeration

C

```
typedef enum {
    USART_RECEIVE_MODE_CONTINUOUS,
    USART_RECEIVE_MODE_SINGLE,
```

```
    USART_RECEIVE_MODES_NONE
} USART_RECEIVE_MODES;
```

Description

/* USART Receive modes

This enumeration lists the USART receive modes.

Members

Members	Description
USART_RECEIVE_MODE_CONTINUOUS	Continuous Receive mode
USART_RECEIVE_MODE_SINGLE	Single Receive mode
USART_RECEIVE_MODES_NONE	No Receive mode

Remarks

This enumeration may not exist for all devices.

5.6.25.5.5.8 USART_SYNC_CLOCK_SOURCE Enumeration**C**

```
typedef enum {
    USART_SYNC_CLOCK_SOURCE_MASTER,
    USART_SYNC_CLOCK_SOURCE_SLAVE
} USART_SYNC_CLOCK_SOURCE;
```

Description

/* USART Clock sources

This enumeration lists the USART clock sources.

Members

Members	Description
USART_SYNC_CLOCK_SOURCE_MASTER	USART clock source is from master
USART_SYNC_CLOCK_SOURCE_SLAVE	USART clock source is from slave

Remarks

None.

5.6.25.5.5.9 USART_SYNC_MODES Enumeration**C**

```
typedef enum {
    USART_ASYNC_MODE,
    USART_SYNC_MODE
} USART_SYNC_MODES;
```

Description

/* USART Synchronisation Modes

Lists enumerations selecting asynchronous communication or synchronous communication.

Members

Members	Description
USART_ASYNC_MODE	Normal UART. Asynchronous
USART_SYNC_MODE	Synchronous

Remarks

This enumeration is not used for devices which does not have the synchronous communication option (i.e., always asynchronous).

5.6.25.5.10 USART_TRANSMIT_INTR_MODE Enumeration**C**

```
typedef enum {
    USART_TRANSMIT_FIFO_EMPTY,
    USART_TRANSMIT_FIFO_FULL,
    USART_TRANSMIT_FIFO_IDLE,
    USART_TRANSMIT_FIFO_NOT_FULL
} USART_TRANSMIT_INTR_MODE;
```

Description

/* Transmit Interrupt mode configuration

This data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

Members

Members	Description
USART_TRANSMIT_FIFO_EMPTY	Interrupt when the transmit buffer becomes empty
USART_TRANSMIT_FIFO_FULL	Interrupt when the last transmission is over and all transmit operations are completed
USART_TRANSMIT_FIFO_IDLE	Interrupt when all characters are transmitted
USART_TRANSMIT_FIFO_NOT_FULL	Interrupt when atleast one location is empty in the transmit buffer

Remarks

None.

5.6.25.5.6 Feature Existence Functions**5.6.25.5.6.1 PLIB_USART_ExistsAlternateIO Function****C**

```
bool PLIB_USART_ExistsAlternateIO(
    USART_MODULE_ID index
);
```

Description

This function identifies whether the AlternateIOEnableControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_AlternateIOEnable](#)
- [PLIB_USART_AlternateIODisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AlternateIOEnableControl feature is supported on the device
- false - The AlternateIOEnableControl feature is not supported on the device

Remarks

None.

5.6.25.5.6.2 PLIB_USART_ExistsBaudRate Function**C**

```
bool PLIB_USART_ExistsBaudRate(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the BaudRate feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_BaudRateSet](#)
- [PLIB_USART_BaudRateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BaudRate feature is supported on the device
- false - The BaudRate feature is not supported on the device

Remarks

None.

5.6.25.5.6.3 PLIB_USART_ExistsBaudRateAutoDetect Function**C**

```
bool PLIB_USART_ExistsBaudRateAutoDetect(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the BaudRateAutoDetect feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_BaudRateAutoDetectEnable](#)
- [PLIB_USART_BaudRateAutoDetectIsComplete](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BaudRateAutoDetect feature is supported on the device
- false - The BaudRateAutoDetect feature is not supported on the device

Remarks

None.

5.6.25.5.6.4 PLIB_USART_ExistsBaudRateAutoDetectOverflow Function**C**

```
bool PLIB_USART_ExistsBaudRateAutoDetectOverflow(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the BaudRateAutoDetectOverflow feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_BaudRateAutoDetectOverflowHasOccurred](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BaudRateAutoDetectOverflow feature is supported on the device
- false - The BaudRateAutoDetectOverflow feature is not supported on the device

Remarks

None.

5.6.25.5.6.5 PLIB_USART_ExistsBaudRateBitMode Function**C**

```
bool PLIB_USART_ExistsBaudRateBitMode(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the BaudRateMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_BaudRateModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BaudRateMode feature is supported on the device
- false - The BaudRateMode feature is not supported on the device

Remarks

None.

5.6.25.5.6.6 PLIB_USART_ExistsBaudRateHigh Function**C**

```
bool PLIB_USART_ExistsBaudRateHigh(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the BaudRateHigh feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_BaudRateHighSet](#)
- [PLIB_USART_BaudRateHighDisable](#)
- [PLIB_USART_BaudRateHighEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BaudRateHigh feature is supported on the device
- false - The BaudRateHigh feature is not supported on the device

Remarks

None.

5.6.25.5.6.7 PLIB_USART_ExistsEnable Function**C**

```
bool PLIB_USART_ExistsEnable(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_Disable](#)

- [PLIB_USART_Enable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Remarks

None.

5.6.25.5.6.8 PLIB_USART_ExistsHandshakeMode Function

C

```
bool PLIB_USART_ExistsHandshakeMode(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the HandShakeMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_HandshakeModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The HandShakeMode feature is supported on the device
- false - The HandShakeMode feature is not supported on the device

Remarks

None.

5.6.25.5.6.9 PLIB_USART_ExistsIrDA Function

C

```
bool PLIB_USART_ExistsIrDA(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the IrDAControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_IrDADisable](#)

- [PLIB_USART_IrDAEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The IrDAControl feature is supported on the device
- false - The IrDAControl feature is not supported on the device

Remarks

None.

5.6.25.5.6.10 PLIB_USART_ExistsLineControlMode Function

C

```
bool PLIB_USART_ExistsLineControlMode(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the LineControlMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_LineControlModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LineControlMode feature is supported on the device
- false - The LineControlMode feature is not supported on the device

Remarks

None.

5.6.25.5.6.11 PLIB_USART_ExistsLoopback Function

C

```
bool PLIB_USART_ExistsLoopback(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the Loopback feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_LoopbackEnable](#)

- [PLIB_USART_LoopbackDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Loopback feature is supported on the device
- false - The Loopback feature is not supported on the device

Remarks

None.

5.6.25.5.6.12 PLIB_USART_ExistsOperationMode Function**C**

```
bool PLIB_USART_ExistsOperationMode(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the OperationMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_OperationModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OperationMode feature is supported on the device
- false - The OperationMode feature is not supported on the device

Remarks

None.

5.6.25.5.6.13 PLIB_USART_ExistsReceiver Function**C**

```
bool PLIB_USART_ExistsReceiver(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the Receiver feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_ReceiverByteReceive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Receiver feature is supported on the device
- false - The Receiver feature is not supported on the device

Remarks

None.

5.6.25.5.6.14 PLIB_USART_ExistsReceiverAddressAutoDetect Function**C**

```
bool PLIB_USART_ExistsReceiverAddressAutoDetect(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiverAddressAutoDetect feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_ReceiverAddressAutoDetectEnable](#)
- [PLIB_USART_ReceiverAddressAutoDetectDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverAddressAutoDetect feature is supported on the device
- false - The ReceiverAddressAutoDetect feature is not supported on the device

Remarks

None.

5.6.25.5.6.15 PLIB_USART_ExistsReceiverAddressDetect Function**C**

```
bool PLIB_USART_ExistsReceiverAddressDetect(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiverAddressDetect feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_ReceiverAddressDetectEnable](#)
- [PLIB_USART_ReceiverAddressDetectDisable](#)
- [PLIB_USART_ReceiverAddressIsReceived](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverAddressDetect feature is supported on the device
- false - The ReceiverAddressDetect feature is not supported on the device

Remarks

None.

5.6.25.5.6.16 PLIB_USART_ExistsReceiverDataAvailableStatus Function**C**

```
bool PLIB_USART_ExistsReceiverDataAvailableStatus(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiverDataAvailable feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_ReceiverDataIsAvailable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverDataAvailable feature is supported on the device
- false - The ReceiverDataAvailable feature is not supported on the device

Remarks

None.

5.6.25.5.6.17 PLIB_USART_ExistsReceiverEnable Function**C**

```
bool PLIB_USART_ExistsReceiverEnable(  
    USART_MODULE_ID index  
);
```


Description

This function identifies whether the ReceiverEnableControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_ReceiverEnable](#)
- [PLIB_USART_ReceiverDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverEnableControl feature is supported on the device
- false - The ReceiverEnableControl feature is not supported on the device

Remarks

None.

5.6.25.5.6.18 PLIB_USART_ExistsReceiverFramingErrorStatus Function**C**

```
bool PLIB_USART_ExistsReceiverFramingErrorStatus(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiverFramingError feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_ReceiverFramingErrorHasOccurred](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverFramingError feature is supported on the device
- false - The ReceiverFramingError feature is not supported on the device

Remarks

None.

5.6.25.5.6.19 PLIB_USART_ExistsReceiverIdleStateLowEnable Function**C**

```
bool PLIB_USART_ExistsReceiverIdleStateLowEnable(  
    USART_MODULE_ID index
```

```
);
```

Description

This function identifies whether the ReceiverPolarityInvert feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_ReceiverIdleStateLowEnable](#)
- [PLIB_USART_ReceiverIdleStateLowDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverPolarityInvert feature is supported on the device
- false - The ReceiverPolarityInvert feature is not supported on the device

Remarks

None.

5.6.25.5.6.20 PLIB_USART_ExistsReceiverIdleStatus Function**C**

```
bool PLIB_USART_ExistsReceiverIdleStatus(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiverIdle feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_ReceiverIdle](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverIdle feature is supported on the device
- false - The ReceiverIdle feature is not supported on the device

Remarks

None.

5.6.25.5.6.21 PLIB_USART_ExistsReceiverInterruptMode Function**C**

```
bool PLIB_USART_ExistsReceiverInterruptMode(  

```

```
    USART_MODULE_ID index
);
```

Description

This function identifies whether the ReceiverInterruptMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_ReceiverInterruptModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverInterruptMode feature is supported on the device
- false - The ReceiverInterruptMode feature is not supported on the device

Remarks

None.

5.6.25.5.6.22 PLIB_USART_ExistsReceiverMode Function

C

```
bool PLIB_USART_ExistsReceiverMode(
    USART_MODULE_ID index
);
```

Description

This function identifies whether the ReceiverMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_ReceiverModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverMode feature is supported on the device
- false - The ReceiverMode feature is not supported on the device

Remarks

None.

5.6.25.5.6.23 PLIB_USART_ExistsReceiverOverrunStatus Function

C

```
bool PLIB_USART_ExistsReceiverOverrunStatus(
    USART_MODULE_ID index
);
```

```
);
```

Description

This function identifies whether the ReceiverOverrunError feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_ReceiverOverrunErrorClear](#)
- [PLIB_USART_ReceiverOverrunHasOccurred](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverOverrunError feature is supported on the device
- false - The ReceiverOverrunError feature is not supported on the device

Remarks

None.

5.6.25.5.6.24 PLIB_USART_ExistsReceiverParityErrorStatus Function

C

```
bool PLIB_USART_ExistsReceiverParityErrorStatus(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the ReceiverParityError feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_ReceiverParityErrorHasOccurred](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ReceiverParityError feature is supported on the device
- false - The ReceiverParityError feature is not supported on the device

Remarks

None.

5.6.25.5.6.25 PLIB_USART_ExistsStopInIdle Function

C

```
bool PLIB_USART_ExistsStopInIdle(  

```

```
    USART_MODULE_ID index
);
```

Description

This function identifies whether the StopInIdle feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_StopInIdleEnable](#)
- [PLIB_USART_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Remarks

None.

5.6.25.5.6.26 PLIB_USART_ExistsSyncClockPolarity Function

C

```
bool PLIB_USART_ExistsSyncClockPolarity(
    USART_MODULE_ID index
);
```

Description

This function identifies whether the SyncClockPolarity feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_SyncClockPolarityIdleHighDisable](#)
- [PLIB_USART_SyncClockPolarityIdleHighEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SyncClockPolarity feature is supported on the device
- false - The SyncClockPolarity feature is not supported on the device

Remarks

None.

5.6.25.5.6.27 PLIB_USART_ExistsSyncClockSource Function

C

```
bool PLIB_USART_ExistsSyncClockSource(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the SyncClockSource feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_SyncClockSourceSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SyncClockSource feature is supported on the device
- false - The SyncClockSource feature is not supported on the device

Remarks

None.

5.6.25.5.6.28 PLIB_USART_ExistsSyncMode Function

C

```
bool PLIB_USART_ExistsSyncMode(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the SyncMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_SyncModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SyncMode feature is supported on the device
- false - The SyncMode feature is not supported on the device

Remarks

None.

5.6.25.5.6.29 PLIB_USART_ExistsTransmitter Function

C

```
bool PLIB_USART_ExistsTransmitter(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the Transmitter feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_TransmitterByteSend](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Transmitter feature is supported on the device
- false - The Transmitter feature is not supported on the device

Remarks

None.

5.6.25.5.6.30 PLIB_USART_ExistsTransmitter9BitsSend Function

C

```
bool PLIB_USART_ExistsTransmitter9BitsSend(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the Transmitter9Bits feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_Transmitter9BitsSend](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Transmitter9Bits feature is supported on the device
- false - The Transmitter9Bits feature is not supported on the device

Remarks

None.

5.6.25.5.6.31 PLIB_USART_ExistsTransmitterBreak Function

C

```
bool PLIB_USART_ExistsTransmitterBreak(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterBreak feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_TransmitterBreakSend](#)
- [PLIB_USART_TransmitterBreakSendIsComplete](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitterBreak feature is supported on the device
- false - The TransmitterBreak feature is not supported on the device

Remarks

None.

5.6.25.5.6.32 PLIB_USART_ExistsTransmitterBufferFullStatus Function

C

```
bool PLIB_USART_ExistsTransmitterBufferFullStatus(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterBufferFull feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_TransmitterBufferIsFull](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitterBufferFull feature is supported on the device
- false - The TransmitterBufferFull feature is not supported on the device

Remarks

None.

5.6.25.5.6.33 PLIB_USART_ExistsTransmitterEmptyStatus Function

C

```
bool PLIB_USART_ExistsTransmitterEmptyStatus(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterEmpty feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_TransmitterIsEmpty](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitterEmpty feature is supported on the device
- false - The TransmitterEmpty feature is not supported on the device

Remarks

None.

5.6.25.5.6.34 PLIB_USART_ExistsTransmitterEnable Function

C

```
bool PLIB_USART_ExistsTransmitterEnable(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterEnableControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_TransmitterEnable](#)
- [PLIB_USART_TransmitterDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitterEnableControl feature is supported on the device
- false - The TransmitterEnableControl feature is not supported on the device

Remarks

None.

5.6.25.5.6.35 PLIB_USART_ExistsTransmitterIdleIsLow Function

C

```
bool PLIB_USART_ExistsTransmitterIdleIsLow(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterIdleIsLow feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_TransmitterIdleIsLowDisable](#)
- [PLIB_USART_TransmitterIdleIsLowEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitterIdleIsLow feature is supported on the device
- false - The TransmitterIdleIsLow feature is not supported on the device

Remarks

None.

5.6.25.5.6.36 PLIB_USART_ExistsTransmitterIdleStatus Function

C

```
bool PLIB_USART_ExistsTransmitterIdleStatus(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterIdle feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_TransmitterIdle](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitterIdle feature is supported on the device
- false - The TransmitterIdle feature is not supported on the device

Remarks

None.

5.6.25.5.6.37 PLIB_USART_ExistsTransmitterInterruptMode Function

C

```
bool PLIB_USART_ExistsTransmitterInterruptMode(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the TransmitterInterruptMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- [PLIB_USART_TransmitterInterruptModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TransmitterInterruptMode feature is supported on the device
- false - The TransmitterInterruptMode feature is not supported on the device

Remarks

None.

5.6.25.5.6.38 PLIB_USART_ExistsWakeOnStart Function

C

```
bool PLIB_USART_ExistsWakeOnStart(  
    USART_MODULE_ID index  
);
```

Description

This function identifies whether the WakeOnStart feature is available on the USART module. When this function returns true, these functions are supported on the device:

- [PLIB_USART_WakeOnStartEnable](#)
- [PLIB_USART_WakeOnStartDisable](#)
- [PLIB_USART_WakeOnStartIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WakeOnStart feature is supported on the device
- false - The WakeOnStart feature is not supported on the device

Remarks

None.

5.6.25.6 Files

Files

Name	Description
plib_usart.h	Universal Synchronous/Asynchronous Receiver/Transmitter (USART) Peripheral Library interface header for USART functions.

Description

5.6.25.6.1 plib_usart.h

USART Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USART Peripheral Library for all families of Microchip microcontrollers. The functions in this file are common to the USART module.

Functions

	Name	Description
≡	PLIB_USART_AlternateIODisable	Disables the use of alternate I/O pins for the receiver and transmitter.
≡	PLIB_USART_AlternateIOEnable	Enables the use of alternate I/O pins for the receiver and transmitter.
≡	PLIB_USART_BaudRateAutoDetectEnable	Enables baud rate measurement on the next character, which requires reception of the Sync character.
≡	PLIB_USART_BaudRateAutoDetectIsComplete	Gets the state of the automatic baud detection.
≡	PLIB_USART_BaudRateAutoDetectOverflowHasOccurred	Identifies if the automatic baud timer has overflowed.
≡	PLIB_USART_BaudRateGet	Gets the baud rate current in use.
≡	PLIB_USART_BaudRateHighDisable	Disables the high baud rate selection.
≡	PLIB_USART_BaudRateHighEnable	Enables high baud rate selection.
≡	PLIB_USART_BaudRateHighSet	Sets the baud rate to the desired value.
≡	PLIB_USART_BaudRateModeSelect	Enables the 16-bit Baud Rate Generator (BRG).
≡	PLIB_USART_BaudRateSet	Sets the baud rate to the desired value.
≡	PLIB_USART_Disable	Disables the specific USART module
≡	PLIB_USART_Enable	Enables the specific USART module.
≡	PLIB_USART_ExistsAlternateIO	Identifies whether the AlternateIOEnableControl feature exists on the USART module.
≡	PLIB_USART_ExistsBaudRate	Identifies whether the BaudRate feature exists on the USART module.
≡	PLIB_USART_ExistsBaudRateAutoDetect	Identifies whether the BaudRateAutoDetect feature exists on the USART module.
≡	PLIB_USART_ExistsBaudRateAutoDetectOverflow	Identifies whether the BaudRateAutoDetectOverflow feature exists on the USART module.

	<u>PLIB_USART_ExistsBaudRateBitMode</u>	Identifies whether the BaudRateMode feature exists on the USART module.
	<u>PLIB_USART_ExistsBaudRateHigh</u>	Identifies whether the BaudRateHigh feature exists on the USART module.
	<u>PLIB_USART_ExistsEnable</u>	Identifies whether the EnableControl feature exists on the USART module.
	<u>PLIB_USART_ExistsHandshakeMode</u>	Identifies whether the HandShakeMode feature exists on the USART module.
	<u>PLIB_USART_ExistsIrDA</u>	Identifies whether the IrDAControl feature exists on the USART module.
	<u>PLIB_USART_ExistsLineControlMode</u>	Identifies whether the LineControlMode feature exists on the USART module.
	<u>PLIB_USART_ExistsLoopback</u>	Identifies whether the Loopback feature exists on the USART module.
	<u>PLIB_USART_ExistsOperationMode</u>	Identifies whether the OperationMode feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiver</u>	Identifies whether the Receiver feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverAddressAutoDetect</u>	Identifies whether the ReceiverAddressAutoDetect feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverAddressDetect</u>	Identifies whether the ReceiverAddressDetect feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverDataAvailableStatus</u>	Identifies whether the ReceiverDataAvailable feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverEnable</u>	Identifies whether the ReceiverEnableControl feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverFramingErrorStatus</u>	Identifies whether the ReceiverFramingError feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverIdleStateLowEnable</u>	Identifies whether the ReceiverPolarityInvert feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverIdleStatus</u>	Identifies whether the ReceiverIdle feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverInterruptMode</u>	Identifies whether the ReceiverInterruptMode feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverMode</u>	Identifies whether the ReceiverMode feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverOverrunStatus</u>	Identifies whether the ReceiverOverrunError feature exists on the USART module.
	<u>PLIB_USART_ExistsReceiverParityErrorStatus</u>	Identifies whether the ReceiverParityError feature exists on the USART module.
	<u>PLIB_USART_ExistsStopInIdle</u>	Identifies whether the StopInIdle feature exists on the USART module.
	<u>PLIB_USART_ExistsSyncClockPolarity</u>	Identifies whether the SyncClockPolarity feature exists on the USART module.
	<u>PLIB_USART_ExistsSyncClockSource</u>	Identifies whether the SyncClockSource feature exists on the USART module.
	<u>PLIB_USART_ExistsSyncMode</u>	Identifies whether the SyncMode feature exists on the USART module.
	<u>PLIB_USART_ExistsTransmitter</u>	Identifies whether the Transmitter feature exists on the USART module.
	<u>PLIB_USART_ExistsTransmitter9BitsSend</u>	Identifies whether the Transmitter9Bits feature exists on the USART module.

	<u>PLIB_USART_ExistsTransmitterBreak</u>	Identifies whether the TransmitterBreak feature exists on the USART module.
	<u>PLIB_USART_ExistsTransmitterBufferFullStatus</u>	Identifies whether the TransmitterBufferFull feature exists on the USART module.
	<u>PLIB_USART_ExistsTransmitterEmptyStatus</u>	Identifies whether the TransmitterEmpty feature exists on the USART module.
	<u>PLIB_USART_ExistsTransmitterEnable</u>	Identifies whether the TransmitterEnableControl feature exists on the USART module.
	<u>PLIB_USART_ExistsTransmitterIdleLow</u>	Identifies whether the TransmitterIdleLow feature exists on the USART module.
	<u>PLIB_USART_ExistsTransmitterIdleStatus</u>	Identifies whether the TransmitterIdle feature exists on the USART module.
	<u>PLIB_USART_ExistsTransmitterInterruptMode</u>	Identifies whether the TransmitterInterruptMode feature exists on the USART module.
	<u>PLIB_USART_ExistsWakeOnStart</u>	Identifies whether the WakeOnStart feature exists on the USART module.
	<u>PLIB_USART_HandshakeModeSelect</u>	Sets the data flow configuration.
	<u>PLIB_USART_IrDADisable</u>	Disables the IrDA encoder and decoder.
	<u>PLIB_USART_IrDAEnable</u>	Enables the IrDA encoder and decoder.
	<u>PLIB_USART_LineControlModeSelect</u>	Sets the data flow configuration.
	<u>PLIB_USART_LoopbackDisable</u>	Disables Loopback mode.
	<u>PLIB_USART_LoopbackEnable</u>	Enables Loopback mode.
	<u>PLIB_USART_OperationModeSelect</u>	Configures the operation mode of the USART module.
	<u>PLIB_USART_ReceiverAddressAutoDetectDisable</u>	Disables the automatic Address Detect mode.
	<u>PLIB_USART_ReceiverAddressAutoDetectEnable</u>	Setup the automatic Address Detect mode.
	<u>PLIB_USART_ReceiverAddressDetectDisable</u>	Enables the Address Detect mode.
	<u>PLIB_USART_ReceiverAddressDetectEnable</u>	Enables the Address Detect mode.
	<u>PLIB_USART_ReceiverAddressIsReceived</u>	Checks and return if the data received is an address.
	<u>PLIB_USART_ReceiverByteReceive</u>	Data to be received in the Byte mode.
	<u>PLIB_USART_ReceiverDataIsAvailable</u>	Identifies if the receive data is available for the specified USART module.
	<u>PLIB_USART_ReceiverDisable</u>	Disables the USART receiver.
	<u>PLIB_USART_ReceiverEnable</u>	Enables the USART receiver.
	<u>PLIB_USART_ReceiverFramingErrorHasOccurred</u>	Gets the framing error status.
	<u>PLIB_USART_ReceiverIdleStateLowDisable</u>	Disables receive polarity inversion.
	<u>PLIB_USART_ReceiverIdleStateLowEnable</u>	Enables receive polarity inversion.
	<u>PLIB_USART_ReceiverInterruptModeSelect</u>	Sets the USART receiver FIFO level.
	<u>PLIB_USART_ReceiverIsIdle</u>	Identifies if the receiver is idle.
	<u>PLIB_USART_ReceiverModeSelect</u>	Disables Single Receive mode.
	<u>PLIB_USART_ReceiverOverrunErrorClear</u>	Clears a USART Overrun error.
	<u>PLIB_USART_ReceiverOverrunHasOccurred</u>	Identifies if there was a receiver overrun error.
	<u>PLIB_USART_ReceiverParityErrorHasOccurred</u>	Gets the parity error status.
	<u>PLIB_USART_StopInIdleDisable</u>	Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).
	<u>PLIB_USART_StopInIdleEnable</u>	Discontinues operation when the device enters Idle mode.
	<u>PLIB_USART_SyncClockPolarityIdleHighDisable</u>	Sets the idle state of the clock to low.
	<u>PLIB_USART_SyncClockPolarityIdleHighEnable</u>	Sets the idle state of the clock to high.

	PLIB_USART_SyncClockSourceSelect	Sets the clock source.
	PLIB_USART_SyncModeSelect	Selects the USART mode to be asynchronous.
	PLIB_USART_Transmitter9BitsSend	Data to be transmitted in the byte mode with the 9th bit.
	PLIB_USART_TransmitterBreakSend	Transmits the break character.
	PLIB_USART_TransmitterBreakSendIsComplete	Returns the status of the break transmission
	PLIB_USART_TransmitterBufferIsFull	Gets the transmit buffer full status.
	PLIB_USART_TransmitterByteSend	Data to be transmitted in the Byte mode.
	PLIB_USART_TransmitterDisable	Disables the specific USART module transmitter.
	PLIB_USART_TransmitterEnable	Enables the specific USART module transmitter.
	PLIB_USART_TransmitterIdleIsLowDisable	Disables the Transmit Idle Low state.
	PLIB_USART_TransmitterIdleIsLowEnable	Enables the Transmit Idle Low state.
	PLIB_USART_TransmitterInterruptModeSelect	Set the USART transmitter interrupt mode.
	PLIB_USART_TransmitterIsEmpty	Gets the transmit shift register empty status.
	PLIB_USART_TransmitterIsIdle	Gets the transmit buffer full status.
	PLIB_USART_WakeOnStartDisable	Disables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartEnable	Enables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartIsEnabled	Gets the state of the sync break event completion.

File Name

plib_usart.h

Company

Microchip Technology Inc.

5.6.26 USB Peripheral Library

5.6.26.1 Introduction

This library provides a low-level abstraction of the USB module on Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

USB Peripheral Library for Microchip Microcontrollers



USB Overview

USB is an asynchronous serial interface with a tiered star configuration. USB is implemented as a master/slave configuration. On a given bus, there can be multiple (up to 127) slaves (devices), but there is only one master (host). There are three possible module implementations: host, device and OTG dual role. The user should have an understanding of the USB documents available on the USB implementers web site (www.usb.org).

Device

The USB device accepts data from the host and responds to requests for data. It performs some peripheral functions, such as a mouse or data storage device.

- Functionality may be class or vendor-specific
- Draws 100 mA or less before configuration
- Can draw up to 500 mA after successful negotiation with the host
- Can support low-speed, full-speed or hi-speed protocol. Hi-speed support requires implementation of full-speed. (*Low, Full, and Hi-Speed supported by this module.*)
- Supports control transfers. Supports data transfers required for implementation
- Optionally supports Session Request Protocol (SRP)
- Can be bus-powered or self-powered

Host

The host is the master in a USB system and is responsible for identifying all devices connected to it (enumeration), initiating all transfers, allocating bus bandwidth and supplying power to any bus-powered USB devices connected directly to it. There are two types of hosts.

USB Standard Host:

- A large variety of devices are supported
- This host supports all USB transfer types
- USB hubs are supported to allow connection of multiple devices simultaneously
- Device drivers can be updated to support new devices
- A type 'A' receptacle is used for each port
- Each port must be able to deliver a minimum of 100 mA for a configured or unconfigured

- device, and optionally, up to 500 mA for a configured device
- Full-Speed and Low-Speed must be supported. Hi-Speed can be supported.
- This is a typical personal computer implementation

Embedded Host

- Only supports a specific list of devices, referred to as a Targeted Peripheral List (TPL)
- This type of host is only required to support transfer types required by devices in the TPL
- USB hub support is optional (*Provided in this module*)
- Device drivers are not required to be pocketable
- A type 'A' receptacle is used for each port
- Only speeds required by devices in the TPL must be supported. (*Low, Full, and Hi-Speed supported by this module.*)
- Each port must be able to deliver a minimum of 100 mA for a configured or unconfigured device, and optionally, up to 500 mA for a configured device
- This is a typical implementation for a microcontroller

USB On-the-Go (OTG)

The OTG dual role device supports both USB host and device functionality. OTG dual role devices use a micro-AB receptacle. This allows a micro-A or a micro-B plug to be attached. Both the micro-A and micro-B plugs have an additional pin, the ID pin, to signify the [connect](#) type. The plug type, micro-A or micro-B, determines the default role of the OTG device, host or USB device. An OTG device will perform the role of a host when a micro-A plug is detected. When a micro-B plug is detected, the role of a USB device is performed.

When an OTG device is directly connected to another OTG device using an OTG cable, micro-A to micro-B, Host Negotiation Protocol (HNP) can be used to swap the roles of the host and USB device between the two without disconnecting and reconnecting cabling. To differentiate between the two OTG devices, the term, "A-device", is used to refer to the device connected to the micro-A plug and "B-device" is used to refer to the OTG device connected to the micro-B plug.

OTG dual role operating as a host (A-device):

- Only supports a specific list of devices, referred to as a Targeted Peripheral List (TPL). Generic class support is not allowed.
- Only required to support transaction types required by devices in the TPL
- USB hub support is optional. (*Multi-point support provided by this module.*)
- Device drivers are not required to be pocketable
- Only a single micro-AB receptacle is used
- Only full speed must be supported. Hi-Speed and/or Low-Speed can be supported. (*Low, Full, and Hi-Speed supported by this module.*)
- The USB port must be able to deliver a minimum of 8 mA for a configured or unconfigured device, and optionally, up to 500 mA for a configured device
- Supports Host Negotiation Protocol (HNP) and Session Request Protocol (SRP). The host can switch roles to become a device. The initial role as a host or device is determined by the plug type, micro-A or micro-B, inserted into the micro-AB receptacle
- The A-device supplies VBUS power, when the bus is powered, even if the roles are swapped using HNP

OTG dual role operating as a USB device (B-device):

- Class or vendor-specific functionality
- Draws 8 mA or less before configuration
- Is typically self-powered due to low-current requirements, but can draw up to 500 mA after successful negotiation with the host

- Only a single micro-AB receptacle is used
- Must support Full-Speed. Support of low speed and/or Hi-Speed is optional
- Supports control transactions. Supports data transactions required for implementation.
- Supports Session Request Protocol (SRP) and/or Host Negotiation Protocol (HNP). (*This module supports both SRP and HNP.*)
- The A-device supplies VBUS power, when the bus is powered, even if the roles are swapped using HNP

Additional Features of the Hi-Speed USB Module

- Operates either as a function controller of a Hi-Speed/Full-Speed USB device or as the host/device in a point-to-point or multi-point communications with other USB function
- Supports OTG communications with on or more Hi-Speed, Full-Speed, or Low-Speed devices
- Provides soft [connect/disconnect](#).
- In addition to Endpoint Zero, supports seven transmit and seven receive endpoints
- Dynamic FIFO sizing for Endpoints 1-7. (Endpoint Zero FIFO fixed at 64 bytes.) FIFOs use module-internal SRAM.
- Module-internal eight channel DMA with access to all FIFOs
- All host transaction scheduling supported in hardware
- Supports Link Power Management

5.6.26.2 Release Notes

MPLAB Harmony Version: v0.70b **USB PLIB Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.26.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.26.4 Using the Library

This topic describes the basic architecture of the USB Peripheral Library and provides information and examples on how to use it.

Interface Header File: peripheral.h

The interface to the USB Peripheral Library is defined in the "**plib_usb.h**" header file, which is included by the "peripheral.h" peripheral library header file. Any C language source (.c) file that uses the USB Peripheral Library must include "peripheral.h".

Library File:

The USB Peripheral Library is part of the processor-specific peripheral library archive (.a) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

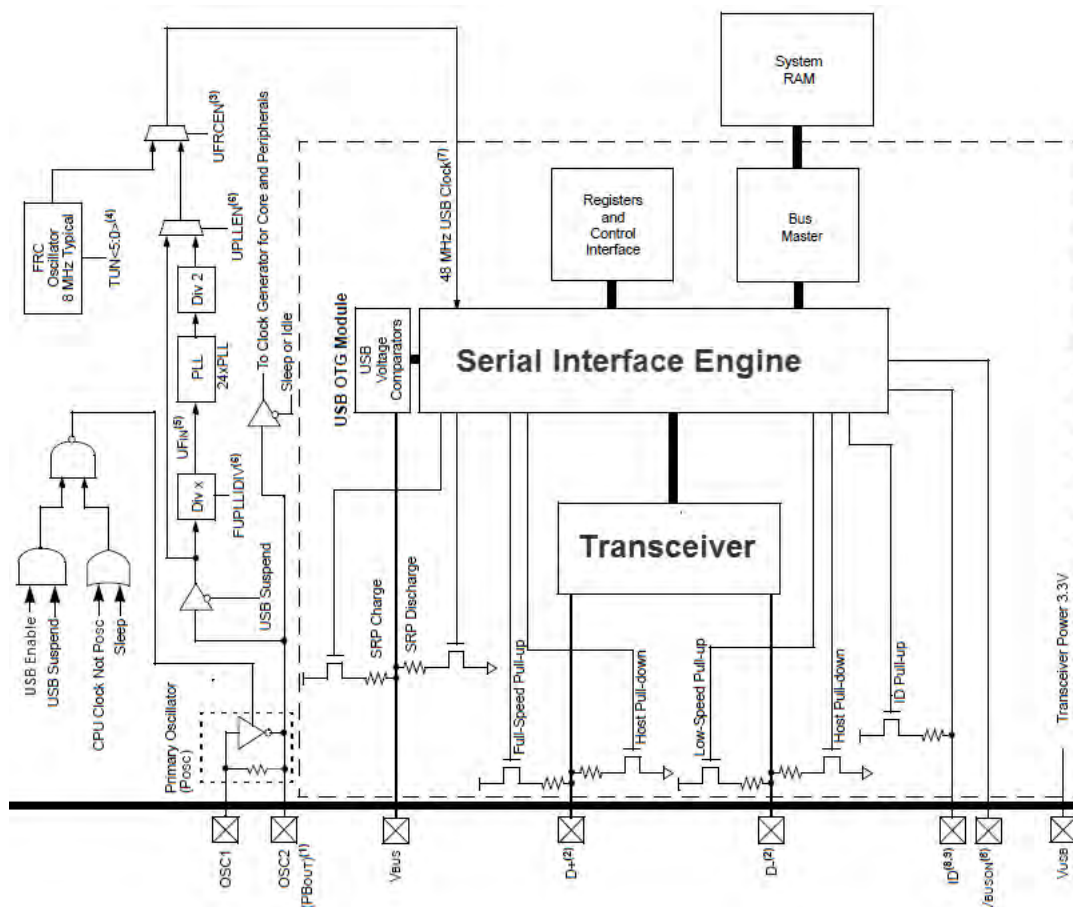
5.6.26.4.1 Hardware Abstraction Models

This section describes the hardware abstraction model for the USB Peripheral Library.

Description

Hardware Abstraction Model

Figure 1. shows the USB Module for the PIC32 family of devices.



- Note**
- 1: PB clock is only available on this pin for select EC modes.
 - 2: Pins can be used as digital inputs when USB is not enabled.
 - 3: This bit field is contained in the OSCCON register.
 - 4: This bit field is contained in the OSCSTRM register.
 - 5: USB PLL U_{FIN} requirements: $4\text{ MHz} \leq U_{FIN} \leq 5\text{ MHz}$.
 - 6: This bit field is contained in the DEVCFG2 register.
 - 7: A 48 MHz clock is required for proper USB operation.
 - 8: Pins can be used as GPIO when the USB OTG module is disabled.
 - 9: Pin is pulled high internally when USB OTG module is enabled.

Figure 1: PIC32 USB Module

The USB OTG module contains analog and digital components to provide a USB 2.0 full-speed and low-speed embedded host, full-speed device, or On-The-Go (OTG) implementation with a minimum of external components. This module in Host mode is intended for use as an embedded host and therefore does not implement a Universal Host Controller Interface (UHCI) or a Open Host Controller Interface (OHCI).

The USB OTG module consists of the clock generator, the USB voltage comparators, the transceiver, the Serial Interface Engine (SIE), a dedicated USB Bus Master, pull-up and pull-down resistors and the register interface.

The clock generator provides the 48 MHz clock, which is required for USB full-speed and low-speed communication. The voltage comparators monitor the voltage on the VBUS pin to determine the state of the bus. The transceiver provides the analog translation between the USB bus and the digital logic. The SIE is a state machine that transfers data to and from the endpoint buffers, and generates the hardware protocol for data transfers. The USB Bus Master transfers data between the data buffers in RAM and the SIE. The integrated pull-up and pull-down resistors eliminate the need for external signaling components. The register interface allows the CPU to configure and communicate with the module.

Figure 2 shows the USB module found in PIC24/dsPIC33 devices.

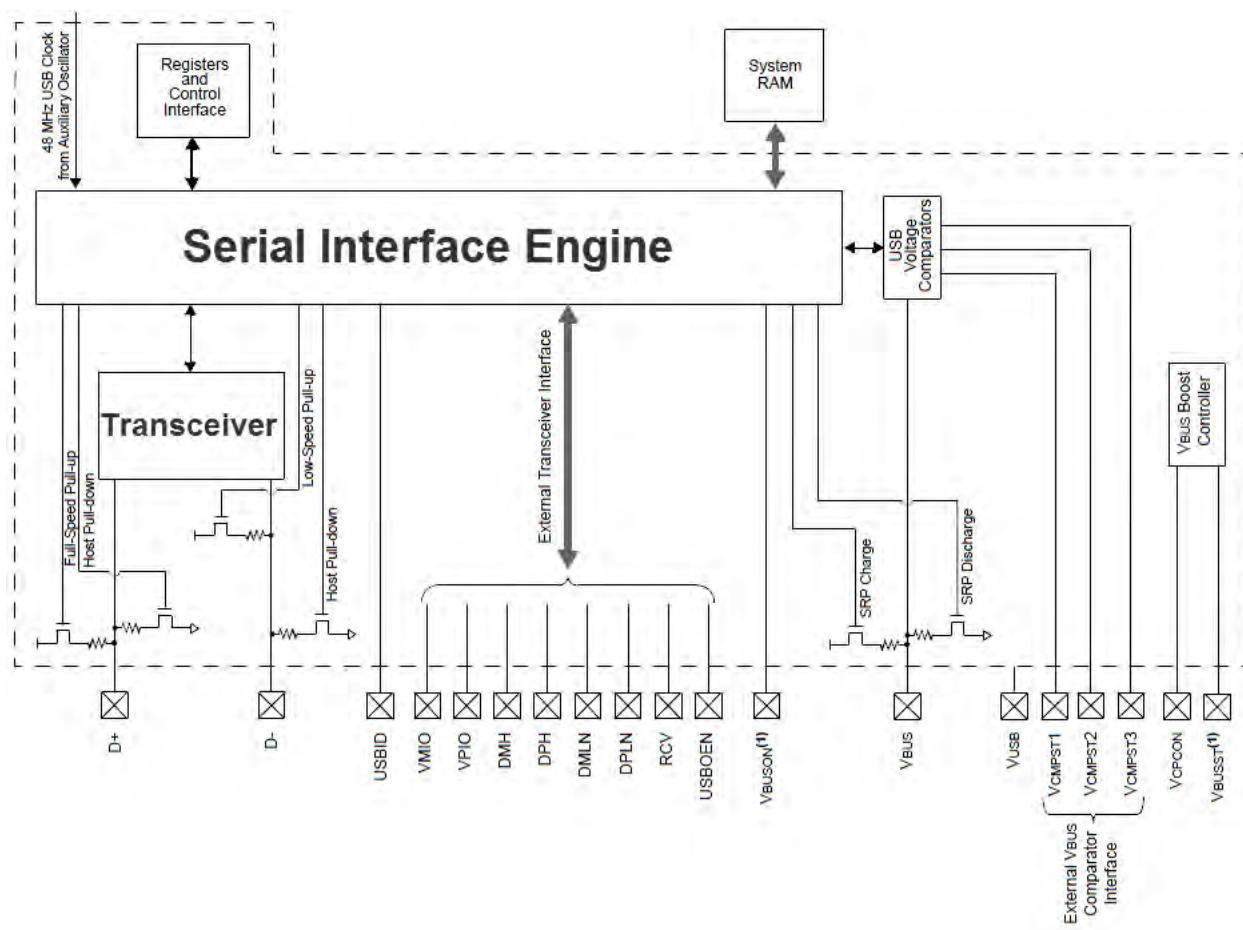
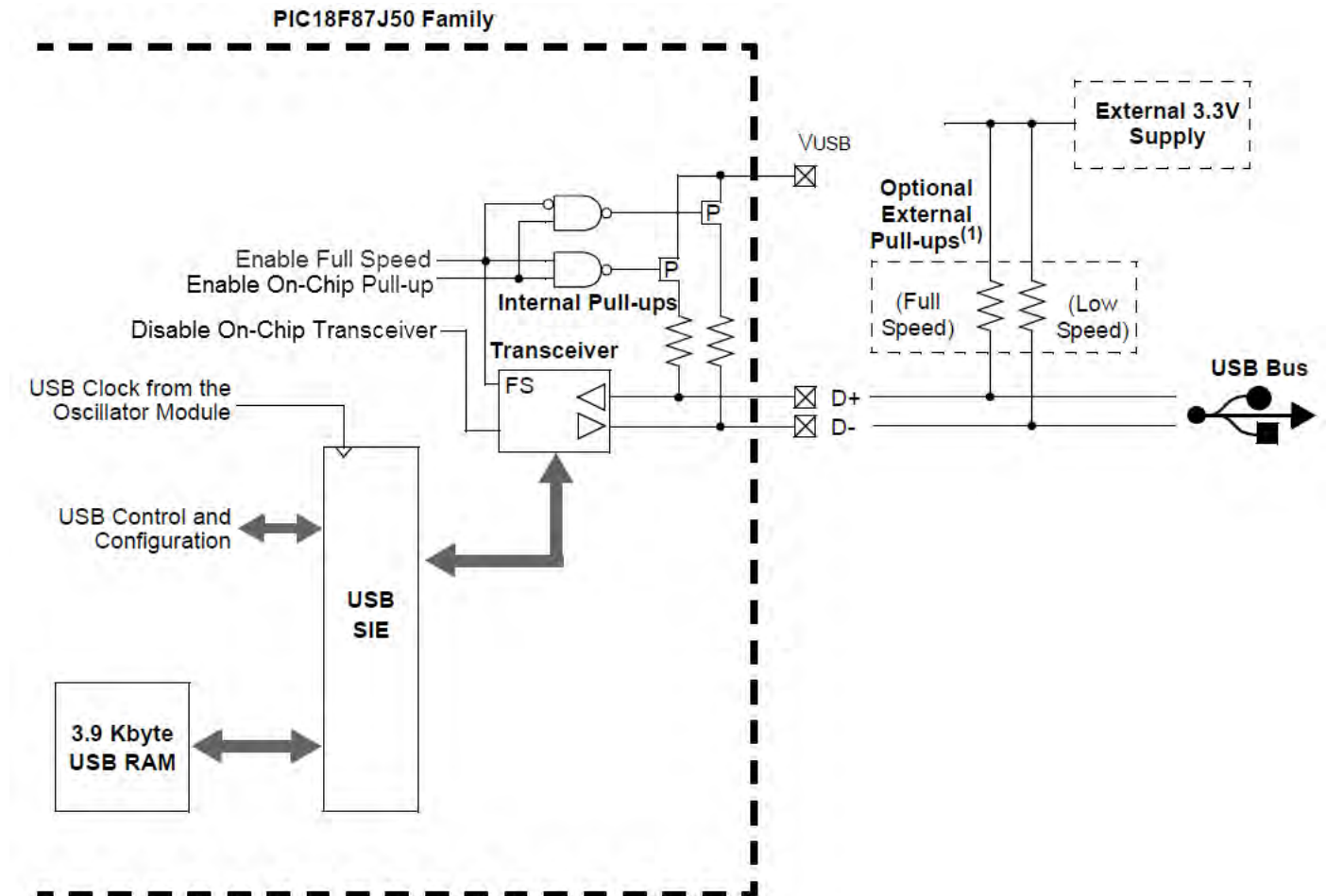


Figure 2: PIC24/dsPIC33 USB Module

The two implementations are nearly identical, but the PIC24/dsPIC33 USB module also supports using an external USB transceiver.

Figure 3 provides an example of a USB module on an 8-bit device.

**Figure 3: PIC18 USB Module**

5.6.26.4.2 Library Overview

This library provides the low level abstraction of the USB Peripheral on Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Figure 4 describes the abstraction model employed by the USB Peripheral Library.

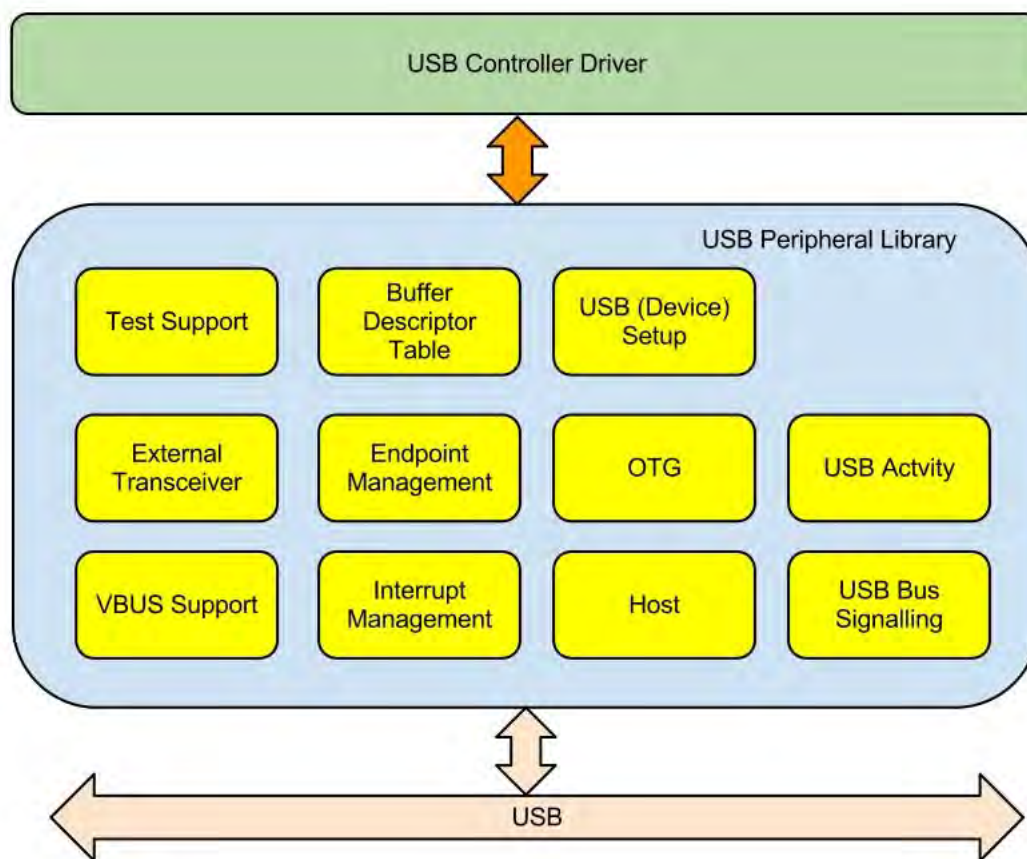


Figure 4: USB Peripheral Library Abstraction Model

The peripheral functions are grouped together under different function groups. These application must then use a combination of function from the different function groups to achieve the desired USB peripheral operation.

Library Interface Section	Description
USB Setup Functions	This section provides functions to perform general USB peripheral setup such as functions to set USB Speed, control on-chip pull ups, etc.
Buffer Descriptor Table Functions	This section provides functions that allow the application to setup, configure and access/modify the Buffer Descriptors in the Buffer Descriptor Table.
USB Activity Functions	This section provides function that allow the application to monitor bus conditions on the USB.
USB Bus Signaling Functions	This section provides functions that allow the application to generate Reset and Resume Signaling.
Last Transaction Status Functions	This section provides functions that allow the application to query conditions of the USB peripheral.
Endpoints Functions	This section provides functions that allow the application to manage endpoints.
Interrupts Functions	This section provides functions that allow the application to enable, disable and query the status interrupts in the USB peripheral.
Host Functions	This section provides functions that are required to operate the USB module while in Host mode.

On-The-Go (OTG) Functions	This section provides functions that are required to operate the USB module while in OTG mode.
External Transceiver Support Functions	This section provides function that are required to operate to enable/disable the external transceiver interface.
VBus Support Functions	This section provides functions that allow VBUS level monitoring and VBUS boost PWM module control.
Test Support Functions	This section provides functions that enable/disable test signals needed for eye pattern measurement.

5.6.26.4.3 USB Buffers and the Buffer Descriptor Table (BDT)

This section describes USB buffers and the Buffer Descriptor Table.

Description

All USB endpoints are implemented using buffers and control bits in RAM. Both software and the USB module have access to these buffers and control bits. To arbitrate this access a semaphore flag system is used.

Each endpoint can be configured for transmit only, receive only or transmit *and* receive. Transmit and receive functions have separate buffers. For each buffer there is a Buffer Descriptor (BD) in the Buffer Descriptor Table (BDT). On most devices the BDT has room for two transmit and two receive buffers for each endpoint, supporting ping-pong buffering. Buffer descriptors must be aligned on 512 byte boundaries to enable the USB module to correctly read each entry.

The starting address of the Buffer Descriptor Table is set by the [PLIB_USB_BDTBaseAddressSet](#).

```
#define USB_MAX_EP_NUMBER 15

#define USB_PING_PONG_MODE_SELECTION  USB_PING_PONG__FULL_PING_PONG

#if (USB_PING_PONG_MODE_SELECTION == USB_PING_PONG__NO_PING_PONG)
    #define BDT_NUM_ENTRIES ((USB_MAX_EP_NUMBER + 1) * 2)
#elif (USB_PING_PONG_MODE_SELECTION == USB_PING_PONG__EP0_OUT_ONLY)
    #define BDT_NUM_ENTRIES (((USB_MAX_EP_NUMBER + 1) * 2)+1)
#elif (USB_PING_PONG_MODE_SELECTION == USB_PING_PONG__FULL_PING_PONG)
    #define BDT_NUM_ENTRIES ((USB_MAX_EP_NUMBER + 1) * 4)
#elif (USB_PING_PONG_MODE_SELECTION == USB_PING_PONG__ALL_BUT_EP0)
    #define BDT_NUM_ENTRIES (((USB_MAX_EP_NUMBER + 1) * 4)-2)
#else
    #error "No ping pong mode defined."
#endif

#if defined(__18CXX)
    #pragma udata gBDT=USB_BDT_ADDRESS
#elif defined(__PIC32MX__)
    volatile union USB_BDT_ENTRY_TAG gBDT[BDT_NUM_ENTRIES] __attribute__((aligned (512)));
#else
    volatile union USB_BDT_ENTRY_TAG gBDT[BDT_NUM_ENTRIES];
#endif//defined(__18CXX)
```

(For PIC18 devices, this isn't necessary since the BDT always starts at 0x0400 in memory.)

Each Buffer Descriptor entry in the BDT has a **bufferAddress** pointer to the Buffer Descriptor's buffer in memory (RAM or FLASH).

Figure 5 shows the structure for most PIC24 and all PIC32 devices. These devices support ping-pong buffering for all endpoints.

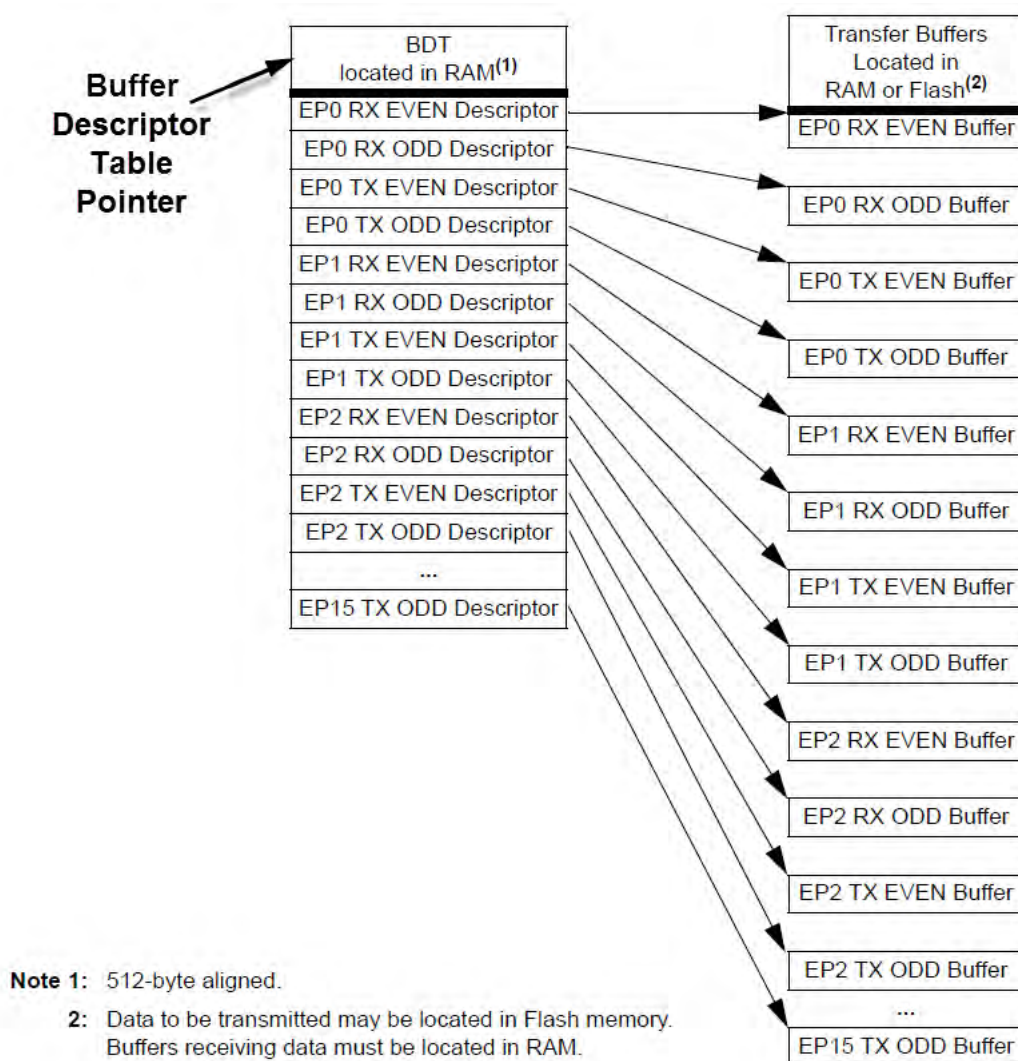


Figure 5: Buffer Descriptor Table and Endpoint Buffers

With 16 endpoints (Endpoints 0 - 15) there are 64 possible entries in the Buffer Descriptor Table:

```
//                                     (Even,Odd) or (Ping,Pong)
//                                     (Rx, Tx) |
//                                     (EP0 to EPn) |
volatile USB_BDT_ENTRY myBDT[(USB_MAX_EP_NUMBER + 1)][2][2] __attribute__((aligned(512)))
);
```

On PIC18F and PIC24F devices, ping-pong buffering can be disabled for all of some of the endpoints. Figure 6 shows the buffering options available for PIC24F devices and Figure 7 shows the buffering options available for PIC18F devices.

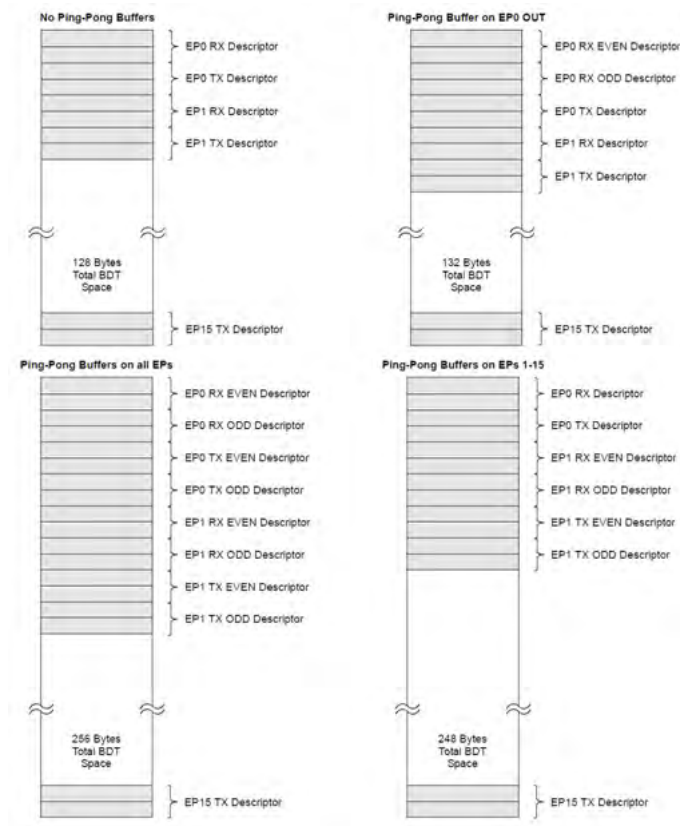
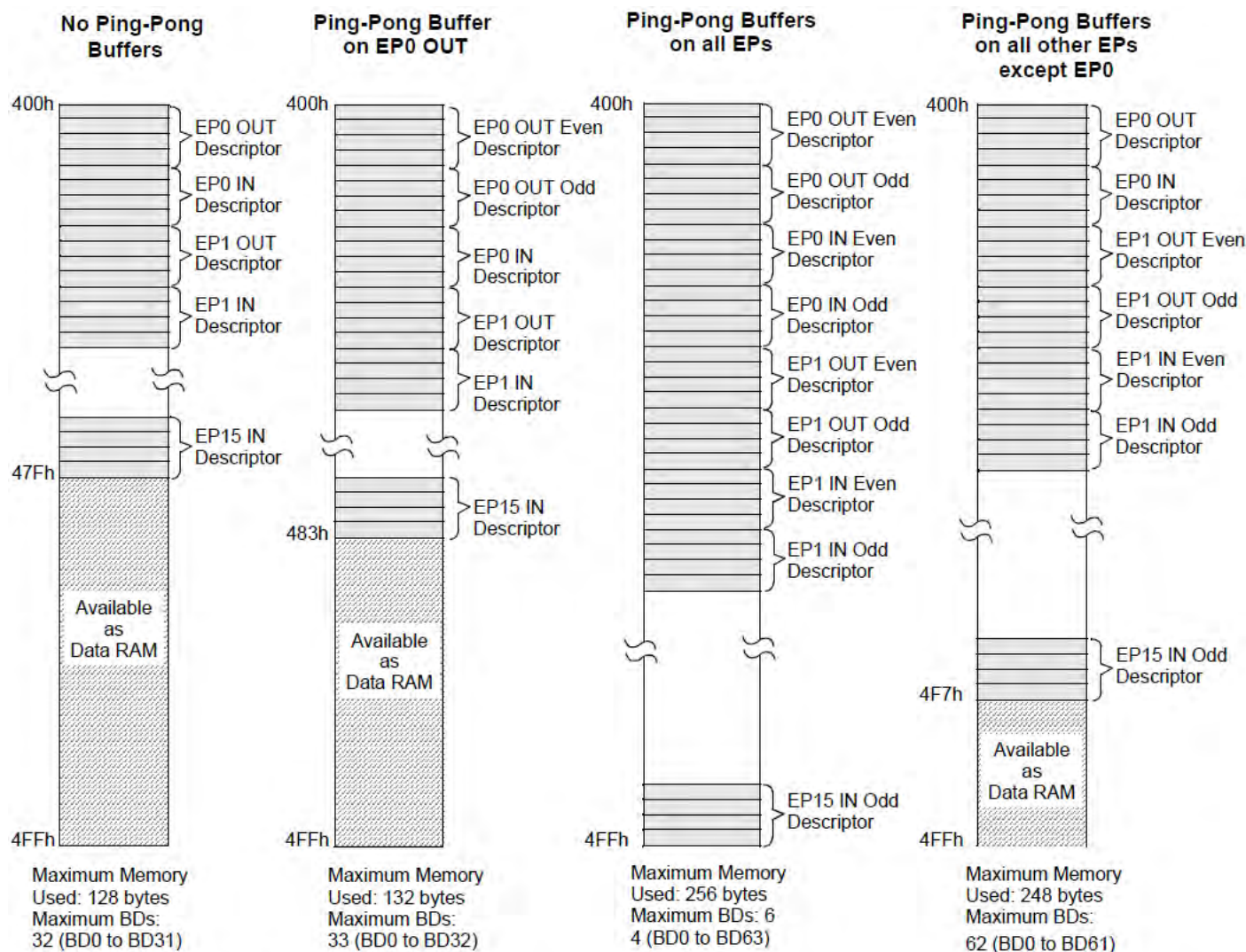


Figure 6: Buffer Descriptor Table Mapping for Various PIC24F Ping-Pong Configurations



Note: Memory area not shown to scale.

Figure 7: Buffer Descriptor Table Mapping for Various PIC18F Ping-Pong Configurations

Peripheral library functions are provided to support reading each BD while hiding the details of BDT addressing that can change when ping-pong buffering is disabled for one or more endpoints. The format of Buffer Descriptors vary from family to family and it is not important to know the differences, since peripheral library functions hide these details. To understand the information stored in each Buffer Descriptor Table entry it is informative to look at the C language description for one family of devices. (All device families support the same information but with different ways of packing the data into memory.)

For most PIC24/dsPIC33 devices and all PIC32 devices each BD is 8 bytes long and is described by this C code:

```
typedef union _USB_BDT_ENTRY __attribute__((packed))
{
    uint64_t      dValue;           //Double Long Integer Value
    uint32_t      lValue[2];        //Long Integer Values
    uint16_t      sValue[4];        //Short Integer Values
    struct __attribute__((packed))
    {
        USB_BD_STATUS  bufferStatus; //((RW) Buffer Status
        uint16_t        byteCount:10; //((RW) Byte Count
        uint8_t          : 6;         //(( ) Reserved
    }
}
```

```

        uint32_t      bufferAddress;  //(RW) Buffer Address in Data Ram or Flash
    };
} USB_BDT_ENTRY;

```

For transmit buffers **byteCount** is set to the length of the buffer using [PLIB_USB_BufferByteCountSet](#). For receive buffers [PLIB_USB_BufferByteCountSet](#) is used to set the maximum allowable receive data length. [PLIB_USB_BufferByteCountGet](#) is used to determine how many bytes were actually transmitted or received. The memory address of each buffer is found in **bufferAddress**. The functions [PLIB_USB_BufferAddressSet](#) and [PLIB_USB_BufferAddressGet](#) support this field in the Buffer Descriptor.

Buffer status is stored in the 16 least significant bits of each BDT entry:

```

typedef union _USB_BD_STATUS __attribute__((packed))
{
    uint16_t      sValue;  //Short Integer Value
    struct __attribute__((packed))
    {
        uint8_t      :2;  //( ) Reserved
        uint8_t      stallEnable      :1;  //( W) Buffer Stall Enable
        uint8_t      dataToggleSyncEnable :1;  //( W) Data Toggle Synch Enable
        uint8_t      :1;  //( ) Reserved
        uint8_t      :1;  //( ) Reserved
        uint8_t      dataToggle      :1;  //(RW) Data Toggle Synch Value
        uint8_t      USBOwnsBuffer    :1;  //(RW) USB Ownership of buffer and BDT entry
    };
    struct __attribute__((packed))
    {
        uint8_t      :2;  //( ) Reserved
        uint8_t      packetID          :4;  //(R ) Packet Identifier (PID)
    };
} USB_BD_STATUS;

```

The fields **stallEnable** and **dataToggleSyncEnable** in the Buffer Descriptor status structure are writable but not readable. The function [PLIB_USB_BufferStallEnable](#) sets the **stallEnable** bit. (Hardware will clear the bit after receiving a SETUP token from the host.) The functions [PLIB_USB_BufferDataToggleSyncEnable](#) and [PLIB_USB_BufferDataToggleSyncDisable](#) manipulate **dataToggleSyncEnable**.

On a read from the status structure **stallEnable** and **dataToggleSyncEnable** and the next two (reserved) bits are replaced by the Packet Identifier (PID), called **packetID** in the structure. The function [PLIB_USB_BufferPIDGet](#) reads **packetID**.

The rest of the status structure is writable and readable: **dataToggle** and **USBOwnsBuffer**.

The **dataToggle** (DATA0 or DATA1) of a receive buffer is determined by using [PLIB_USB_BufferDataToggleGet](#). For transmit buffers the **dataToggle** value is set using [PLIB_USB_BufferDataToggleSelect](#).

The field **USBOwnsBuffer** is used by both software and the USB module as a semaphore. The function [PLIB_USB_BufferReleaseToUSB](#) releases the buffer to the USB module and the function [PLIB_USB_BufferReleasedToSW](#) returns a boolean **true** when the buffer is released back to software by the USB module. As long as [PLIB_USB_BufferReleasedToSW](#) returns **false**, software should not change the Buffer Descriptor Table entry or its associated buffer.

5.6.26.4.4 USB Setup Example

This section provides example code for setting up the USB module.

Description

Following is an example of how to set up the USB module:

```

#define USB_MAX_EP_NUMBER 15

#define USB_PING_PONG_MODE_SELECTION  USB_PING_PONG__FULL_PING_PONG

```

```

#if (USB_PING_PONG_MODE_SELECTION == USB_PING_PONG_NO_PING_PONG)
#define BDT_NUM_ENTRIES ((USB_MAX_EP_NUMBER + 1) * 2)
#elif (USB_PING_PONG_MODE_SELECTION == USB_PING_PONG_EP0_OUT_ONLY)
#define BDT_NUM_ENTRIES (((USB_MAX_EP_NUMBER + 1) * 2)+1)
#elif (USB_PING_PONG_MODE_SELECTION == USB_PING_PONG_FULL_PING_PONG)
#define BDT_NUM_ENTRIES ((USB_MAX_EP_NUMBER + 1) * 4)
#elif (USB_PING_PONG_MODE_SELECTION == USB_PING_PONG_ALL_BUT_EP0)
#define BDT_NUM_ENTRIES (((USB_MAX_EP_NUMBER + 1) * 4)-2)
#else
#error "No ping pong mode defined."
#endif

#if defined(__18CXX)
#pragma udata gBDT=USB_BDT_ADDRESS
#elif defined(__PIC32MX__)
volatile union USB_BDT_ENTRY_TAG gBDT[BDT_NUM_ENTRIES] __attribute__ (( aligned (512) ));
#else
volatile union USB_BDT_ENTRY_TAG gBDT[BDT_NUM_ENTRIES];
#endif//defined(__18CXX))

unsigned int bufferTransactionCount[BDT_NUM_ENTRIES];

// Turn off USB module
PLIB_USB_Disable( usbID );

// Setup the Hardware
if ( usbModuleSetup.StopInIdle )
{
    PLIB_USB_StopInIdleEnable( usbID );
}
else
{
    PLIB_USB_StopInIdleDisable( usbID );
}

if ( usbModuleSetup.SuspendInSleep )
{
    PLIB_USB_AutoSuspendEnable( usbID );
}
else
{
    PLIB_USB_AutoSuspendDisable( usbID );
}

PLIB_USB_OperatingModeSelect( usbID, usbModuleSetup.OpMode );
PLIB_USB_PingPongModeSelect( usbID, usbModuleSetup.ppMode );

// Reset all ping pong buffers to "Even"
PLIB_USB_PingPongFreeze( usbID );
PLIB_USB_PingPongUnfreeze( usbID );

// Interrupt flag cleared on the safer side
IFS1bits.USBIF = 0;

// Enable USB module interrupts
PLIB_USB_InterruptEnable( usbID, DRV_USB_GEN_INT_ENABLES );

// Disable all OTG interrupts
PLIB_USB_OTG_InterruptDisable( usbID, DRV_USB_OTB_INT_ENABLES );

// Enable all Error interrupts
PLIB_USB_ErrorInterruptEnable( usbID, DRV_USB_ERR_INT_ENABLES );

// Enable the interrupt source in case of interrupt mode

```

```

    IEC1bits.USBIE = 1;

    // Setting the Interrupt Priority in case of interrupt mode.
    IPC11bits.USBIP = 4;

    // Initialize BDT
    for ( iEntry = 0; iEntry < BDT_NUM_ENTRIES; iEntry++ )
    {
    #if (defined(__18CXX) || defined(__PIC24F__))
        gBDT[iEntry].lValue = 0ul;
    #elif (defined(__dsPIC33E__) || defined(__PIC24E__) || defined(__PIC32MX__))
        gBDT[iEntry].lValue[0] = 0ul;
        gBDT[iEntry].lValue[1] = 0ul;
    #else
        #error "Unknown device type"
    #endif
        bufferTransactionCount[iEntry] = 0;
    } //end for ( iEntry = 0; iEntry < BDT_NUM_ENTRIES; iEntry++ )

    // Inform USB module of BDT's address in memory
    #if defined(__PIC32MX__)
        // For PIC32
        PLIB_USB_BDTBaseAddressSet( usbID , (void *)((uint32_t)KVA_TO_PA(gBDT)) );
    #else
        // Everybody else
        PLIB_USB_BDTBaseAddressSet(MY_USB_INSTANCE, (void *)&gBDT );
    #endif

    /* *****
    /* SETUP ENDPOINT ZERO */
    /* *****
    PLIB_USB_BufferAddressSet( usbID, (void*)gBDT, PLIB_USB_PingPongModeGet( usbID ),
        0, USB_BUFFER_RX, USB_BUFFER_EVEN, pSetupRcvBuffer );

    // Configure Endpoint Zero control register
    PLIB_USB_EP0LSDirectConnectDisable( usbID ); // For Hosts, included for completeness
    PLIB_USB_EP0NakRetryEnable( usbID ); // For Hosts, included for completeness
    PLIB_USB_EPnControlTransferEnable( usbID, 0 ); // Enable control transfers
    PLIB_USB_EPnRxSelect( usbID, 0, USB_EP_RX ); // Enable receive
    PLIB_USB_EPnHandshakeEnable( usbID, 0 ); // Enable handshaking

    // Setup Endpoint Zero's receive buffer BDT entries
    ppMode = PLIB_USB_PingPongModeGet( usbID ); // Ping Pong Mode needed for BDT entry indexing

    // Clear PID bits
    PLIB_USB_BufferPIDBitsClear(usbID, (void*)gBDT, ppMode, 0, USB_BUFFER_RX, USB_BUFFER_EVEN);
//Rx0

    // Disable buffer stall
    PLIB_USB_BufferStallDisable(usbID, (void*)gBDT, ppMode, 0, USB_BUFFER_RX, USB_BUFFER_EVEN);
//Rx0

    // Enable Data Toggle Synchronization
    PLIB_USB_BufferDataToggleSyncEnable(usbID, (void*)gBDT, ppMode, 0, USB_BUFFER_RX, USB_BUFFER_EVEN);
//Rx0

    // Set Data0/1 to Data0
    PLIB_USB_BufferDataToggleSelect(usbID, (void*)gBDT, ppMode, 0, USB_BUFFER_RX, USB_BUFFER_EVEN, USB_BUFFER_DATA0);
//Rx0

    // Setup packets have 8 bytes of data payload
    PLIB_USB_BufferByteCountSet(usbID, (void*)gBDT, ppMode, 0, USB_BUFFER_RX, USB_BUFFER_EVEN, 8);

    // Release buffers to USB module

```

```
PLIB_USB_BufferReleaseToUSB(usbID, (void*)gBDT, ppMode, 0, USB_BUFFER_RX, USB_BUFFER_EVEN);
PLIB_USB_Enable( gDrvUSBObj[hDriver].usbID ); // Turn on USB module
```

5.6.26.5 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.26.6 Library Interface

Data Types and Constants

	Name	Description
	USB_BUFFER_DATA01	Provides enumeration data toggle for a buffer.
	USB_BUFFER_DIRECTION	Provides enumeration transmit/receive direction for a buffer.
	USB_BUFFER_PING_PONG	Enumerates the ping-pong buffer (Even vs. Odd).
	USB_BUFFER_SCHEDULE_DATA01	Provides enumeration data toggle for a buffer.
	USB_EP_TXRX	Provides enumeration transmit/receive setup for an endpoint.
	USB_OPMODES	Provides enumeration of operating modes supported by USB.
	USB_OTG_INTERRUPTS	Provides enumeration of interrupts related to the USB On-The-Go (OTG) module.
	USB_OTG_PULL_UP_PULL_DOWN	USB OTG pull-Up and pull-Down resistors for D+ and D- .
	USB_PID	Legal PID values.
	USB_PING_PONG_MODE	Supports the four modes of ping-pong buffering for PIC18 and PIC24F devices.
	USB_PING_PONG_STATE	Decodes which buffer (Even vs. Odd) was used for the last transaction.
	USB_TOKEN_SPEED	Provides enumeration of available token speeds.
	USB_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

Buffer Descriptor Table Functions

	Name	Description
	PLIB_USB_BDTBaseAddressGet	Returns the base address of the Buffer Descriptor Table.
	PLIB_USB_BDTBaseAddressSet	Sets the base address for the Buffer Descriptor Table for PIC24 and PIC32 devices.
	PLIB_USB_BufferAddressGet	Gets the memory address of an endpoint buffer.
	PLIB_USB_BufferAddressSet	Sets the endpoint buffer address.
	PLIB_USB_BufferAllCancelReleaseToUSB	Cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.
	PLIB_USB_BufferByteCountGet	Returns the endpoint buffer byte count.
	PLIB_USB_BufferByteCountSet	Sets the buffer byte count.
	PLIB_USB_BufferCancelReleaseToUSB	Cancels release of the endpoint buffer by software, allowing software to again access the buffer.
	PLIB_USB_BufferClearAll	Clears (zeros out) entries in the Buffer Descriptor Table.
	PLIB_USB_BufferClearAllDTSEnable	Clears the endpoint descriptor entry and enables data toggle synchronization.
	PLIB_USB_BufferDataToggleGet	Returns data synchronization (DATA0 or DATA1) for the endpoint buffer.
	PLIB_USB_BufferDataToggleSelect	Sets the endpoint buffer to DATA0 or DATA1.

	<u>PLIB_USB_BufferDataToggleSyncDisable</u>	Disables DATA0/DATA1 synchronization between the device and host.
	<u>PLIB_USB_BufferDataToggleSyncEnable</u>	Enables DATA0/DATA1 synchronization between the device and host.
	<u>PLIB_USB_BufferEP0RxStatusInitialize</u>	Initializes the Endpoint 0 RX endpoint buffer descriptors.
	<u>PLIB_USB_BufferIndexGet</u>	Gets the Buffer Descriptor Table index for a buffer.
	<u>PLIB_USB_BufferPIDBitsClear</u>	Clears the Buffer Status bits in the Buffer Descriptor Table.
	<u>PLIB_USB_BufferPIDGet</u>	Returns the token packet ID (PID) from the endpoint buffer status.
	<u>PLIB_USB_BufferReleasedToSW</u>	Returns the boolean flag value of 'true' when the buffer has been released by the USB module.
	<u>PLIB_USB_BufferReleaseToUSB</u>	Releases the endpoint buffer by software, allowing the USB module access to the buffer.
	<u>PLIB_USB_BufferSchedule</u>	Hands over a buffer to the USB module along with the buffer address and byte count.
	<u>PLIB_USB_BufferStallDisable</u>	Disables STALL handshaking for the associated endpoint buffer.
	<u>PLIB_USB_BufferStallEnable</u>	Enables STALL handshaking for the associated endpoint buffer.
	<u>PLIB_USB_BufferStallGet</u>	Returns the buffer stall status for an endpoint/direction/ping-pong.

Endpoints Functions

	Name	Description
	<u>PLIB_USB_EP0HostSetup</u>	Sends token to the specified address.
	<u>PLIB_USB_EP0LSDirectConnectDisable</u>	Disables direct connection to a low-speed device for Endpoint 0.
	<u>PLIB_USB_EP0LSDirectConnectEnable</u>	Enables direct connection to a low-speed device for Endpoint 0.
	<u>PLIB_USB_EP0NakRetryDisable</u>	Disables retrying of NAKed transactions.
	<u>PLIB_USB_EP0NakRetryEnable</u>	Enables retrying NAK'd transactions for Endpoint 0.
	<u>PLIB_USB_EPnAttributesClear</u>	Clears the set attributes of the specified endpoint.
	<u>PLIB_USB_EPnAttributesSet</u>	Configures attributes of the endpoint such as direction, handshake capability and direction.
	<u>PLIB_USB_EPnControlTransferDisable</u>	Disables endpoint control transfers.
	<u>PLIB_USB_EPnControlTransferEnable</u>	Enables endpoint control transfers.
	<u>PLIB_USB_EPnDirectionDisable</u>	Disables the specified endpoint direction.
	<u>PLIB_USB_EPnHandshakeDisable</u>	Disables endpoint handshaking.
	<u>PLIB_USB_EPnHandshakeEnable</u>	Enables endpoint handshaking.
	<u>PLIB_USB_EPnIsStalled</u>	Tests whether the endpoint epValue is stalled.
	<u>PLIB_USB_EPnRxDisable</u>	Disables an endpoint's ability to process IN tokens.
	<u>PLIB_USB_EPnRxEnable</u>	Enables an endpoint to process IN tokens.
	<u>PLIB_USB_EPnRxSelect</u>	Selects receive capabilities of an endpoint.
	<u>PLIB_USB_EPnStallClear</u>	Clears an endpoint's stalled flag.
	<u>PLIB_USB_EPnTxDisable</u>	Disables an endpoint's ability to process OUT tokens.
	<u>PLIB_USB_EPnTxEnable</u>	Enables an endpoint to process OUT tokens.
	<u>PLIB_USB_EPnTxRxSelect</u>	Selects transmit and/or receive capabilities of an endpoint.
	<u>PLIB_USB_EPnTxSelect</u>	Selects transmit capabilities of an endpoint.

Error Interrupts Functions

	Name	Description
	<u>PLIB_USB_ErrorInterruptDisable</u>	Disables an error interrupt for the USB module.
	<u>PLIB_USB_ErrorInterruptEnable</u>	Enables an error interrupt for the USB module.
	<u>PLIB_USB_ErrorInterruptFlagAllGet</u>	Returns a logically ORed bit map of active error interrupt flags.
	<u>PLIB_USB_ErrorInterruptFlagClear</u>	Clears an error interrupt flag for the USB module.

	PLIB_USB_ErrorInterruptFlagGet	Tests an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagSet	Sets an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptIsEnabled	Returns true if interrupts are enabled.

External Transceiver Support Functions

	Name	Description
	PLIB_USB_I2CInterfaceForExtModuleDisable	Specifies external module(s) are controlled via dedicated pins.
	PLIB_USB_I2CInterfaceForExtModuleEnable	Specifies external module(s) are controlled via the I2C interface.
	PLIB_USB_TransceiverDisable	Disables the on-chip transceiver
	PLIB_USB_TransceiverEnable	Enables the on-chip transceiver.

Feature Existence Functions

	Name	Description
	PLIB_USB_ExistsActivityPending	Identifies whether the ActivityPending feature exists on the USB module.
	PLIB_USB_ExistsALL_Interrupt	Identifies whether the ALL_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsAutomaticSuspend	Identifies whether the AutomaticSuspend feature exists on the USB module.
	PLIB_USB_ExistsBDTBaseAddress	Identifies whether the BDTBaseAddress feature exists on the USB module.
	PLIB_USB_ExistsBDTFunctions	Identifies whether the BDTFunctions feature exists on the USB module.
	PLIB_USB_ExistsBufferFreeze	Identifies whether the BufferFreeze feature exists on the USB module.
	PLIB_USB_ExistsDeviceAddress	Identifies whether the DeviceAddress feature exists on the USB module.
	PLIB_USB_ExistsEP0LowSpeedConnect	Identifies whether the EP0LowSpeedConnect feature exists on the USB module.
	PLIB_USB_ExistsEP0NAKRetry	Identifies whether the EP0NAKRetry feature exists on the USB module.
	PLIB_USB_ExistsEPnRxEnable	Identifies whether the EPnRxEnableEnhanced feature exists on the USB module.
	PLIB_USB_ExistsEPnTxRx	Identifies whether the EPnTxRx feature exists on the USB module.
	PLIB_USB_ExistsERR_Interrupt	Identifies whether the ERR_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsERR_InterruptStatus	Identifies whether the ERR_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsEyePattern	Identifies whether the EyePattern feature exists on the USB module.
	PLIB_USB_ExistsFrameNumber	Identifies whether the FrameNumber feature exists on the USB module.
	PLIB_USB_ExistsGEN_Interrupt	Identifies whether the GEN_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsGEN_InterruptStatus	Identifies whether the GEN_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsHostBusyWithToken	Identifies whether the HostBusyWithToken feature exists on the USB module.
	PLIB_USB_ExistsHostGeneratesReset	Identifies whether the HostGeneratesReset feature exists on the USB module.
	PLIB_USB_ExistsLastDirection	Identifies whether the LastDirection feature exists on the USB module.
	PLIB_USB_ExistsLastEndpoint	Identifies whether the LastEndpoint feature exists on the USB module.
	PLIB_USB_ExistsLastPingPong	Identifies whether the LastPingPong feature exists on the USB module.
	PLIB_USB_ExistsLastTransactionDetails	Identifies whether the LastTransactionDetails feature exists on the USB module.
	PLIB_USB_ExistsLiveJState	Identifies whether the LiveJState feature exists on the USB module.
	PLIB_USB_ExistsLiveSingleEndedZero	Identifies whether the LiveSingleEndedZero feature exists on the USB module.

	PLIB_USB_ExistsModuleBusy	Identifies whether the ModuleBusy feature exists on the USB module.
	PLIB_USB_ExistsModulePower	Identifies whether the ModulePower feature exists on the USB module.
	PLIB_USB_ExistsNextTokenSpeed	Identifies whether the NextTokenSpeed feature exists on the USB module.
	PLIB_USB_ExistsOnChipPullup	Identifies whether the OnChipPullup feature exists on the USB module.
	PLIB_USB_ExistsOnChipTransceiver	Identifies whether the OnChipTransceiver feature exists on the USB module.
	PLIB_USB_ExistsOpModeSelect	Identifies whether the OpModeSelect feature exists on the USB module.
	PLIB_USB_ExistsOTG_ASessionValid	Identifies whether the OTG_ASessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_BSessionEnd	Identifies whether the OTG_BSessionEnd feature exists on the USB module.
	PLIB_USB_ExistsOTG_IDPinState	Identifies whether the OTG_IDPinState feature exists on the USB module.
	PLIB_USB_ExistsOTG_Interrupt	Identifies whether the OTG_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsOTG_InterruptStatus	Identifies whether the OTG_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsOTG_LineState	Identifies whether the OTG_LineState feature exists on the USB module.
	PLIB_USB_ExistsOTG_PullUpPullDown	Identifies whether the OTG_PullUpPullDown feature exists on the USB module.
	PLIB_USB_ExistsOTG_SessionValid	Identifies whether the OTG_SessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusCharge	Identifies whether the OTG_VbusCharge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusDischarge	Identifies whether the OTG_VbusDischarge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusPowerOnOff	Identifies whether the OTG_VbusPowerOnOff feature exists on the USB module.
	PLIB_USB_ExistsPacketTransfer	Identifies whether the PacketTransfer feature exists on the USB module.
	PLIB_USB_ExistsPingPongMode	Identifies whether the PingPongMode feature exists on the USB module.
	PLIB_USB_ExistsResumeSignaling	Identifies whether the ResumeSignaling feature exists on the USB module.
	PLIB_USB_ExistsSleepEntryGuard	Identifies whether the SleepEntryGuard feature exists on the USB module.
	PLIB_USB_ExistsSOFThreshold	Identifies whether the SOFThreshold feature exists on the USB module.
	PLIB_USB_ExistsSpeedControl	Identifies whether the SpeedControl feature exists on the USB module.
	PLIB_USB_ExistsStartOfFrames	Identifies whether the StartOfFrames feature exists on the USB module.
	PLIB_USB_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USB module.
	PLIB_USB_ExistsSuspend	Identifies whether the Suspend feature exists on the USB module.
	PLIB_USB_ExistsTokenEP	Identifies whether the TokenEP feature exists on the USB module.
	PLIB_USB_ExistsTokenPID	Identifies whether the TokenPID feature exists on the USB module.
	PLIB_USB_ExistsUOEMonitor	Identifies whether the UOEMonitor feature exists on the USB module.

Host Functions

	Name	Description
	PLIB_USB_IsBusyWithToken	Indicates whether there is a token being executed by the USB module as Host.
	PLIB_USB_SOFDisable	Disables the automatic generation of the SOF token.
	PLIB_USB_SOFEnable	Enables the automatic generation of the SOF token every 1 ms.
	PLIB_USB_SOFThresholdGet	Returns the Start-of-Frame (SOF) Count bits.

	PLIB_USB_SOFThresholdSet	Sets the Start-of-Frame (SOF) threshold value.
	PLIB_USB-TokenEPGet	Returns the specified Endpoint address.
	PLIB_USB-TokenEPSet	Sets the Endpoint address for a host transaction.
	PLIB_USB-TokenPIDGet	Returns the token transaction type.
	PLIB_USB-TokenPIDSet	Sets the token transaction type to pidValue.
	PLIB_USB-TokenSpeedSelect	Selects low speed or full speed for subsequent token executions.

Interrupts Functions

	Name	Description
	PLIB_USB_InterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USB_InterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USB_InterruptEnableGet	Returns the enable/disable status of general USB module interrupts
	PLIB_USB_InterruptFlagAllGet	Returns a logically ORed bit map of active general USB interrupt flags.
	PLIB_USB_InterruptFlagClear	Clears a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagGet	Tests a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagSet	Sets a general interrupt flag for the USB module.
	PLIB_USB_InterruptIsEnabled	Returns true if interrupts are enabled.

Last Transaction Status Functions

	Name	Description
	PLIB_USB_LastTransactionDetailsGet	Returns the details of the last completed transaction.
	PLIB_USB_LastTransactionDirectionGet	Indicates the direction of the last transaction.
	PLIB_USB_LastTransactionEndPtGet	Returns the endpoint number of the last USB transfer.
	PLIB_USB_LastTransactionPingPongStateGet	Indicates whether the last transaction was to an EVEN buffer or an ODD buffer.

On-The-Go (OTG) Functions

	Name	Description
	PLIB_USB_OTG_BSessionHasEnded	Returns the status of the B-Session End Indicator bit.
	PLIB_USB_OTG_IDPinStateIsTypeA	Returns the ID Pin state.
	PLIB_USB_OTG_LineStateIsStable	Returns the status of the Line Stable Indicator bit.
	PLIB_USB_OTG_PullUpPullDownSetup	Enables or disables pull-up and pull-down resistors.
	PLIB_USB_OTG_SessionValid	Returns the status of the Session Valid Indicator bit.
	PLIB_USB_OTG_VBusChargeDisable	Disables VBUS line charge.
	PLIB_USB_OTG_VBusChargeEnable	Enables the VBUS line to be charged through a pull-up resistor.
	PLIB_USB_OTG_VBusChargeTo3V	Sets the VBUS line to charge to 3.3V.
	PLIB_USB_OTG_VBusChargeTo5V	Sets the VBUS line to charge to 5V.
	PLIB_USB_OTG_VBusDischargeDisable	Disables VBUS line discharge.
	PLIB_USB_OTG_VBusDischargeEnable	Enables VBUS line to be discharged through a resistor.
	PLIB_USB_OTG_VBusPowerOff	Turns off power on the VBUS Line.
	PLIB_USB_OTG_VBusPowerOn	Turns on power for the VBUS line.
	PLIB_USB_OTG_VBusValid	Returns the status of the A-VBUS valid indicator.

OTG Interrupts Functions

	Name	Description
	PLIB_USB_OTG_InterruptDisable	Disables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptEnable	Enables a USB On-The-Go (OTG) Interrupt for the USB module.

	PLIB_USB_OTG_InterruptFlagClear	Clears a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagGet	Tests a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagSet	Sets a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptIsEnabled	Returns whether or not interrupts are enabled.

Other Functions

	Name	Description
	PLIB_USB_ModuleIsBusy	Indicates if the USB module is not ready to be enabled.
	PLIB_USB_PingPongFreeze	Resets all Ping-Pong buffer pointers to even buffers.
	PLIB_USB_PingPongReset	Resets the USB peripheral internal Ping-Pong indicator to point to even buffers.
	PLIB_USB_PingPongUnfreeze	Enables Ping-Pong buffering.
	PLIB_USB_TokenSend	Sends token to the specified address.

Test Support Functions

	Name	Description
	PLIB_USB_EyePatternDisable	Disables the USB eye pattern test.
	PLIB_USB_EyePatternEnable	Enables USB eye pattern test.

USB Activity Functions

	Name	Description
	PLIB_USB_ActivityPending	Returns whether or not USB activity is pending.
	PLIB_USB_FrameNumberGet	Returns the USB frame number.
	PLIB_USB_JStateIsActive	Live differential receiver J State flag.
	PLIB_USB_PacketTransferDisable	Disables the Serial Interface Engine (SIE).
	PLIB_USB_PacketTransferEnable	Re-enables the Serial Interface Engine (SIE), allowing token and packet processing.
	PLIB_USB_PacketTransferIsDisabled	Indicates that a setup token has been received from the Host and that token/packet processing is disabled.
	PLIB_USB_SE0InProgress	Returns whether a single-ended zero event is in progress.

USB Bus Signaling Functions

	Name	Description
	PLIB_USB_ResetSignalDisable	Disables reset signalling on the USB bus.
	PLIB_USB_ResetSignalEnable	Enables reset signalling on the USB bus.
	PLIB_USB_ResumeSignalingDisable	Disables resume signaling.
	PLIB_USB_ResumeSignalingEnable	Enables resume signaling.

USB Setup Functions

	Name	Description
	PLIB_USB_AllInterruptEnable	Configures the USB peripheral general interrupts, error interrupts and OTG interrupts.
	PLIB_USB_AutoSuspendDisable	Disables USB OTG Auto-suspend mode.
	PLIB_USB_AutoSuspendEnable	Enables USB Auto-suspend mode.
	PLIB_USB_DeviceAddressGet	Returns the address of the USB module in Device mode.
	PLIB_USB_DeviceAddressSet	Sets the USB Device's address.
	PLIB_USB_Disable	Disables (powers down) the USB module.
	PLIB_USB_Enable	Enables (powers up) the USB module.
	PLIB_USB_FullSpeedDisable	Forces the USB module to operate at low speed.

	PLIB_USB_FullSpeedEnable	Enables the USB to operate at full speed.
	PLIB_USB_OnChipPullUpDisable	Disables on-chip pull-ups.
	PLIB_USB_OnChipPullUpEnable	Enables on-chip pull-ups.
	PLIB_USB_OperatingModeSelect	Selects the operating mode of the USB module.
	PLIB_USB_PingPongModeGet	Returns the Ping-Pong Configuration setting.
	PLIB_USB_PingPongModeSelect	Selects the Ping-Pong Configuration setting.
	PLIB_USB_SleepGuardDisable	This function disables Sleep Guard. Entry into Sleep mode is immediate.
	PLIB_USB_SleepGuardEnable	Entry into Sleep mode is blocked if bus activity is detected or if an interrupt is pending.
	PLIB_USB_StopInIdleDisable	Allows the USB module to continue operation when the device enters Idle mode.
	PLIB_USB_StopInIdleEnable	Enables USB module operation to stop when the device enters Idle mode.
	PLIB_USB_SuspendDisable	Disables USB OTG Suspend mode.
	PLIB_USB_SuspendEnable	Enables USB Suspend mode.
	PLIB_USB_UOEMonitorDisable	Disables the OE signal output.
	PLIB_USB_UOEMonitorEnable	Enables the OE signal output.

VBus Support Functions

	Name	Description
	PLIB_USB_ExternalComparatorMode2Pin	Sets the 2-pin input configuration for VBUS comparators.
	PLIB_USB_ExternalComparatorMode3Pin	Sets the 3-pin input configuration for VBUS Comparators.
	PLIB_USB_PWMCounterDisable	Disables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMCounterEnable	Enables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMDisable	Disables the PWM Generator.
	PLIB_USB_PWMEnable	Enables the PWM Generator.
	PLIB_USB_PWMPolarityActiveLow	Sets the PWM output to active-high and resets low.
	PLIB_USB_PWMPolarityActiveHigh	Sets the PWM output to active-low and resets high.
	PLIB_USB_VBoostDisable	Disables the On-Chip 5V Boost Regulator Circuit Disabled bit.
	PLIB_USB_VBoostEnable	Enables the On-Chip 5V Boost Regulator Circuit Enabled bit.
	PLIB_USB_VBUSComparatorDisable	Disables the on-chip VBUS Comparator.
	PLIB_USB_VBUSComparatorEnable	Enables the on-chip VBUS Comparator.
	PLIB_USB_VBUSPullUpDisable	Disables the pull-up on the VBUS pin.
	PLIB_USB_VBUSPullUpEnable	Enables the pull-up on the VBUS pin.

Description

This library provides the low level abstraction of the USB Peripheral on Microchip family of microcontrollers with a convenient C language interface.

5.6.26.6.1 USB Setup Functions

5.6.26.6.1.1 PLIB_USB_AllInterruptEnable Function

C

```
void PLIB_USB_AllInterruptEnable(
    USB_MODULE_ID index,
    USB_INTERRUPTS usbInterruptsFlag,
    USB_ERROR_INTERRUPTS usbErrorInterruptsFlag,
```

```
    USB_OTG_INTERRUPTS otgInterruptFlag  
);
```

Description

This function configures the USB peripheral general interrupts, error interrupts and OTG interrupts.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
usbInterruptsFlag	General interrupts to be configured
usbErrorInterruptsFlag	USB Error interrupts to be configured
otgInterruptFlag	OTG interrupts to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsALL_Interrupt](#) in your application to determine whether this feature is available.

Example

```
// This code snippet disables all OTG interrupts, disables  
// the SOF interrupt and enables all error interrupts.  
USB_OTG_INTERRUPTS otgInterruptEnables = ~USB_OTG_INT_ALL ;  
USB_INTERRUPTS generalInterruptEnables = USB_INT_ALL & ~USB_INT_SOF ;  
USB_ERROR_INTERRUPTS errorInterruptEnables = USB_ERR_INT_ALL ;  
  
PLIB_USB_AllInterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS usbInterruptsFlag,  
    USB_ERROR_INTERRUPTS usbErrorInterruptsFlag,  
    USB_OTG_INTERRUPTS otgInterruptFlag);
```

5.6.26.6.1.2 PLIB_USB_AutoSuspendDisable Function**C**

```
void PLIB_USB_AutoSuspendDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables USB OTG Auto-suspend mode. The USB OTG module will operate normally and does not automatically suspend upon entry to Sleep mode. Software must use [PLIB_USB_SuspendEnable](#) to suspend the module, including the USB 48 MHz clock

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 devices.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsAutomaticSuspend](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_AutoSuspendDisable(MY_USB_INSTANCE);
```

5.6.26.6.1.3 PLIB_USB_AutoSuspendEnable Function**C**

```
void PLIB_USB_AutoSuspendEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables USB Auto-suspend mode. The USB module automatically suspends upon entry to Sleep mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 devices.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsAutomaticSuspend](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_AutoSuspendEnable(MY_USB_INSTANCE);
```

5.6.26.6.1.4 PLIB_USB_DeviceAddressGet Function**C**

```
uint8_t PLIB_USB_DeviceAddressGet(  
    USB_MODULE_ID index  
);
```

Description

This function returns the address of the USB module in Device mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

Device Address

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_USB_ExistsDeviceAddress` in your application to determine whether this feature is available.

Example

```
myUSBAddress = PLIB_USB_DeviceAddressGet(MY_USB_INSTANCE);
```

5.6.26.6.1.5 PLIB_USB_DeviceAddressSet Function**C**

```
void PLIB_USB_DeviceAddressSet(  
    USB_MODULE_ID index,  
    uint8_t address  
);
```

Description

This function sets the USB Device's address as part of enumeration.

Preconditions

USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
address	USB address

Returns

None.

Remarks

Not applicable for PIC18 devices, since USB Host functionality is not available on PIC18 devices.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_USB_ExistsDeviceAddress` in your application to determine whether this feature is available.

Example

```
uint8_t myUSBAddress = . . . ;  
PLIB_USB_DeviceAddressSet(MY_USB_INSTANCE, myUSBAddress);
```

5.6.26.6.1.6 PLIB_USB_Disable Function**C**

```
void PLIB_USB_Disable(  
    USB_MODULE_ID index  
);
```

Description

This function disables (powers down) the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

For PIC24/dsPIC33 and PIC32 devices, the USB module must be in Device mode before the USB module is powered down.

For PIC32 devices, all reads or writes to module registers after powering down the module will be invalid until [PLIB_USB_ModuleIsBusy](#) (MY_USB_INSTANCE) == false.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsModulePower](#) in your application to determine whether this feature is available.

Example

```
#if ( defined(__PIC24E__) || defined(__dsPIC33E__) || defined(__PIC32MX__) )
    // Disable Host, Device, or OTG before powering down
    PLIB_USB_OperatingModeSelect( MY_USB_INSTANCE, USB_OPMODE_NONE );
#endif
    // Turn off USB
    PLIB_USB_Disable(MY_USB_INSTANCE);
#ifdef __PIC32MX__
    // For PIC32, wait until module is no longer busy before trying to
    // access any USB module registers.
    while ( PLIB_USB_ModuleIsBusy (MY_USB_INSTANCE) )
    {
        //wait
    }
#endif
    // Can now read or modify USB module status
```

5.6.26.6.1.7 PLIB_USB_Enable Function**C**

```
void PLIB_USB_Enable(
    USB_MODULE_ID index
);
```

Description

This function enables (powers up) the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not valid on PIC18 devices until a valid clock source is provided to the USB module.

See also [PLIB_USB_ModuleIsBusy](#).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_USB_ExistsModulePower` your application to determine whether this feature is available.

Example

```
// Complete Needed setup for the module  
PLIB_USB_Enable(MY_USB_INSTANCE);
```

5.6.26.6.1.8 PLIB_USB_FullSpeedDisable Function**C**

```
void PLIB_USB_FullSpeedDisable(  
    USB_MODULE_ID index  
);
```

Description

This function forces the USB module to operate at low speed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

For PIC18 devices, use only before the USB module is enabled to select low speed. In addition, an input clock of 6 MHz instead of 48 MHz is required.

For all others devices: Host mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsSpeedControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_FullSpeedDisable(MY_USB_INSTANCE);
```

5.6.26.6.1.9 PLIB_USB_FullSpeedEnable Function**C**

```
void PLIB_USB_FullSpeedEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the USB to operate at full speed.

Preconditions

Use only before the USB module is enabled by calling [PLIB_USB_Enable](#).

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

For PIC18 devices, use only before the USB module is enabled to select low speed. In addition, an input clock of 6 MHz instead of 48 MHz is required.

For all others devices: Host mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsSpeedControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_FullSpeedEnable(MY_USB_INSTANCE);
```

5.6.26.6.1.10 PLIB_USB_OnChipPullUpDisable Function**C**

```
void PLIB_USB_OnChipPullUpDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables on-chip pull-ups.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

PIC18 devices only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOnChipPullup](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OnChipPullUpDisable(MY_USB_INSTANCE);
```

5.6.26.6.1.11 PLIB_USB_OnChipPullUpEnable Function**C**

```
void PLIB_USB_OnChipPullUpEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables on-chip pull-ups. Pull-up on D+ in Full-Speed mode. Pull-up on D- in Low-Speed mode.

Preconditions

Use only before the USB module is enabled by calling [PLIB_USB_Enable](#).

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

PIC18 devices only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOnChipPullup](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OnChipPullUpEnable(MY_USB_INSTANCE);
```

5.6.26.6.1.12 PLIB_USB_OperatingModeSelect Function**C**

```
void PLIB_USB_OperatingModeSelect(  
    USB_MODULE_ID index,  
    USB_OPMODES opMode  
);
```

Description

This function selects the operating mode of the USB module, either Host, Device, or OTG.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
opMode	Selected operating mode: USB_OPMODE_DEVICE, USB_OPMODE_HOST, or USB_OPMODE_OTG

Returns

None.

Remarks

Not needed on PIC18 devices, since only UPB_OPMODE_DEVICE is supported and there are no other operating modes. Use USB_OPMODE_NONE to disable all possible functions.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOpModeSelect](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OperatingModeSelect( MY_USB_INSTANCE, USB_OPMODE_DEVICE );
```

5.6.26.6.1.13 PLIB_USB_PingPongModeGet Function**C**

```
USB_PING_PONG_MODE PLIB_USB_PingPongModeGet(  
    USB_MODULE_ID index  
);
```

Description

This function returns the Ping-Pong Configuration setting.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

Ping-Pong Mode - One of USB_PING_PONG__ALL_BUT_EP0, USB_PING_PONG__FULL_PING_PONG, USB_PING_PONG__EP0_OUT_ONLY, USB_PING_PONG__NO_PING_PONG

Remarks

For PIC18F and PIC24F devices only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsPingPongMode](#) in your application to determine whether this feature is available.

Example

```
ppConfig = PLIB_USB_PingPongModeGet(MY_USB_INSTANCE);
```

5.6.26.6.1.14 PLIB_USB_PingPongModeSelect Function**C**

```
void PLIB_USB_PingPongModeSelect(  
    USB_MODULE_ID index,  
    USB_PING_PONG_MODE ppConfig  
);
```

Description

This function selects the Ping-Pong Configuration setting.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
ppConfig	Ping-Pong configuration selection. One of USB_PING_PONG__ALL_BUT_EP0, USB_PING_PONG__FULL_PING_PONG, USB_PING_PONG__EP0_OUT_ONLY, USB_PING_PONG__NO_PING_PONG

Returns

None.

Remarks

For PIC18F and PIC24F devices only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsPingPongMode](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_PingPongModeSelect(MY_USB_INSTANCE, USB_PING_PONG__ALL_BUT_EP0);
```

5.6.26.6.1.15 PLIB_USB_SleepGuardDisable Function

C

```
void PLIB_USB_SleepGuardDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables Sleep Guard. Entry into Sleep mode is immediate.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18, dsPIC or PIC24 devices. Not available on all PIC32 devices. Refer to the specific device data sheet for details.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsSleepEntryGuard](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_SleepGuardDisable(MY_USB_INSTANCE);
```

5.6.26.6.1.16 PLIB_USB_SleepGuardEnable Function

C

```
void PLIB_USB_SleepGuardEnable(  
    USB_MODULE_ID index  
);
```

Description

This function block entry into Sleep mode if bus activity is detected or if an interrupt is pending.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18, dsPIC, or PIC24 devices. Not available on all PIC32 devices. Refer to the specific device data sheet for details. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsSleepEntryGuard](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_SleepGuardEnable(MY_USB_INSTANCE);
```

5.6.26.6.1.17 PLIB_USB_StopInIdleDisable Function**C**

```
void PLIB_USB_StopInIdleDisable(  
    USB_MODULE_ID index  
);
```

Description

This function allows the USB module to continue operation when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_StopInIdleDisable(MY_USB_INSTANCE);
```

5.6.26.6.1.18 PLIB_USB_StopInIdleEnable Function**C**

```
void PLIB_USB_StopInIdleEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables USB module operation to stop when the device enters Idle mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsStopInIdle](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_StopInIdleEnable(MY_USB_INSTANCE);
```

5.6.26.6.1.19 PLIB_USB_SuspendDisable Function**C**

```
void PLIB_USB_SuspendDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables USB OTG Suspend mode. The USB OTG module will operate normally.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 devices.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsSuspend](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_SuspendDisable(MY_USB_INSTANCE);
```

5.6.26.6.1.20 PLIB_USB_SuspendEnable Function**C**

```
void PLIB_USB_SuspendEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables USB Suspend mode. The 48 MHz USB clock will be gated off. The transceiver is placed in a low-power state.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 devices.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsSuspend](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_SuspendEnable(MY_USB_INSTANCE);
```

5.6.26.6.1.21 PLIB_USB_UOEMonitorDisable Function**C**

```
void PLIB_USB_UOEMonitorDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables the OE signal output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsUOEMonitor](#) in your application to determine whether this feature is available.

Example

```
//Disable the OE output.  
PLIB_USB_UOEMonitorDisable(MY_USB_INSTANCE);
```

5.6.26.6.1.22 PLIB_USB_UOEMonitorEnable Function**C**

```
void PLIB_USB_UOEMonitorEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the OE signal output.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsUOEMonitor](#) in your application to determine whether this feature is available.

Example

```
//Enable the OE output.  
PLIB_USB_UOEMonitorEnable(MY_USB_INSTANCE);
```

5.6.26.6.2 Buffer Descriptor Table Functions

5.6.26.6.2.1 PLIB_USB_BDTBaseAddressGet Function

C

```
void* PLIB_USB_BDTBaseAddressGet(  
    USB_MODULE_ID index  
);
```

Description

This function returns the base address of the Buffer Descriptor Table.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Always set to 0x400 for PIC18 devices. Must be set for dsPIC33E/PIC24E, PIC24F, and PIC32 devices using [PLIB_USB_BDTBaseAddressSet](#).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTBaseAddress](#) in your application to determine whether this feature is available.

Example

```
void * pMyBDT;  
pMyBDT = PLIB_USB_BDTBaseAddressGet(MY_USB_INSTANCE);
```

5.6.26.6.2.2 PLIB_USB_BDTBaseAddressSet Function

C

```
void PLIB_USB_BDTBaseAddressSet(  
    USB_MODULE_ID index,  
    void* address  
);
```

Description

This function sets the base address for the Buffer Descriptor Table. This function is only available on PIC24 and PIC32 devices.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
address	Physical memory address in RAM of Buffer Descriptor Table

Returns

None.

Remarks

Available on dsPIC33E/PIC24E, PIC24F, and PIC32. For PIC18 devices the address is fixed at 0x400. The address of the Buffer Descriptor Table must be 512 byte-aligned. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTBaseAddress](#) in your application to determine whether this feature is available.

Example

```
#if defined(__PIC32MX__)
// For PIC32
PLIB_USB_BDTBaseAddressSet(MY_USB_INSTANCE, (void *)((uint32_t)KVA_TO_PA(&myBDT)) );
#else
// Everybody else
PLIB_USB_BDTBaseAddressSet(MY_USB_INSTANCE, (void *)&myBDT );
#endif
```

5.6.26.6.2.3 PLIB_USB_BufferAddressGet Function**C**

```
void* PLIB_USB_BufferAddressGet(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function gets the memory address of an endpoint buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint Value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

Buffer address in memory.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.4 PLIB_USB_BufferAddressSet Function****C**

```
void PLIB_USB_BufferAddressSet(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong,
    void* bufferAddress
);
```

Description

This function sets the endpoint buffer address.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferAddress	address in memory of endpoint transmit or receive buffer

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.5 PLIB_USB_BufferAllCancelReleaseToUSB Function****C**

```
void PLIB_USB_BufferAllCancelReleaseToUSB(
    USB_MODULE_ID index,
    void * pBDT,
    USB_PING_PONG_MODE ppMode,
    int nEndpoints
);
```

Description

This function cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pBDT	Pointer to the Buffer Descriptor Table
ppMode	Buffer Descriptor Table Ping-Pong mode
nEndpoints	Number of endpoints in the Buffer-Descriptor table

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example

```
//Cancel all buffer releases to USB.
//BDT has 3 Endpoints.

PLIB_USB_BufferAllCancelReleaseToUSB(MY_USB_INSTANCE, pBDT,
                                       USB_PING_PONG_NO_PING_PONG, 3);
```

5.6.26.6.2.6 PLIB_USB_BufferByteCountGet Function**C**

```
uint16_t PLIB_USB_BufferByteCountGet(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function returns the endpoint buffer byte count, the actual number of bytes transmitted or received.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

Endpoint buffer byte count.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.7 PLIB_USB_BufferByteCountSet Function**

C

```
void PLIB_USB_BufferByteCountSet(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong,
    uint16_t bufferByteCount
);
```

Description

This function sets the number of bytes to be transmitted or the maximum number of bytes to be received.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferByteCount	number of bytes to be transmitted or the maximum number of bytes to be received

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.8 PLIB_USB_BufferCancelReleaseToUSB Function**

C

```
void PLIB_USB_BufferCancelReleaseToUSB(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
```

```
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function cancels the release of the endpoint buffer by software, allowing software to again access the buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.9 PLIB_USB_BufferClearAll Function****C**

```
void PLIB_USB_BufferClearAll(
    USB_MODULE_ID index,
    void * pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function clears (zeros out) the entries in the Buffer Descriptor Table.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX

bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example

5.6.26.6.2.10 PLIB_USB_BufferClearAllDTSEnable Function

C

```
void PLIB_USB_BufferClearAllDTSEnable(  
    USB_MODULE_ID index,  
    void * pBDT,  
    USB_PING_PONG_MODE ppMode,  
    uint8_t epValue,  
    USB_BUFFER_DIRECTION bufferDirection  
);
```

Description

This function clears the endpoint descriptor entry and enables data toggle synchronization.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pBDT	pointer to Buffer Descriptor Table
pingpong	Ping-Pong mode.
epvalue	Endpoint to be be affected
bufferDirection	Endpoint direction

Returns

None.

Remarks

None.

Example

```
//Clear endpoint 6 buffer descriptor transmit entry and  
//enable data toggle synchronization.  
  
PLIB_USB_BufferClearAllDTSEnable ( MY_USB_INSTANCE, pBDT,  
    USB_PING_PONG_NO_PING_PONG, 6, USB_BUFFER_TX);
```

5.6.26.6.2.11 PLIB_USB_BufferDataToggleGet Function

C

```
USB_BUFFER_DATA01 PLIB_USB_BufferDataToggleGet(  
    USB_MODULE_ID index,  
    void* pBDT,
```

```

    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);

```

Description

This function returns data synchronization (DATA0 or DATA1) for the endpoint buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

Data Toggle value, USB_BUFFER_DATA0 or USB_BUFFER_DATA1, for the buffer

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.12 PLIB_USB_BufferDataToggleSelect Function****C**

```

void PLIB_USB_BufferDataToggleSelect(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong,
    USB_BUFFER_DATA01 bufferData01
);

```

Description

This function sets the endpoint buffer to DATA0 or DATA1.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferData01	USB_BUFFER_DATA0 or USB_BUFFER_DATA1

Returns

None.

Remarks

See [PLIB_USB_BufferDataToggleGet](#) to determine the received data toggle setting. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.13 PLIB_USB_BufferDataToggleSyncDisable Function**

C

```
void PLIB_USB_BufferDataToggleSyncDisable(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function disables DATA0/DATA1 synchronization between the device and host.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.14 PLIB_USB_BufferDataToggleSyncEnable Function**

C

```
void PLIB_USB_BufferDataToggleSyncEnable(
```

```
USB_MODULE_ID index,  
void* pBDT,  
USB_PING_PONG_MODE ppMode,  
uint8_t epValue,  
USB_BUFFER_DIRECTION bufferDirection,  
USB_BUFFER_PING_PONG bufferPingPong  
);
```

Description

This function enables DATA0/DATA1 synchronization between the device and host.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.15 PLIB_USB_BufferEP0RxStatusInitialize Function****C**

```
void PLIB_USB_BufferEP0RxStatusInitialize(  
    USB_MODULE_ID index,  
    void* pBDT,  
    USB_PING_PONG_MODE ppMode,  
    USB_BUFFER_PING_PONG pingpong,  
    uint16_t bufferByteCount  
);
```

Description

This function initializes the Endpoint 0 RX endpoint buffer descriptors. This function will clear the Endpoint 0 RX Buffer Descriptor status field, load the endpoint size and release the buffer to the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pBDT	Pointer to Buffer Descriptor Table
pingpong	Ping-Pong mode

bufferByteCount	size of the EP0 RX buffer in bytes

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example

```
//Initialize EP0 RX even buffer descriptor and release back to
//USB. Buffer size is 64
```

```
PLIB_USB_BufferEP0RxStatusInitialize ( MY_USB_INSTANCE, pBDT,
    USB_PING_PONG_NO_PING_PONG, USB_BUFFER_EVEN, 64);
```

5.6.26.6.2.16 PLIB_USB_BufferIndexGet Function**C**

```
uint8_t PLIB_USB_BufferIndexGet(
    USB_MODULE_ID index,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function gets the Buffer Descriptor Table index for a buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

Buffer index into the Buffer Descriptor Table.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.17 PLIB_USB_BufferPIDBitsClear Function****C**

```
void PLIB_USB_BufferPIDBitsClear(
    USB_MODULE_ID index,
```

```

void* pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong
);

```

Description

This function clears the Buffer Status bits in the Buffer Descriptor Table.

Preconditions

The associated buffer must have been released by the USB module (i.e., [PLIB_USB_BufferReleasedToSW](#) returns 'true').

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

None.

Remarks

Call [PLIB_USB_BufferPIDBitsClear](#) before setting buffer control bits. This is equivalent to: [PLIB_USB_BufferCancelReleaseToUSB\(...\)](#) [PLIB_USB_BufferDataToggleSelect\(...,USB_BUFFER_DATA0\)](#) [PLIB_USB_BufferDataToggleSyncDisable\(...\)](#) [PLIB_USB_BufferStallDisable\(...\)](#) This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.18 PLIB_USB_BufferPIDGet Function****C**

```

uint8_t PLIB_USB_BufferPIDGet(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);

```

Description

This function returns the token packet ID (PID) from the endpoint buffer status.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest

pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

Endpoint buffer packet ID (PID).

Remarks

There is no equivalent "Set" routine, since this field is read-only in the buffer status register within the Buffer Descriptor Table. It is set when the buffer has been transmitted or received by the USB module and the usbOwnsBuffer field has been cleared by the USB module, releasing the buffer for software access. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.19 PLIB_USB_BufferReleasedToSW Function****C**

```
bool PLIB_USB_BufferReleasedToSW(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function returns the boolean flag value of 'true' when the buffer has been released by the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

- true - The buffer has been released by hardware
- false - The buffer is still controlled by hardware

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.20 PLIB_USB_BufferReleaseToUSB Function****C**

```

void PLIB_USB_BufferReleaseToUSB(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);

```

Description

This function releases the endpoint buffer by software, allowing the USB module access to buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example**5.6.26.6.2.21 PLIB_USB_BufferSchedule Function****C**

```

void PLIB_USB_BufferSchedule(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong,
    void * bufferAddress,
    int16_t bufferByteCount,
    USB_BUFFER_SCHEDULE_DATA01 bufferData01
);

```

Description

This function sets the endpoint descriptor buffer address, sets the [send](#)/receive byte count, and then hands over the buffer to the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferAddress	Address of the application buffer
bufferByteCount	Send or expected receive byte count
bufferData01	USB_BUFFER_SET_DATA0, USB_BUFFER_SET_DATA1, or USB_BUFFER_DONTCHANGE (The last choice leaves the existing DATA0/1 value of the buffer alone.)

Returns

None.

Remarks

This function does the work of three other functions: [PLIB_USB_BufferAddressSet](#), [PLIB_USB_BufferByteCountSet](#), [PLIB_USB_BufferReleaseToUSB](#)

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

5.6.26.6.2.22 PLIB_USB_BufferStallDisable Function**C**

```
void PLIB_USB_BufferStallDisable(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function disables STALL handshaking for the associated endpoint buffer.

Preconditions

The associated buffer must have been released by the USB module (i.e., [PLIB_USB_BufferReleasedToSW](#) returns 'true').

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX

bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

None.

Remarks

Release of a STALL handshake for the buffer is done by hardware when the host sends a SETUP token to the associated endpoint or resets the USB module. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available. .

Example**5.6.26.6.2.23 PLIB_USB_BufferStallEnable Function****C**

```
void PLIB_USB_BufferStallEnable(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function enables STALL handshaking for the associated endpoint buffer.

Preconditions

The associated buffer must have been released by the USB module (i.e. [PLIB_USB_BufferReleasedToSW](#) returns 'true').

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

None.

Remarks

Release of a STALL handshake for the buffer is done by hardware when the host sends a SETUP token to the associated endpoint or resets the USB module. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example

5.6.26.6.2.24 PLIB_USB_BufferStallGet Function

C

```
bool PLIB_USB_BufferStallGet(
    USB_MODULE_ID index,
    void* pBDT,
    USB_PING_PONG_MODE ppMode,
    uint8_t epValue,
    USB_BUFFER_DIRECTION bufferDirection,
    USB_BUFFER_PING_PONG bufferPingPong
);
```

Description

This function returns the buffer stall status for an endpoint/direction/ping-pong.

Preconditions

the associated buffer must have been released by the USB module (i.e., [PLIB_USB_BufferReleasedToSW](#) returns 'true').

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Returns

- true - Buffer stall is enabled
- false - Buffer stall is not enabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBDTFunctions](#) in your application to determine whether this feature is available.

Example

5.6.26.6.3 Endpoints Functions

5.6.26.6.3.1 PLIB_USB_EP0HostSetup Function

C

```
void PLIB_USB_EP0HostSetup(
    USB_MODULE_ID index
);
```

Description

This function configures endpoint 0 for typical host operation. Control transfers are enable. Transmit and Receive is enabled. Handshaking is enabled. Low Speed connection is disabled. NAK retry is disabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_Interrupt](#) in your application to determine whether this feature is available.

Example

```
// Configure endpoint 0 for host operation  
PLBIB_USB_EP0HostSetup(USB_MODULE_ID index);
```

5.6.26.6.3.2 PLIB_USB_EP0LSDirectConnectDisable Function**C**

```
void PLIB_USB_EP0LSDirectConnectDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables direct connection to a low-speed device for Endpoint 0.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Host mode and U1EP0 only. Not applicable for PIC18 devices, since USB Host functionality is not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEP0LowSpeedConnect](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EP0LSDirectConnectDisable(MY_USB_INSTANCE);
```

5.6.26.6.3.3 PLIB_USB_EP0LSDirectConnectEnable Function**C**

```
void PLIB_USB_EP0LSDirectConnectEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables direct connection to a low-speed device for Endpoint 0.

Preconditions

USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Host Mode and U1EP0 only. Not applicable for PIC18 devices, since USB Host functionality is not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEP0LowSpeedConnect](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EP0LSDirectConnectEnable(MY_USB_INSTANCE);
```

5.6.26.6.3.4 PLIB_USB_EP0NakRetryDisable Function**C**

```
void PLIB_USB_EP0NakRetryDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables retrying of NAKed transactions.

Preconditions

The USB module must be in Host or OTG modes.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Host/OTG only. Not applicable for PIC18 devices, since USB Host and OTG functionality is not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEP0NAKRetry](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EP0NakRetryDisable(MY_USB_INSTANCE);
```

5.6.26.6.3.5 PLIB_USB_EP0NakRetryEnable Function**C**

```
void PLIB_USB_EP0NakRetryEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables retrying NAK'd transactions for Endpoint 0.

Preconditions

The USB module must be in Host or OTG modes.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Host/OTG only. Not applicable for PIC18 devices, since USB Host or OTG functionality is not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEP0NAKRetry](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EP0NakRetryEnable(MY_USB_INSTANCE);
```

5.6.26.6.3.6 PLIB_USB_EPnAttributesClear Function**C**

```
void PLIB_USB_EPnAttributesClear(  
    USB_MODULE_ID index,  
    uint8_t epValue  
);
```

Description

Clears the set attributes of the specified endpoint. The endpoint transmit receive, handshake and setup packet handling capability is disabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Returns

None.

Remarks

None.

Example

```
// This clears up the endpoint 0 attributes and thus disables  
// the endpoints  
PLIB_USB_EPnAttributesClear(MY_USB_INSTANCE, 0);
```

5.6.26.6.3.7 PLIB_USB_EPnAttributesSet Function

C

```
void PLIB_USB_EPnAttributesSet(  
    USB_MODULE_ID index,  
    uint8_t epValue,  
    int direction,  
    bool isControl,  
    bool handshake  
);
```

Description

Configures attributes of the endpoint such as direction, handshake capability and direction. If the isControl flag is true, then the direction and handshake parameters are ignored and the endpoint is configured for control transfers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
direction	Endpoint direction, if 1 then RX and if 0 then TX.
isControl	If true endpoint is configured for control transfers.
handshake	If true, then handshake is enabled on the endpoint else it is disabled.

Returns

None.

Remarks

None.

Example

```
// This enables endpoint 0 for control transfers.  
PLIB_USB_EPnAttributesSet(MY_USB_INSTANCE, 0, 0, true, true);  
// This enables endpoint 2 for non control transfer, direction  
// is RX and handshake enable.  
PLIB_USB_EPnAttributesSet(MY_USB_INSTANCE, 2, 1, false, true);
```

5.6.26.6.3.8 PLIB_USB_EPnControlTransferDisable Function

C

```
void PLIB_USB_EPnControlTransferDisable(  
    USB_MODULE_ID index,  
    uint8_t epValue  
);
```

Description

This function disables endpoint control transfers when endpoint transmit and endpoint receive are both enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EPnControlTransferDisable(MY_USB_INSTANCE, someEP);
```

5.6.26.6.3.9 PLIB_USB_EPnControlTransferEnable Function**C**

```
void PLIB_USB_EPnControlTransferEnable(  
    USB_MODULE_ID index,  
    uint8_t epValue  
);
```

Description

This function enables endpoint control transfers when endpoint transmit and endpoint receive are both enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EPnControlTransferEnable(MY_USB_INSTANCE, someEP);
```

5.6.26.6.3.10 PLIB_USB_EPnDirectionDisable Function**C**

```
void PLIB_USB_EPnDirectionDisable(  
    USB_MODULE_ID index,  
    uint8_t epValue,  
    int direction  
);
```

Description

Disables the specified endpoint direction.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
direction	If 1, then TX direction is disabled. If 0 RX direction is disabled.

Returns

None.

Remarks

None.

Example

```
// This function disables the TX direction of endpoint 1  
PLIB_USB_EPnDirectionDisable(MY_USB_INSTANCE, 1, 1);
```

5.6.26.6.3.11 PLIB_USB_EPnHandshakeDisable Function**C**

```
void PLIB_USB_EPnHandshakeDisable(  
    USB_MODULE_ID index,  
    uint8_t epValue  
);
```

Description

This function disables endpoint handshaking. Typically used for Isochronous endpoints.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EPnHandshakeDisable(MY_USB_INSTANCE, someEP);
```

5.6.26.6.3.12 PLIB_USB_EPnHandshakeEnable Function**C**

```
void PLIB_USB_EPnHandshakeEnable(  
    USB_MODULE_ID index,  
    uint8_t epValue  
);
```

```
);
```

Description

This function enables endpoint handshaking.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EPnHandshakeEnable(MY_USB_INSTANCE, someEP);
```

5.6.26.6.3.13 PLIB_USB_EPnIsStalled Function

C

```
bool PLIB_USB_EPnIsStalled(  
    USB_MODULE_ID index,  
    uint8_t epValue  
);
```

Description

This function tests whether the endpoint epValue is stalled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Returns

- true - The endpoint is stalled
- false - The endpoint is not stalled

Remarks

Not valid before an endpoint is enabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
if( PLIB_USB_EPnIsStalled(MY_USB_INSTANCE, someEP) )  
{  
    // Handle the stall
```



```
}
```

5.6.26.6.3.14 PLIB_USB_EPnRxDisable Function

C

```
void PLIB_USB_EPnRxDisable(  
    USB_MODULE_ID index,  
    uint8_t endpoint  
);
```

Description

This function disables an endpoint's ability to process IN tokens.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
//De-provision endpoint 3 to process IN Token  
PLIB_USB_EPnRxDisable(MY_USB_INSTANCE, 3);
```

5.6.26.6.3.15 PLIB_USB_EPnRxEnable Function

C

```
void PLIB_USB_EPnRxEnable(  
    USB_MODULE_ID index,  
    uint8_t endpoint  
);
```

Description

This function enables an endpoint to process IN tokens.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
//Provision endpoint 3 to process IN Token
PLIB_USB_EPnRxEnable(MY_USB_INSTANCE, 3);
```

5.6.26.6.3.16 PLIB_USB_EPnRxSelect Function

C

```
void PLIB_USB_EPnRxSelect(
    USB_MODULE_ID index,
    uint8_t epValue,
    USB_EP_TXRX epTxRx
);
```

Description

This function selects receive capabilities of an endpoint.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
epTxRx	Transmit/Receive setting for endpoint: USB_EP_RX, USB_EP_NOTXRX

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EPnRxSelect(MY_USB_INSTANCE, someEP, USB_EP_RX);
```

5.6.26.6.3.17 PLIB_USB_EPnStallClear Function

C

```
void PLIB_USB_EPnStallClear(
    USB_MODULE_ID index,
    uint8_t epValue
);
```

Description

This function clears an endpoint's stalled flag.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Returns

None.

Remarks

Not valid before an endpoint is enabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
if( PLIB_USB_EPnIsStalled(MY_USB_INSTANCE, someEP) )
{
    // Handle the stall
    PLIB_USB_EPnStallClear(MY_USB_INSTANCE, someEP);
}
```

5.6.26.6.3.18 PLIB_USB_EPnTxDisable Function**C**

```
void PLIB_USB_EPnTxDisable(
    USB_MODULE_ID index,
    uint8_t endpoint
);
```

Description

This function disables an endpoint's ability to process OUT tokens.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
//De-provision endpoint 3 to process OUT Token
PLIB_USB_EPnTxDisable(MY_USB_INSTANCE, 3);
```

5.6.26.6.3.19 PLIB_USB_EPnTxEnable Function**C**

```
void PLIB_USB_EPnTxEnable(
```

```
    USB_MODULE_ID index,  
    uint8_t endpoint  
);
```

Description

This function enables an endpoint to process OUT tokens.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Returns

None.

Remarks

None.

Example

```
//Provision endpoint 3 to process OUT Token  
PLIB_USB_EPnTxEnable(MY_USB_INSTANCE, 3);
```

5.6.26.6.3.20 PLIB_USB_EPnTxRxSelect Function**C**

```
void PLIB_USB_EPnTxRxSelect(  
    USB_MODULE_ID index,  
    uint8_t epValue,  
    USB_EP_TXRX epTxRx  
);
```

Description

This function selects transmit and/or receive capabilities of an endpoint.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
epTxRx	Transmit/Receive setting for endpoint: USB_EP_TX, USB_EP_RX, USB_EP_TX_RX, USB_EP_NOTXRX

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EPnTxRxSelect(MY_USB_INSTANCE, someEP, USB_EP_TXRX);
```

5.6.26.6.3.21 PLIB_USB_EPnTxSelect Function**C**

```
void PLIB_USB_EPnTxSelect(  
    USB_MODULE_ID index,  
    uint8_t epValue,  
    USB_EP_TXRX epTxRx  
);
```

Description

This function selects transmit capabilities of an endpoint.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
epTxRx	Transmit/Receive setting for endpoint: USB_EP_TX, USB_EP_NOTXRX

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEPnRxEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EPnTxSelect(MY_USB_INSTANCE, someEP, USB_EP_TX);
```

5.6.26.6.4 Interrupts Functions**5.6.26.6.4.1 PLIB_USB_InterruptDisable Function****C**

```
void PLIB_USB_InterruptDisable(  
    USB_MODULE_ID index,  
    USB_INTERRUPTS interruptFlag  
);
```

Description

This function disables a general interrupt source for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_Interrupt](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_InterruptDisable( MY_USB_INSTANCE, USB_INT_ERROR );
```

5.6.26.6.4.2 PLIB_USB_InterruptEnable Function**C**

```
void PLIB_USB_InterruptEnable(  
    USB_MODULE_ID index,  
    USB_INTERRUPTS interruptFlag  
);
```

Description

This function enables general interrupt sources of the USB module to trigger a USB interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_Interrupt](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_InterruptEnable( MY_USB_INSTANCE, USB_INT_ERROR );
```

5.6.26.6.4.3 PLIB_USB_InterruptEnableGet Function**C**

```
USB_INTERRUPTS PLIB_USB_InterruptEnableGet(  
    USB_MODULE_ID index  
);
```

Description

Returns the enable/disable status of general USB module interrupts

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

A bit map containing status of enabled interrupts.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_Interrupt](#) in your application to determine whether this feature is available.

Example

```
USB_INTERRUPTS enabledInterrupts;
enabledInterrupts = PLIB_USB_InterruptEnableGet( MY_USB_INSTANCE);
if(enabledInterrupts|USB_INT_ATTACH)
{
    // This means Attach interrupt is enabled.
}
```

5.6.26.6.4.4 PLIB_USB_InterruptFlagAllGet Function**C**

```
USB_INTERRUPTS PLIB_USB_InterruptFlagAllGet(
    USB_MODULE_ID index
);
```

Description

This function returns a logically ORed bit map of active general USB interrupt flags.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

Returns a logically ORed bit map of active general USB interrupt flags.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
USB_INTERRUPTS interruptEnables;

interruptEnables = PLIB_USB_InterruptFlagAllGet(MY_USB_INSTANCE);
if(interruptEnables | USB_INT_DEVICE_RESET)
{
    // Device received reset signalling
}
```

5.6.26.6.4.5 PLIB_USB_InterruptFlagClear Function**C**

```
void PLIB_USB_InterruptFlagClear(
```

```
    USB_MODULE_ID index,  
    USB_INTERRUPTS interruptFlag  
);
```

Description

This function clears a general interrupt source flag for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_InterruptFlagClear( MY_USB_INSTANCE, USB_INT_ERROR );
```

5.6.26.6.4.6 PLIB_USB_InterruptFlagGet Function**C**

```
bool PLIB_USB_InterruptFlagGet(  
    USB_MODULE_ID index,  
    USB_INTERRUPTS interruptFlag  
);
```

Description

This function tests a general interrupt source flag for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_USB_InterruptFlagGet(MY_USB_INSTANCE, USB_INT_ANY) )  
    if ( PLIB_USB_InterruptFlagGet(MY_USB_INSTANCE, USB_INT_ERROR) )  
    {  
        PLIB_USB_InterruptFlagClear(MY_USB_INSTANCE, USB_INT_ERROR);  
    }
```



```
        // Error clean up
    }
    if ( PLIB_USB_InterruptFlagGet(MY_USB_INSTANCE, USB_INT_HOST_DETACH) )
    {
        PLIB_USB_InterruptFlagClear(MY_USB_INSTANCE, USB_INT_HOST_DETACH);
        // Device detached clean up
    }
    .
    .
    .
}
```

5.6.26.6.4.7 PLIB_USB_InterruptFlagSet Function

C

```
void PLIB_USB_InterruptFlagSet(
    USB_MODULE_ID index,
    USB_INTERRUPTS interruptFlag
);
```

Description

This function sets a general interrupt source flag for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_InterruptFlagSet( MY_USB_INSTANCE, USB_INT_ERROR );
```

5.6.26.6.4.8 PLIB_USB_InterruptIsEnabled Function

C

```
bool PLIB_USB_InterruptIsEnabled(
    USB_MODULE_ID index,
    USB_INTERRUPTS interruptFlag
);
```

Description

This function returns true if interrupts are enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

- true - Interrupts are enabled
- false - Interrupts are not enabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_Interrupt](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_USB_InterruptIsEnabled( MY_USB_INSTANCE, USB_INT_ERROR ) )
{
}
```

5.6.26.6.5 Error Interrupts Functions

5.6.26.6.5.1 PLIB_USB_ErrorInterruptDisable Function

C

```
void PLIB_USB_ErrorInterruptDisable(
    USB_MODULE_ID index,
    USB_ERROR_INTERRUPTS interruptFlag
);
```

Description

This function disables an error interrupt source for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsERR_Interrupt](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_ErrorInterruptDisable( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

5.6.26.6.5.2 PLIB_USB_ErrorInterruptEnable Function

C

```
void PLIB_USB_ErrorInterruptEnable(  
    USB_MODULE_ID index,  
    USB_ERROR_INTERRUPTS interruptFlag  
);
```

Description

This function enables error interrupt sources of the USB module to trigger a USB interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsERR_Interrupt](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_ErrorInterruptEnable( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

5.6.26.6.5.3 PLIB_USB_ErrorInterruptFlagAllGet Function

C

```
USB_ERROR_INTERRUPTS PLIB_USB_ErrorInterruptFlagAllGet(  
    USB_MODULE_ID index  
);
```

Description

This function returns a logically ORed bit map of active error interrupt flags.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

Returns a logically ORed bit map of active error interrupt flags.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsERR_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
USB_ERROR_INTERRUPTS errorInterruptEnables;

errorInterruptEnables = PLIB_USB_ErrorInterruptFlagAllGet(MY_USB_INSTANCE);
if(errorInterruptEnables | USB_ERR_INT_DEVICE_EOF_ERROR)
{
    // End of frame error occurred.
}
```

5.6.26.6.5.4 PLIB_USB_ErrorInterruptFlagClear Function**C**

```
void PLIB_USB_ErrorInterruptFlagClear(
    USB_MODULE_ID index,
    USB_ERROR_INTERRUPTS interruptFlag
);
```

Description

This function clears an error interrupt source flag for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

None.

Example

```
PLIB_USB_ErrorInterruptFlagClear( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

5.6.26.6.5.5 PLIB_USB_ErrorInterruptFlagGet Function**C**

```
bool PLIB_USB_ErrorInterruptFlagGet(
    USB_MODULE_ID index,
    USB_ERROR_INTERRUPTS interruptFlag
);
```

Description

This function tests an error interrupt source flag for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsERR_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_USB_ErrorInterruptFlagGet(MY_USB_INSTANCE,USB_ERR_INT_ANY) )
{
    if ( PLIB_USB_ErrorInterruptFlagGet(MY_USB_INSTANCE,USB_ERR_INT_PID_CHECK_FAILURE) )
    {
        PLIB_USB_ErrorInterruptFlagClear(MY_USB_INSTANCE,USB_ERR_INT_PID_CHECK_FAILURE);
        // PID Error Check failure cleanup
    }
    if ( PLIB_USB_ErrorInterruptFlagGet(MY_USB_INSTANCE,USB_ERR_INT_DEVICE_EOF_ERROR) )
    {
        PLIB_USB_ErrorInterruptFlagClear(MY_USB_INSTANCE,USB_ERR_INT_DEVICE_EOF_ERROR);
        // EOF error cleanup
    }
    .
    .
    .
}
```

5.6.26.6.5.6 PLIB_USB_ErrorInterruptFlagSet Function

C

```
void PLIB_USB_ErrorInterruptFlagSet(
    USB_MODULE_ID index,
    USB_ERROR_INTERRUPTS interruptFlag
);
```

Description

This function sets an error interrupt source flag for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsERR_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_ErrorInterruptFlagSet( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

5.6.26.6.5.7 PLIB_USB_ErrorInterruptIsEnabled Function

C

```
bool PLIB_USB_ErrorInterruptIsEnabled(
```

```
    USB_MODULE_ID index,  
    USB_INTERRUPTS interruptFlag  
);
```

Description

This function determines whether interrupts are enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

- true - Interrupts are enabled
- false - Interrupts are not enabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsERR_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_USB_ErrorInterruptIsEnabled( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 ) )  
{  
  
}
```

5.6.26.6.6 Last Transaction Status Functions

5.6.26.6.6.1 PLIB_USB_LastTransactionDetailsGet Function

C

```
void PLIB_USB_LastTransactionDetailsGet(  
    USB_MODULE_ID index,  
    USB_BUFFER_DIRECTION * direction,  
    USB_PING_PONG_STATE * pingpong,  
    uint8_t * endpoint  
);
```

Description

This function returns the details of the last completed transaction.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
direction	Return value contains direction of the last transfer
pingpong	Return value contains Ping-Pong indication of the the last transfer
endpoint	Return value contains the endpoint which processed the last transfer

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsLastTransactionDetails](#) in your application to determine whether this feature is available.

Example

```
USB_BUFFER_DIRECTION direction;
USB_PING_PONG_STATE pingpong;
uint8_t endpoint;

interruptEnables = PLIB_USB_InterruptFlagAllGet(MY_USB_INSTANCE);
if(interruptEnables | USB_INT_TOKEN_DONE)
{
    // Find out details of the token
    PLIB_USB_LastTransactionDetailsGet(MY_USB_INSTANCE, &direction,
                                      &pingpong, &endpoint);
}
```

5.6.26.6.2 PLIB_USB_LastTransactionDirectionGet Function**C**

```
USB_BUFFER_DIRECTION PLIB_USB_LastTransactionDirectionGet(
    USB_MODULE_ID index
);
```

Description

This function indicates the direction of the last transaction, either transmit or receive.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

USB_LastDirection: USB_RECEIVE_TRANSFER or USB_TRANSMIT_TRANSFER

Remarks

None.

Example

See [PLIB_USB_LastTransactionEndPtGet](#).

5.6.26.6.3 PLIB_USB_LastTransactionEndPtGet Function**C**

```
uint8_t PLIB_USB_LastTransactionEndPtGet(
    USB_MODULE_ID index
);
```

Description

This function returns the endpoint number of the last USB transfer, which is actually the index into the Buffer Descriptor Table of the last USB transfer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

endPoint - Endpoint of last completed USB transfer

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsLastDirection](#) in your application to determine whether this feature is available.

Example

```
while ( !PLIB_USB_INT_FlagGet(MY_USB_INSTANCE, USB_INT_GEN, TOKEN_DONE) )
{
    // Do nothing, wait until completion of next transaction
}

// Retrieve information relating to the last completed transaction
endpoint = PLIB_USB_LastTransactionEndPtGet(MY_USB_INSTANCE);
direction = PLIB_USB_LastTransactionDirectionGet(MY_USB_INSTANCE);
pingPongState = PLIB_USB_LastTransactionPingPongStateGet(MY_USB_INSTANCE);

// Clearing the Token Processing Done flag advances the status FIFO to
// oldest transaction in the FIFO. Wait for completion of next transaction
// before using PLIB_USB_Last*Get functions again to read status.
PLIB_USB_INT_FlagClear(MY_USB_INSTANCE, USB_INT_GEN, TOKEN_DONE);
```

5.6.26.6.6.4 PLIB_USB_LastTransactionPingPongStateGet Function**C**

```
USB_PING_PONG_STATE PLIB_USB_LastTransactionPingPongStateGet(
    USB_MODULE_ID index
);
```

Description

This function indicates whether the last transaction was to an Even buffer or an Odd buffer.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

[USB_PING_PONG_STATE](#).

Remarks

Not valid if PIC18 or PIC24F endpoint does not have ping-pong buffering enabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsLastPingPong](#) in your application to determine whether this feature is available.

Example

See [PLIB_USB_LastTransactionEndPtGet](#).

5.6.26.6.7 Host Functions

5.6.26.6.7.1 PLIB_USB_IsBusyWithToken Function

C

```
bool PLIB_USB_IsBusyWithToken(  
    USB_MODULE_ID index  
);
```

Description

This function indicates whether there is a token being executed by the USB module as Host. Software should check that the previous token is finished before issuing a new token.

Preconditions

USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not applicable for PIC18 devices since PIC18 devices do not support Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsHostBusyWithToken](#) in your application to determine whether this feature is available.

Example

```
while( PLIB_USB_IsBusyWithToken(MY_USB_INSTANCE) )  
{  
    // do nothing  
}  
// Issue new token
```

5.6.26.6.7.2 PLIB_USB_SOFDisable Function

C

```
void PLIB_USB_SOFDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables the automatic generation of the SOF token.

Preconditions

USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not applicable for PIC18 devices since PIC18 devices do not support Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsStartOfFrames](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_SOFDisable(MY_USB_INSTANCE);
```

5.6.26.6.7.3 PLIB_USB_SOFEnable Function**C**

```
void PLIB_USB_SOFEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the automatic generation of the SOF token every 1 ms.

Preconditions

USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not applicable for PIC18 devices since PIC18 devices do not support Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsStartOfFrames](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_SOFEnable(MY_USB_INSTANCE);
```

5.6.26.6.7.4 PLIB_USB_SOFThresholdGet Function**C**

```
uint8_t PLIB_USB_SOFThresholdGet(  
    USB_MODULE_ID index  
);
```

Description

This function returns the Start-of-Frame (SOF) Count bits. (Value represents 10 + (packet size of n bytes);).

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

SOF threshold value.

Remarks

Host mode only.

```
SOF Threshold Value = packet byte count + 10
                    = 0b0100_1010 = 0x4A = 74 for 64-byte packet
                    = 0b0010_1010 = 0x2A = 42 for 32-byte packet
                    = 0b0001_1010 = 0x1A = 26 for 16-byte packet
                    = 0x0001_0010 = 0x12 = 18 for 8-byte packet
```

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsSOFThreshold](#) in your application to determine whether this feature is available.

Example

```
thresholdSOF = PLIB_USB_SOFThresholdGet(MY_USB_INSTANCE);
```

5.6.26.6.7.5 PLIB_USB_SOFThresholdSet Function**C**

```
void PLIB_USB_SOFThresholdSet(
    USB_MODULE_ID index,
    uint8_t threshold
);
```

Description

This function sets the Start-of-Frame (SOF) threshold value.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
threshold	SOF threshold

Returns

None.

Remarks

Host mode only.

```
SOF Threshold Value = packet byte count + 10
                    = 0b0100_1010 = 0x4A = 74 for 64-byte packet
                    = 0b0010_1010 = 0x2A = 42 for 32-byte packet
                    = 0b0001_1010 = 0x1A = 26 for 16-byte packet
                    = 0x0001_0010 = 0x12 = 18 for 8-byte packet
```

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsSOFThreshold](#) in your application to determine whether this feature is available.

Example

```
// Set SOF threshold for 64-byte packets
PLIB_USB_SOFThresholdSet(MY_USB_INSTANCE, 64+10);
```

5.6.26.6.7.6 PLIB_USB-TokenEPGet Function**C**

```
uint8_t PLIB_USB-TokenEPGet(
    USB_MODULE_ID index
);
```

Description

This function returns the address of the specified Endpoint.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

Endpoint value - $0 \leq \text{epValue} \leq \text{Module Maximum Endpoint}$.

Remarks

Host mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsTokenEP](#) in your application to determine whether this feature is available.

Example

```
someEP = PLIB_USB-TokenEPGet(MY_USB_INSTANCE);
```

5.6.26.6.7.7 PLIB_USB-TokenEPSet Function**C**

```
void PLIB_USB-TokenEPSet(
    USB_MODULE_ID index,
    uint8_t epValue
);
```

Description

This function sets the Endpoint address for a host transaction.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, $0 \leq \text{epValue} \leq \text{Module Maximum Endpoint}$

Returns

None.

Remarks

Host mode only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsTokenEP](#) in your application to determine whether this feature is available.

Example

```
uint8_t someEP = 0x03;
PLIB_USB-TokenEPSet(MY_USB_INSTANCE, someEP);
```

5.6.26.6.7.8 PLIB_USB-TokenPIDGet Function

C

```
USB_PID PLIB_USB-TokenPIDGet(
    USB_MODULE_ID index
);
```

Description

This function returns the token transaction type.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

Packet ID of token, USB_PID_SETUP, USB_PID_IN, or USB_PID_OUT

Remarks

Host mode only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsTokenPID](#) in your application to determine whether this feature is available.

Example

```
somePID = PLIB_USB-TokenPIDGet(MY_USB_INSTANCE);
```

5.6.26.6.7.9 PLIB_USB-TokenPIDSet Function

C

```
void PLIB_USB-TokenPIDSet(
    USB_MODULE_ID index,
    USB_PID pidValue
);
```

Description

This function sets the token transaction type to pidValue.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

pidValue	USB_PID_SETUP, USB_PID_IN, or USB_PID_OUT

Returns

None.

Remarks

Host mode only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsTokenPID](#) in your application to determine whether this feature is available.

Example

```
somePID = USB_PID_SETUP;
PLIB_USB-TokenPIDSet (MY_USB_INSTANCE, somePID );
```

5.6.26.6.7.10 PLIB_USB-TokenSpeedSelect Function

C

```
void PLIB_USB-TokenSpeedSelect(
    USB_MODULE_ID index,
    USB_TOKEN_SPEED tokenSpeed
);
```

Description

This function selects low speed or full speed for subsequent token executions.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
tokenSpeed	Speed for next token execution: USB_LOWSPEED_TOKENS or USB_FULLSPEED_TOKENS

Returns

None.

Remarks

Not applicable for PIC18 devices since PIC18 devices do not support Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsNextTokenSpeed](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB-TokenSpeedSet (MY_USB_INSTANCE, USB_LOWSPEED_TOKENS);
```

5.6.26.6.8 USB Bus Signaling Functions

5.6.26.6.8.1 PLIB_USB_ResetSignalDisable Function

C

```
void PLIB_USB_ResetSignalDisable(
    USB_MODULE_ID index
);
```

Description

This function disables reset signalling on the USB bus.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not applicable for PIC18, since USB Host functionality is Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsHostGeneratesReset](#) in your application to determine whether this feature is available.

Example

See [PLIB_USB_ResetSignalEnable](#).

5.6.26.6.8.2 PLIB_USB_ResetSignalEnable Function**C**

```
void PLIB_USB_ResetSignalEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables reset signalling on the USB bus.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not applicable for PIC18, since USB Host functionality is Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsHostGeneratesReset](#) in your application to determine whether this feature is available.

Example

```
// Snippet to perform a software reset:  
PLIB_USB_ResetSignalEnable(MY_USB_INSTANCE);  
// ... delay 50ms  
PLIB_USB_ResetSignalDisable(MY_USB_INSTANCE);
```

5.6.26.6.8.3 PLIB_USB_ResumeSignalingDisable Function

C

```
void PLIB_USB_ResumeSignalingDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables resume signaling.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsResumeSignaling](#) in your application to determine whether this feature is available.

Example

See [PLIB_USB_ResumeSignalingEnable](#).

5.6.26.6.8.4 PLIB_USB_ResumeSignalingEnable Function

C

```
void PLIB_USB_ResumeSignalingEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables resume signalling. Resume allows the peripheral to perform a remote wake-up by executing resume signaling.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Software must enable resume signaling for 10 ms if the device is in Device mode, or for 25 ms if the device is in Host mode, and then disable resume signaling to enable remote wake-up. In Host mode, the USB module will append a low-speed EOP to the end resume signaling when it is disabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsResumeSignaling](#) in your application to determine whether this

feature is available.

Example

```
// Perform resume signaling:
PLIB_USB_ResumeSignalingEnable(MY_USB_INSTANCE);
// Delay 10ms
PLIB_USB_ResumeSignalingDisable(MY_USB_INSTANCE);
```

5.6.26.6.9 On-The-Go (OTG) Functions

5.6.26.6.9.1 PLIB_USB_OTG_BSessionHasEnded Function

C

```
bool PLIB_USB_OTG_BSessionHasEnded(
    USB_MODULE_ID index
);
```

Description

This function returns the status of the B-Session End Indicator bit.

Preconditions

The USB module must be in OTG mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

- true - The VBUS Voltage is below VB_SESS_END on the B-device
- false - The VBUS voltage is above VB_SESS_END on the B-device

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_BSessionEnd](#) in your application to determine whether this feature is available.

Example

```
if ( !PLIB_USB_OTG_BSessionHasEnded(MY_USB_INSTANCE) )
{
    // B session valid
}
else
{
    // B session not valid
}
```

5.6.26.6.9.2 PLIB_USB_OTG_IDPinStateIsTypeA Function

C

```
bool PLIB_USB_OTG_IDPinStateIsTypeA(
    USB_MODULE_ID index
);
```

Description

This function returns ID Pin state.

Preconditions

The USB module must be in OTG mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

- true - Type A Cable attached,
- false - No cable is attached or a Type B cable is attached

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_IDPinState](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_USB_OTG_IDPinStateIsTypeA(MY_USB_INSTANCE) )
{
    // Type A cable attached
}
else
{
    // No cable or Type B cable attached
};
```

5.6.26.6.9.3 PLIB_USB_OTG_LineStateIsStable Function**C**

```
bool PLIB_USB_OTG_LineStateIsStable(
    USB_MODULE_ID index
);
```

Description

This function returns the status of the Line Stable Indicator bit.

Preconditions

The USB module must be in OTG mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

- true - The USB line state has been stable for the previous 1 ms
- false - The USB line state has not been stable for the previous 1 ms

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_LineState](#) in your application to determine whether this feature is available.

Example

```
if( PLIB_USB_OTG_LineStateIsStable(MY_USB_INSTANCE) ) {
    // Line has been stable
}
```

```
    // ... rest of code ...  
}
```

5.6.26.6.9.4 PLIB_USB_OTG_PullUpPullDownSetup Function

C

```
void PLIB_USB_OTG_PullUpPullDownSetup(  
    USB_MODULE_ID index,  
    USB_OTG_PULL_UP_PULL_DOWN resistor,  
    bool enableResistor  
);
```

Description

This function enables or disables pull-up and pull-down resistors.

Preconditions

USB On-The-Go (OTG) must be enabled.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
resistor	USB_OTG_DPLUS_PULLUP, USB_OTG_DMINUS_PULLUP, USB_OTG_DPLUS_PULLDN, or USB_OTG_DMINUS_PULLDN
enableResistor	true to enable resistor, false to disable it

Returns

None.

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_PullUpPullDown](#) in your application to determine whether this feature is available.

Example

```
// Enable pull-up resistor for D+  
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE, USB_OTG_DPLUS_PULLUP, true);
```

5.6.26.6.9.5 PLIB_USB_OTG_SessionValid Function

C

```
bool PLIB_USB_OTG_SessionValid(  
    USB_MODULE_ID index  
);
```

Description

This function returns the status of the Session Valid Indicator bit.

Preconditions

The USB module must be in OTG mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

- true - The VBUS voltage is above Session Valid on the A or B device

- false - The VBUS voltage is below Session Valid on the A or B device

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_SessionValid](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_USB_OTG_SessionValid(MY_USB_INSTANCE) )
{
    // Session valid
}
else
{
    // Session not valid
}
```

5.6.26.6.9.6 PLIB_USB_OTG_VBusChargeDisable Function**C**

```
void PLIB_USB_OTG_VBusChargeDisable(
    USB_MODULE_ID index
);
```

Description

This function disables VBUS line charge.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 or PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_VbusCharge](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_VBusChargeDisable(MY_USB_INSTANCE);
```

5.6.26.6.9.7 PLIB_USB_OTG_VBusChargeEnable Function**C**

```
void PLIB_USB_OTG_VBusChargeEnable(
    USB_MODULE_ID index
);
```

Description

This function enables the VBUS line to be charged through a pull-up resistor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 or PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_VbusDischarge](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_VBusChargeEnable(MY_USB_INSTANCE);
```

5.6.26.6.9.8 PLIB_USB_OTG_VBusChargeTo3V Function**C**

```
void PLIB_USB_OTG_VBusChargeTo3V(  
    USB_MODULE_ID index  
);
```

Description

This function sets the VBUS line to charge to 3.3V.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 or PIC32 devices.

Example

```
PLIB_USB_OTG_VBusChargeTo3V(MY_USB_INSTANCE);
```

5.6.26.6.9.9 PLIB_USB_OTG_VBusChargeTo5V Function**C**

```
void PLIB_USB_OTG_VBusChargeTo5V(  
    USB_MODULE_ID index  
);
```

Description

This function sets the VBUS line to charge to 5V.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 or PIC32 devices.

Example

```
PLIB_USB_OTG_VBusChargeTo5V (MY_USB_INSTANCE);
```

5.6.26.6.9.10 PLIB_USB_OTG_VBusDischargeDisable Function**C**

```
void PLIB_USB_OTG_VBusDischargeDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables VBUS line discharge.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 or PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_VbusDischarge](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_VBusDischargeDisable(MY_USB_INSTANCE);
```

5.6.26.6.9.11 PLIB_USB_OTG_VBusDischargeEnable Function**C**

```
void PLIB_USB_OTG_VBusDischargeEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the VBUS line to be discharged through a resistor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 or PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_VbusCharge](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_VBusDischargeEnable(MY_USB_INSTANCE);
```

5.6.26.6.9.12 PLIB_USB_OTG_VBusPowerOff Function**C**

```
void PLIB_USB_OTG_VBusPowerOff(  
    USB_MODULE_ID index  
);
```

Description

This function turns off power on the VBUS Line.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_VbusPowerOnOff](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_VBusPowerOff(MY_USB_INSTANCE);
```

5.6.26.6.9.13 PLIB_USB_OTG_VBusPowerOn Function**C**

```
void PLIB_USB_OTG_VBusPowerOn(  
    USB_MODULE_ID index  
);
```

Description

This function turns on power for the VBUS line.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_VbusPowerOnOff](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_VBusPowerOn(MY_USB_INSTANCE);
```

5.6.26.6.9.14 PLIB_USB_OTG_VBusValid Function**C**

```
bool PLIB_USB_OTG_VBusValid(  
    USB_MODULE_ID index  
);
```

Description

This function returns the status of the A-VBUS valid indicator.

Preconditions

The USB module must be in OTG mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

- true - The VBUS voltage is above VA_VBUS_VLD on the A-device,
- false - The VBUS voltage is below VA_VBUS_VLD on the A-device

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_ASessionValid](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_USB_OTG_VBusValid(MY_USB_INSTANCE) )  
{  
    // Vbus voltage above session valid for A device  
}  
else  
{  
    // Vbus voltage below session valid for A device  
}
```


5.6.26.6.10 OTG Interrupts Functions

5.6.26.6.10.1 PLIB_USB_OTG_InterruptDisable Function

C

```
void PLIB_USB_OTG_InterruptDisable(  
    USB_MODULE_ID index,  
    USB_OTG_INTERRUPTS interruptFlag  
);
```

Description

This function disables a USB On-The-Go (OTG) interrupt source for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_Interrupt](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_InterruptDisable( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

5.6.26.6.10.2 PLIB_USB_OTG_InterruptEnable Function

C

```
void PLIB_USB_OTG_InterruptEnable(  
    USB_MODULE_ID index,  
    USB_OTG_INTERRUPTS interruptFlag  
);
```

Description

This function enables USB On-The-Go (OTG) interrupt sources of the USB module to trigger a USB interrupt.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_Interrupt](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_InterruptEnable( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

5.6.26.6.10.3 PLIB_USB_OTG_InterruptFlagClear Function**C**

```
void PLIB_USB_OTG_InterruptFlagClear(  
    USB_MODULE_ID index,  
    USB_OTG_INTERRUPTS interruptFlag  
);
```

Description

This function clears a USB On-The-Go (OTG) interrupt source flag for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_InterruptFlagClear( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

5.6.26.6.10.4 PLIB_USB_OTG_InterruptFlagGet Function**C**

```
bool PLIB_USB_OTG_InterruptFlagGet(  
    USB_MODULE_ID index,  
    USB_OTG_INTERRUPTS interruptFlag  
);
```

Description

This function tests a USB On-The-Go (OTG) interrupt source flag for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_USB_OTG_InterruptFlagGet(MY_USB_INSTANCE,USB_OTG_INT_ANY) )
{
    if ( PLIB_USB_OTG_InterruptFlagGet(MY_USB_INSTANCE,USB_OTG_INT_BDEVICE_SESSION_END) )
    {
        PLIB_USB_OTG_InterruptFlagClear(MY_USB_INSTANCE,USB_OTG_INT_ADEVICE_VBUS_VALID );
        // Device A Vbus Valid Change
    }
    if ( PLIB_USB_OTG_InterruptFlagGet(MY_USB_INSTANCE,USB_OTG_INT_BDEVICE_SESSION_END) )
    {
        PLIB_USB_OTG_InterruptFlagClear(MY_USB_INSTANCE,USB_OTG_INT_BDEVICE_SESSION_END);
        // Device B Vbus Valid Change
    }
    .
    .
    .
}
```

5.6.26.6.10.5 PLIB_USB_OTG_InterruptFlagSet Function

C

```
void PLIB_USB_OTG_InterruptFlagSet(
    USB_MODULE_ID index,
    USB_OTG_INTERRUPTS interruptFlag
);
```

Description

This function sets a USB On-The-Go (OTG) interrupt source flag for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_InterruptStatus](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_OTG_InterruptFlagSet( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

5.6.26.6.10.6 PLIB_USB_OTG_InterruptIsEnabled Function

C

```
bool PLIB_USB_OTG_InterruptIsEnabled(
```

```
    USB_MODULE_ID index,  
    USB_INTERRUPTS interruptFlag  
);
```

Description

This function returns whether or not interrupts are enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Returns

- true - Interrupts are enabled
- false - Interrupts are not enabled

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOTG_Interrupt](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_USB_OTG_InterruptIsEnabled( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE ) )  
{  
  
}
```

5.6.26.6.11 USB Activity Functions

5.6.26.6.11.1 PLIB_USB_ActivityPending Function

C

```
bool PLIB_USB_ActivityPending(  
    USB_MODULE_ID index  
);
```

Description

This function returns whether or not USB bus activity has been detected, an interrupt is pending, or an interrupt is yet to be generated.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

- true - The USB module should not be suspended
- false - No interrupts are pending or module may be suspended or powered down

Remarks

Not available on PIC18 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsActivityPending](#) in your application to determine whether this feature is available.

Example

```
while ( PLIB_USB_ActivityPending(MY_USB_INSTANCE) )
{
    // Wait
}
// Suspend USB module.
```

5.6.26.6.11.2 PLIB_USB_FrameNumberGet Function**C**

```
uint16_t PLIB_USB_FrameNumberGet(
    USB_MODULE_ID index
);
```

Description

This function returns the USB frame number in the lower 11 bits.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

Current frame number in lower 11 bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsFrameNumber](#) in your application to determine whether this feature is available.

Example

```
frameNumber = PLIB_USB_FrameNumberGet(MY_USB_INSTANCE);
```

5.6.26.6.11.3 PLIB_USB_JStateIsActive Function**C**

```
bool PLIB_USB_JStateIsActive(
    USB_MODULE_ID index
);
```

Description

This function indicates the live JState (differential '0' in low speed, differential '1' in full speed) on the bus.

Preconditions

The USB module must be in Host mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not applicable for PIC18 devices since PIC18 devices do not support Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsLiveJState](#) in your application to determine whether this feature is available.

Example

```
// Enable Host Mode
PLIB_USB_OperatingModeSelect(MY_USB_INSTANCE, USB_OPMODE_HOST);

// Enable D+ and D- pull-down resistors
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE, USB_OTG_DPLUS_PULLDN, true);
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE, USB_OTG_DMINUS_PULLDN, true);

// Disable D+ and D- pull-up resistors
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE, USB_OTG_DPLUS_PULLUP, false);
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE, USB_OTG_DMINUS_PULLUP, false);

// Enable SOF Packet generation
PLIB_USB_SOFEnable(MY_USB_INSTANCE);

// Enable the device attach interrupt
PLIB_USB_INT_Enable(MY_USB_INSTANCE, USB_INT_OTG, ACTIVITY_DETECT);

// Wait for the Attach interrupt.
while( !PLIB_USB_INT_FlagGet(MY_USB_INSTANCE, USB_INT_OTG, ACTIVITY_DETECT) )
{
    //Do nothing
}

// Check JState
if( PLIB_USB_JStateIsActive(MY_USB_INSTANCE) )
{
    // Full Speed
    PLIB_USB_FullSpeedEnable(MY_USB_INSTANCE);
}
else
{
    // Low Speed
    PLIB_USB_FullSpeedDisable(MY_USB_INSTANCE);
}
```

5.6.26.6.11.4 PLIB_USB_PacketTransferDisable Function

C

```
void PLIB_USB_PacketTransferDisable(
    USB_MODULE_ID index
);
```

Description

This function disables the Serial Interface Engine (SIE).

Preconditions

USB module must be in device mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

```
if( PLIB_USB_PacketTransferIsDisabled(MY_USB_INSTANCE) )
{
    // SETUP token received, do the needful operations
    .
    .
    .
    // SETUP handling completed, enable Setup token and packet processing:
    PLIB_USB_PacketTransferDisable(MY_USB_INSTANCE);
}
```

Remarks

Not valid when the USB module is in Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsPacketTransfer](#) in your application to determine whether this feature is available.

5.6.26.6.11.5 PLIB_USB_PacketTransferEnable Function**C**

```
void PLIB_USB_PacketTransferEnable(
    USB_MODULE_ID index
);
```

Description

This function re-enables the Serial Interface Engine (SIE), allowing token and packet processing.

Preconditions

USB module must be in device mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

```
if( PLIB_USB_PacketTransferIsDisabled(MY_USB_INSTANCE) )
{
    // SETUP token received, do the needful operations
    .
    .
    .
    // SETUP handling completed, enable Setup token and packet processing:
    PLIB_USB_PacketTransferEnable(MY_USB_INSTANCE);
}
```

Remarks

Not valid when the USB module is in Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsPacketTransfer](#) in your application to determine whether this feature is available.

5.6.26.6.11.6 PLIB_USB_PacketTransferIsDisabled Function**C**

```
bool PLIB_USB_PacketTransferIsDisabled(
```

```
    USB_MODULE_ID index
);
```

Description

This function indicates that a setup token has been received from the Host and that the Serial Interface Engine (SIE) has been turned off, disabling token and packet processing.

Preconditions

USB module must be in device mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Not valid when USB is Host. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsPacketTransfer](#) in your application to determine whether this feature is available.

Example

```
if( PLIB_USB_PacketTransferIsDisabled(MY_USB_INSTANCE) )
{
    // SETUP token received, do the needful operations
    .
    .
    .
    // SETUP handling completed, enable Setup token and packet processing:
    PLIB_USB_PacketTransferEnable(MY_USB_INSTANCE);
}
```

5.6.26.6.11.7 PLIB_USB_SE0InProgress Function**C**

```
bool PLIB_USB_SE0InProgress(
    USB_MODULE_ID index
);
```

Description

This function returns whether a single-ended zero event is in progress.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

- true - A single ended zero event (SE0) is occurring
- false - A single-ended zero event (SE0) is not occurring

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsLiveSingleEndedZero](#) in your application to determine whether this feature is available.

Example

```
if( PLIB_USB_SE0InProgress(MY_USB_INSTANCE) )
{
    // handle the SE0 event
}
```

5.6.26.6.12 External Transceiver Support Functions

5.6.26.6.12.1 PLIB_USB_I2CInterfaceForExtModuleDisable Function

C

```
void PLIB_USB_I2CInterfaceForExtModuleDisable(
    USB_MODULE_ID index
);
```

Description

Specifies that external module(s) are controlled via dedicated pins.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_I2CInterfaceForExtModuleDisable(MY_USB_INSTANCE);
```

5.6.26.6.12.2 PLIB_USB_I2CInterfaceForExtModuleEnable Function

C

```
void PLIB_USB_I2CInterfaceForExtModuleEnable(
    USB_MODULE_ID index
);
```

Description

This function specifies that external module(s) are controlled via the I2C interface.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_I2CInterfaceForExtModuleEnable(MY_USB_INSTANCE);
```

5.6.26.6.12.3 PLIB_USB_TransceiverDisable Function**C**

```
void PLIB_USB_TransceiverDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables the on-chip transceiver and enables the interface to the off-chip transceiver.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

PIC18 devices only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOnChipTransceiver](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_TransceiverDisable(MY_USB_INSTANCE);
```

5.6.26.6.12.4 PLIB_USB_TransceiverEnable Function**C**

```
void PLIB_USB_TransceiverEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the on-chip transceiver. The interface to the off-chip transceiver is disabled.

Preconditions

Use only before the USB module is enabled by calling [PLIB_USB_Enable](#).

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

PIC18 devices only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsOnChipTransceiver](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_TransceiverEnable(MY_USB_INSTANCE);
```

5.6.26.6.13 VBus Support Functions

5.6.26.6.13.1 PLIB_USB_ExternalComparatorMode2Pin Function

C

```
void PLIB_USB_ExternalComparatorMode2Pin(  
    USB_MODULE_ID index  
);
```

Description

This function sets the 2-pin input configuration for VBUS Comparators.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_ExternalComparatorMode2Pin(MY_USB_INSTANCE);
```

5.6.26.6.13.2 PLIB_USB_ExternalComparatorMode3Pin Function

C

```
void PLIB_USB_ExternalComparatorMode3Pin(  
    USB_MODULE_ID index  
);
```

Description

This function sets the 3-pin input configuration for VBUS comparators.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_ExternalComparatorMode3Pin(MY_USB_INSTANCE);
```

5.6.26.6.13.3 PLIB_USB_PWMCounterDisable Function**C**

```
void PLIB_USB_PWMCounterDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables the PWM counter used to generate the VBUS for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_PWMDisable(MY_USB_INSTANCE);  
PLIB_USB_PWMCounterDisable(MY_USB_INSTANCE);
```

5.6.26.6.13.4 PLIB_USB_PWMCounterEnable Function**C**

```
void PLIB_USB_PWMCounterEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the PWM counter used to generate the VBUS for the USB module.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_PWMEnable(MY_USB_INSTANCE);  
PLIB_USB_PWMCounterEnable(MY_USB_INSTANCE);
```

5.6.26.6.13.5 PLIB_USB_PWMDisable Function**C**

```
void PLIB_USB_PWMDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables the PWM Generator. PWM output held in a reset state defined by [PLIB_USB_PWMPolarityActiveHigh](#) or [PLIB_USB_PWMPolarityActiveLow](#).

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_PWMDisable (MY_USB_INSTANCE);
```

5.6.26.6.13.6 PLIB_USB_PWMEnable Function**C**

```
void PLIB_USB_PWMEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the PWM Generator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_PWMEnable(MY_USB_INSTANCE);
```

5.6.26.6.13.7 PLIB_USB_PWMPolarityActiveLow Function**C**

```
void PLIB_USB_PWMPolarityActiveLow(  
    USB_MODULE_ID index  
);
```

Description

This function sets the PWM output to active-high and resets low.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

5.6.26.6.13.8 PLIB_USB_PWMPolarityActiveHigh Function**C**

```
void PLIB_USB_PWMPolarityActiveHigh(  
    USB_MODULE_ID index  
);
```

Description

This function sets the PWM output to active-low and resets high.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_PWMPolarity (MY_USB_INSTANCE);
```

5.6.26.6.13.9 PLIB_USB_VBoostDisable Function**C**

```
void PLIB_USB_VBoostDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables the On-Chip 5V Boost Regulator Circuit Disabled bit.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_VBoostDisable(MY_USB_INSTANCE);
```

5.6.26.6.13.10 PLIB_USB_VBoostEnable Function**C**

```
void PLIB_USB_VBoostEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the On-Chip 5V Boost Regulator Circuit Enabled bit.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_VBoostEnable(MY_USB_INSTANCE);
```

5.6.26.6.13.11 PLIB_USB_VBUSComparatorDisable Function**C**

```
void PLIB_USB_VBUSComparatorDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables the on-chip VBUS Comparator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_VBUSComparatorDisable(MY_USB_INSTANCE);
```

5.6.26.6.13.12 PLIB_USB_VBUSComparatorEnable Function**C**

```
void PLIB_USB_VBUSComparatorEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the on-chip VBUS Comparator.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_VBUSComparatorEnable(MY_USB_INSTANCE);
```

5.6.26.6.13.13 PLIB_USB_VBUSPullUpDisable Function**C**

```
void PLIB_USB_VBUSPullUpDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables the pull-up on the VBUS pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_VBUSPullUpDisable(MY_USB_INSTANCE);
```

5.6.26.6.13.14 PLIB_USB_VBUSPullUpEnable Function**C**

```
void PLIB_USB_VBUSPullUpEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the pull-up on the VBUS pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Example

```
PLIB_USB_VBUSPullUpEnable(MY_USB_INSTANCE);
```

5.6.26.6.14 Test Support Functions

5.6.26.6.14.1 PLIB_USB_EyePatternDisable Function

C

```
void PLIB_USB_EyePatternDisable(  
    USB_MODULE_ID index  
);
```

Description

This function disables the USB eye pattern test.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEyePattern](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EyePatternDisable(MY_USB_INSTANCE);
```

5.6.26.6.14.2 PLIB_USB_EyePatternEnable Function

C

```
void PLIB_USB_EyePatternEnable(  
    USB_MODULE_ID index  
);
```

Description

This function enables the USB eye pattern test.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsEyePattern](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_EyePatternEnable(MY_USB_INSTANCE);
```

5.6.26.6.15 Other Functions

5.6.26.6.15.1 PLIB_USB_ModuleIsBusy Function

C

```
bool PLIB_USB_ModuleIsBusy(  
    USB_MODULE_ID index  
);
```

Description

This function indicates if the USB module is not ready to be enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

- true - USB module is active or disabled, but not ready to be enabled
- false - USB module is not active and is ready to be enabled

Remarks

This feature is not available on PIC18, PIC24, and dsPIC33 devices.

If `PLIB_USB_ModuleIsBusy(MY_USB_INSTANCE) == true` and the USB module is disabled, and all status returned for the module, all enables/disables for the module will produce undefined results.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use `PLIB_USB_ExistsModuleBusy` in your application to determine whether this feature is available.

Example

```
#ifndef (__PIC32MX__)  
while ( PLIB_USB_ModuleIsBusy(MY_USB_INSTANCE) )  
{  
    // wait
```

```
}  
#endif  
    PLIB_USB_Disable(MY_USB_INSTANCE);
```

5.6.26.6.15.2 PLIB_USB_PingPongFreeze Function

C

```
void PLIB_USB_PingPongFreeze(  
    USB_MODULE_ID index  
);
```

Description

This function resets all Ping-Pong buffer pointers to even buffers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

Buffers remain "frozen" at "Even" until they are unfrozen using [PLIB_USB_PingPongUnfreeze](#).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBufferFreeze](#) in your application to determine whether this feature is available.

Example

```
// Reset all ping-pong buffers to "Even"  
PLIB_USB_PingPongFreeze(MY_USB_INSTANCE);  
PLIB_USB_PingPongUnfreeze(MY_USB_INSTANCE);
```

5.6.26.6.15.3 PLIB_USB_PingPongReset Function

C

```
void PLIB_USB_PingPongReset(  
    USB_MODULE_ID index  
);
```

Description

This function resets the USB peripheral internal Ping-Pong indicator to point to Even buffers.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_USB_ExistsBufferFreeze](#) in your application to determine whether this feature is available.

Example

```
PLIB_USB_PingPongReset(MY_USB_INSTANCE);
```

5.6.26.6.15.4 PLIB_USB_PingPongUnfreeze Function

C

```
void PLIB_USB_PingPongUnfreeze(  
    USB_MODULE_ID index  
);
```

Description

This function enables Ping-Pong buffering.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Returns

None.

Remarks

See [PLIB_USB_PingPongFreeze](#). This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsBufferFreeze](#) in your application to determine whether this feature is available.

Example

```
// Reset all Ping-Pong buffers to "Even"  
PLIB_USB_PingPongFreeze(MY_USB_INSTANCE);  
PLIB_USB_PingPongUnfreeze(MY_USB_INSTANCE);
```

5.6.26.6.15.5 PLIB_USB_TokenSend Function

C

```
void PLIB_USB_TokenSend(  
    USB_MODULE_ID index,  
    USB_PID pidValue,  
    uint8_t endpoint,  
    uint8_t deviceAddress,  
    bool isLowSpeed  
);
```

Description

This function sends the specified token to the specified endpoint and address. The token is placed on the bus at the next available time. The token can be executed at low speed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pidValue	PID of the token to be placed on the bus.

endpoint	Device endpoint to which the token should be sent.
isLowSpeed	Is true if the token should be executed at low speed.

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_USB_ExistsGEN_Interrupt](#) in your application to determine whether this feature is available.

Example

```
// Send an OUT token to endpoint 1 device address 2 at full speed  
PLIB_USB_SendToken(MY_USB_INSTANCE, USB_PID_OUT, 1, 2, false);
```

5.6.26.6.16 Data Types and Constants

5.6.26.6.16.1 USB_BUFFER_DATA01 Enumeration

C

```
typedef enum {  
    USB_BUFFER_DATA0,  
    USB_BUFFER_DATA1  
} USB_BUFFER_DATA01;
```

Description

USB Endpoint Buffer Data Toggle Enumeration

This data type provides enumeration data toggle for a buffer.

Members

Members	Description
USB_BUFFER_DATA0	DATA0/1 = 0
USB_BUFFER_DATA1	DATA0/1 = 1

Remarks

None.

5.6.26.6.16.2 USB_BUFFER_DIRECTION Enumeration

C

```
typedef enum {  
    USB_BUFFER_RX,  
    USB_BUFFER_TX  
} USB_BUFFER_DIRECTION;
```

Description

USB Endpoint Buffer Direction Enumeration

This data type provides enumeration transmit/receive direction for a buffer.

Members

Members	Description
USB_BUFFER_RX	Receive
USB_BUFFER_TX	Transmit

Remarks

None.

5.6.26.6.16.3 USB_BUFFER_PING_PONG Enumeration

C

```
typedef enum {  
    USB_BUFFER_EVEN,  
    USB_BUFFER_ODD  
} USB_BUFFER_PING_PONG;
```

Description

Enumeration of USB Buffer Ping-Pong

This data type enumerates the ping-pong buffer (Even vs. Odd).

Members

Members	Description
USB_BUFFER_EVEN	Even Buffer
USB_BUFFER_ODD	Odd Buffer

Remarks

None.

5.6.26.6.16.4 USB_BUFFER_SCHEDULE_DATA01 Enumeration

C

```
typedef enum {  
    USB_BUFFER_DONTCHANGE,  
    USB_BUFFER_SET_DATA0,  
    USB_BUFFER_SET_DATA1  
} USB_BUFFER_SCHEDULE_DATA01;
```

Description

USB Endpoint Buffer Data Toggle Enumeration for Buffer Scheduling

This data type provides enumeration data toggle for a buffer.

Members

Members	Description
USB_BUFFER_DONTCHANGE	Don't Change DATA0/1
USB_BUFFER_SET_DATA0	DATA0/1 = 0
USB_BUFFER_SET_DATA1	DATA0/1 = 1

Remarks

None.

5.6.26.6.16.5 USB_EP_TXRX Enumeration

C

```
typedef enum {  
    USB_EP_NOTXRX,  
    USB_EP_RX,  
    USB_EP_TX,  
}
```

```

    USB_EP_TX_RX
} USB_EP_TXRX;

```

Description

Enumeration of USB Endpoint Transmit/Receive Setup

This data type provides enumeration transmit/receive setup for an endpoint.

Members

Members	Description
USB_EP_NOTXRX	Nothing enabled for endpoint
USB_EP_RX	Receive enabled for endpoint
USB_EP_TX	Transmit enabled for endpoint
USB_EP_TX_RX	Transmit and Receive enabled for endpoint

Remarks

None.

5.6.26.6.16.6 USB_OPMODES Enumeration**C**

```

typedef enum {
    USB_OPMODE_NONE,
    USB_OPMODE_DEVICE,
    USB_OPMODE_HOST,
    USB_OPMODE_OTG
} USB_OPMODES;

```

Description

USB Operating Modes Enumeration

This data type provides enumeration of the operating modes supported by the USB module.

Members

Members	Description
USB_OPMODE_NONE	None
USB_OPMODE_DEVICE	Device
USB_OPMODE_HOST	Host
USB_OPMODE_OTG	OTG

Remarks

PIC18 devices only support USB_OPMODE_DEVICE.

5.6.26.6.16.7 USB_OTG_INTERRUPTS Enumeration**C**

```

typedef enum {
    USB_OTG_INT_ADEVICE_VBUS_VALID,
    USB_OTG_INT_OTG_RESERVED,
    USB_OTG_INT_BDEVICE_SESSION_END,
    USB_OTG_INT_SESSION_VALID,
    USB_OTG_INT_ACTIVITY_DETECT,
    USB_OTG_INT_STABLE_LINE_STATE,
    USB_OTG_INT_ONE_MS_TIMEOUT,
    USB_OTG_INT_ID_STATE_CHANGE,
    USB_OTG_INT_ANY,
}

```



```

    USB_OTG_INT_ALL
} USB_OTG_INTERRUPTS;

```

Description

USB OTG Interrupts Enumeration

This data type provides enumeration of interrupts related to the USB OTG module.

Members

Members	Description
USB_OTG_INT_ADEVICE_VBUS_VALID	State of (Vbus > Va_vbus_vld) on the A device has changed
USB_OTG_INT_OTG_RESERVED	Reserved. Don't use.
USB_OTG_INT_BDEVICE_SESSION_END	State of (Vbus < Vb_sess_end) on the B device has changed
USB_OTG_INT_SESSION_VALID	State of (Vbus > Va_sess_vld) on the A or B devices has changed
USB_OTG_INT_ACTIVITY_DETECT	Activity detected on the D+, D-, ID, or Vbus lines
USB_OTG_INT_STABLE_LINE_STATE	USB line state has been stable for 1 ms, but different from last time
USB_OTG_INT_ONE_MS_TIMEOUT	One millescond timer has expired
USB_OTG_INT_ID_STATE_CHANGE	Change in state of ID pin detected.
USB_OTG_INT_ANY	All or Any of the above
USB_OTG_INT_ALL	All or Any of the above

Remarks

Not applicable on PIC18 devices or if the USB OTG module is not enabled.

5.6.26.6.16.8 USB_OTG_PULL_UP_PULL_DOWN Enumeration**C**

```

typedef enum {
    USB_OTG_DPLUS_PULLUP,
    USB_OTG_DMINUS_PULLUP,
    USB_OTG_DPLUS_PULLDN,
    USB_OTG_DMINUS_PULLDN
} USB_OTG_PULL_UP_PULL_DOWN;

```

Description

Enumeration of Pull-Up and Pull-Down Resistors for OTG

This data type enumerates the OTG Pull-Up and Pull-Down resistors for D+ and D- .

Members

Members	Description
USB_OTG_DPLUS_PULLUP	D+ Pull-Up
USB_OTG_DMINUS_PULLUP	D- Pull-Up
USB_OTG_DPLUS_PULLDN	D+ Pull-Down
USB_OTG_DMINUS_PULLDN	D- Pull-Down

Remarks

Not applicable for PIC18 devices.

5.6.26.6.16.9 USB_PID Enumeration**C**

```

typedef enum {

```

```

    USB_PID_SETUP,
    USB_PID_IN,
    USB_PID_OUT
} USB_PID;

```

Description

Enumeration of Legal Packet IDs (PIDs)

This data type enumerates the valid (i.e., legal) PID values. While the PID field is four bits long, only these values are legal and should be used. The use of any other values may cause unpredictable results.

Members

Members	Description
USB_PID_SETUP	Setup token
USB_PID_IN	IN token
USB_PID_OUT	OUT token

5.6.26.6.16.10 USB_PING_PONG_MODE Enumeration**C**

```

typedef enum {
    USB_PING_PONG_ALL_BUT_EP0,
    USB_PING_PONG_FULL_PING_PONG,
    USB_PING_PONG_EP0_OUT_ONLY,
    USB_PING_PONG_NO_PING_PONG
} USB_PING_PONG_MODE;

```

Description

Enumeration of USB Ping-Pong Modes for PIC18 and PIC24F

This data type supports the four modes of ping-pong buffering for PIC18 and PIC24F devices.

Members

Members	Description
USB_PING_PONG_ALL_BUT_EP0	Ping-Pong buffering on all endpoints except Endpoint Zero
USB_PING_PONG_FULL_PING_PONG	Ping-Pong buffering on all endpoints
USB_PING_PONG_EP0_OUT_ONLY	Ping-Pong buffering on just Endpoint Zero transmit
USB_PING_PONG_NO_PING_PONG	No ping-pong buffering

Remarks

For PIC18 and PIC24F devices only.

5.6.26.6.16.11 USB_PING_PONG_STATE Enumeration**C**

```

typedef enum {
    USB_PING_PONG_EVEN,
    USB_PING_PONG_ODD
} USB_PING_PONG_STATE;

```

Description

Enumeration of USB Ping-Pong Indicator

This data type decodes which buffer (Even vs. Odd) was used for the last transaction.

Members

Members	Description
USB_PING_PONG_EVEN	Last transaction on Even Buffer
USB_PING_PONG_ODD	Last transaction on Odd Buffer

Remarks

None.

5.6.26.6.16.12 USB_TOKEN_SPEED Enumeration**C**

```
typedef enum {  
    USB_LOWSPEED_TOKENS,  
    USB_FULLSPEED_TOKENS  
} USB_TOKEN_SPEED;
```

Description

USB Token Speeds Enumeration

This data type provides enumeration of available token speeds.

Members

Members	Description
USB_LOWSPEED_TOKENS	Low Speed Tokens
USB_FULLSPEED_TOKENS	Full Speed Tokens

Remarks

For Host mode only.

5.6.26.6.16.13 USB_MAX_EP_NUMBER Macro**C**

```
#define USB_MAX_EP_NUMBER 15
```

Description

Maximum number of endpoints

This constant defines the maximum number of endpoints supported (not including EP0). It is used in dimensioning the Buffer Descriptor Table (BDT) array.

5.6.26.6.17 Feature Existence Functions**5.6.26.6.17.1 PLIB_USB_ExistsActivityPending Function****C**

```
bool PLIB_USB_ExistsActivityPending(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the ActivityPending feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_ActivityPending](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ActivityPending feature is supported on the device
- false - The ActivityPending feature is not supported on the device

Remarks

None.

5.6.26.6.17.2 PLIB_USB_ExistsALL_Interrupt Function

C

```
bool PLIB_USB_ExistsALL_Interrupt(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the ALL_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_AllInterruptEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ALL_Interrupt feature is supported on the device
- false - The ALL_Interrupt feature is not supported on the device

Remarks

None.

5.6.26.6.17.3 PLIB_USB_ExistsAutomaticSuspend Function

C

```
bool PLIB_USB_ExistsAutomaticSuspend(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the AutomaticSuspend feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_AutoSuspendDisable](#)

- [PLIB_USB_AutoSuspendEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The AutomaticSuspend feature is supported on the device
- false - The AutomaticSuspend feature is not supported on the device

Remarks

None.

5.6.26.6.17.4 PLIB_USB_ExistsBDTBaseAddress Function

C

```
bool PLIB_USB_ExistsBDTBaseAddress(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the BDTBaseAddress feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_BDTBaseAddressGet](#)
- [PLIB_USB_BDTBaseAddressSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDTBaseAddress feature is supported on the device
- false - The BDTBaseAddress feature is not supported on the device

Remarks

None.

5.6.26.6.17.5 PLIB_USB_ExistsBDTFunctions Function

C

```
bool PLIB_USB_ExistsBDTFunctions(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the BDTFunctions feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_BufferAddressGet](#)
- [PLIB_USB_BufferAddressSet](#)
- [PLIB_USB_BufferByteCountGet](#)
- [PLIB_USB_BufferByteCountSet](#)
- [PLIB_USB_BufferCancelReleaseToUSB](#)
- [PLIB_USB_BufferAllCancelReleaseToUSB](#)
- [PLIB_USB_BufferClearAll](#)
- [PLIB_USB_BufferDataToggleGet](#)
- [PLIB_USB_BufferDataToggleSelect](#)
- [PLIB_USB_BufferDataToggleSyncEnable](#)
- [PLIB_USB_BufferDataToggleSyncDisable](#)
- [PLIB_USB_BufferIndexGet](#)
- [PLIB_USB_BufferPIDBitsClear](#)
- [PLIB_USB_BufferPIDGet](#)
- [PLIB_USB_BufferReleasedToSW](#)
- [PLIB_USB_BufferReleaseToUSB](#)
- [PLIB_USB_BufferSchedule](#)
- [PLIB_USB_BufferStallDisable](#)
- [PLIB_USB_BufferStallEnable](#)
- [PLIB_USB_BufferStallGet](#)
- [PLIB_USB_BufferEP0RxStatusInitialize](#)
- [PLIB_USB_BufferClearAllDTSEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BDTFunctions feature is supported on the device
- false - The BDTFunctions feature is not supported on the device

Remarks

None.

5.6.26.6.17.6 PLIB_USB_ExistsBufferFreeze Function**C**

```
bool PLIB_USB_ExistsBufferFreeze(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the BufferFreeze feature is available on the USB module. When this function returns true, these

functions are supported on the device:

- [PLIB_USB_PingPongFreeze](#)
- [PLIB_USB_PingPongUnfreeze](#)
- [PLIB_USB_PingPongReset](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The BufferFreeze feature is supported on the device
- false - The BufferFreeze feature is not supported on the device

Remarks

None.

5.6.26.6.17.7 PLIB_USB_ExistsDeviceAddress Function

C

```
bool PLIB_USB_ExistsDeviceAddress(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the DeviceAddress feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_DeviceAddressSet](#)
- [PLIB_USB_DeviceAddressGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The DeviceAddress feature is supported on the device
- false - The DeviceAddress feature is not supported on the device

Remarks

None.

5.6.26.6.17.8 PLIB_USB_ExistsEP0LowSpeedConnect Function

C

```
bool PLIB_USB_ExistsEP0LowSpeedConnect(  
    USB_MODULE_ID index
```

```
);
```

Description

This function identifies whether the EP0LowSpeedConnect feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_EP0LSDirectConnectEnable](#)
- [PLIB_USB_EP0LSDirectConnectDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EP0LowSpeedConnect feature is supported on the device
- false - The EP0LowSpeedConnect feature is not supported on the device

Remarks

None.

5.6.26.6.17.9 PLIB_USB_ExistsEP0NAKRetry Function

C

```
bool PLIB_USB_ExistsEP0NAKRetry(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the EP0NAKRetry feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_EP0NakRetryEnable](#)
- [PLIB_USB_EP0NakRetryDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EP0NAKRetry feature is supported on the device
- false - The EP0NAKRetry feature is not supported on the device

Remarks

None.

5.6.26.6.17.10 PLIB_USB_ExistsEPnRxEnable Function

C

```
bool PLIB_USB_ExistsEPnRxEnable(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the EPnRxEnableEnhanced feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_EPnRxEnable](#)
- [PLIB_USB_EPnRxDisable](#)
- [PLIB_USB_EPnTxEnable](#)
- [PLIB_USB_EPnTxDisable](#)
- [PLIB_USB_EPnHandshakeEnable](#)
- [PLIB_USB_EPnHandshakeDisable](#)
- [PLIB_USB_EPnControlTransferEnable](#)
- [PLIB_USB_EPnControlTransferDisable](#)
- [PLIB_USB_EPnIsStalled](#)
- [PLIB_USB_EPnStallClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EPnRxEnableEnhanced feature is supported on the device
- false - The EPnRxEnableEnhanced feature is not supported on the device

Remarks

None.

5.6.26.6.17.11 PLIB_USB_ExistsEPnTxRx Function

C

```
bool PLIB_USB_ExistsEPnTxRx(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the EPnTxRx feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_EPnTxSelect](#)
- [PLIB_USB_EPnRxSelect](#)
- [PLIB_USB_EPnTxRxSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EPnTxRx feature is supported on the device
- false - The EPnTxRx feature is not supported on the device

Remarks

None.

5.6.26.6.17.12 PLIB_USB_ExistsERR_Interrupt Function**C**

```
bool PLIB_USB_ExistsERR_Interrupt(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the ERR_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_ErrorInterruptEnable](#)
- [PLIB_USB_ErrorInterruptDisable](#)
- [PLIB_USB_ErrorInterruptIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ERR_Interrupt feature is supported on the device
- false - The ERR_Interrupt feature is not supported on the device

Remarks

None.

5.6.26.6.17.13 PLIB_USB_ExistsERR_InterruptStatus Function**C**

```
bool PLIB_USB_ExistsERR_InterruptStatus(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the ERR_InterruptStatus feature is available on the USB module. When this function returns true,

these functions are supported on the device:

- [PLIB_USB_ErrorInterruptFlagSet](#)
- [PLIB_USB_ErrorInterruptFlagClear](#)
- [PLIB_USB_ErrorInterruptFlagGet](#)
- [PLIB_USB_ErrorInterruptFlagAllGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ERR_InterruptStatus feature is supported on the device
- false - The ERR_InterruptStatus feature is not supported on the device

Remarks

None.

5.6.26.6.17.14 PLIB_USB_ExistsEyePattern Function

C

```
bool PLIB_USB_ExistsEyePattern(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the EyePattern feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_EyePatternDisable](#)
- [PLIB_USB_EyePatternEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The EyePattern feature is supported on the device
- false - The EyePattern feature is not supported on the device

Remarks

None.

5.6.26.6.17.15 PLIB_USB_ExistsFrameNumber Function

C

```
bool PLIB_USB_ExistsFrameNumber(  

```

```
    USB_MODULE_ID index
);
```

Description

This function identifies whether the FrameNumber feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_FrameNumberGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The FrameNumber feature is supported on the device
- false - The FrameNumber feature is not supported on the device

Remarks

None.

5.6.26.6.17.16 PLIB_USB_ExistsGEN_Interrupt Function

C

```
bool PLIB_USB_ExistsGEN_Interrupt(
    USB_MODULE_ID index
);
```

Description

This function identifies whether the GEN_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_InterruptEnable](#)
- [PLIB_USB_InterruptDisable](#)
- [PLIB_USB_InterruptIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The GEN_Interrupt feature is supported on the device
- false - The GEN_Interrupt feature is not supported on the device

Remarks

None.

5.6.26.6.17.17 PLIB_USB_ExistsGEN_InterruptStatus Function

C

```
bool PLIB_USB_ExistsGEN_InterruptStatus(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the GEN_InterruptStatus feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_InterruptFlagSet](#)
- [PLIB_USB_InterruptFlagClear](#)
- [PLIB_USB_InterruptFlagGet](#)
- [PLIB_USB_InterruptFlagAllGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The GEN_InterruptStatus feature is supported on the device
- false - The GEN_InterruptStatus feature is not supported on the device

Remarks

None.

5.6.26.6.17.18 PLIB_USB_ExistsHostBusyWithToken Function

C

```
bool PLIB_USB_ExistsHostBusyWithToken(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the HostBusyWithToken feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_IsBusyWithToken](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The HostBusyWithToken feature is supported on the device
- false - The HostBusyWithToken feature is not supported on the device

Remarks

None.

5.6.26.6.17.19 PLIB_USB_ExistsHostGeneratesReset Function**C**

```
bool PLIB_USB_ExistsHostGeneratesReset(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the HostGeneratesReset feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_ResetSignalEnable](#)
- [PLIB_USB_ResetSignalDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The HostGeneratesReset feature is supported on the device
- false - The HostGeneratesReset feature is not supported on the device

Remarks

None.

5.6.26.6.17.20 PLIB_USB_ExistsLastDirection Function**C**

```
bool PLIB_USB_ExistsLastDirection(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the LastDirection feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_LastTransactionDirectionGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LastDirection feature is supported on the device

- false - The LastDirection feature is not supported on the device

Remarks

None.

5.6.26.6.17.21 PLIB_USB_ExistsLastEndpoint Function**C**

```
bool PLIB_USB_ExistsLastEndpoint(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the LastEndpoint feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_LastTransactionEndPtGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LastEndpoint feature is supported on the device
- false - The LastEndpoint feature is not supported on the device

Remarks

None.

5.6.26.6.17.22 PLIB_USB_ExistsLastPingPong Function**C**

```
bool PLIB_USB_ExistsLastPingPong(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the LastPingPong feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_LastTransactionPingPongStateGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LastPingPong feature is supported on the device
- false - The LastPingPong feature is not supported on the device

Remarks

None.

5.6.26.6.17.23 PLIB_USB_ExistsLastTransactionDetails Function**C**

```
bool PLIB_USB_ExistsLastTransactionDetails(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the LastTransactionDetails feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_LastTransactionDetailsGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LastTransactionDetails feature is supported on the device
- false - The LastTransactionDetails feature is not supported on the device

Remarks

None.

5.6.26.6.17.24 PLIB_USB_ExistsLiveJState Function**C**

```
bool PLIB_USB_ExistsLiveJState(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the LiveJState feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_JStateIsActive](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LiveJState feature is supported on the device
- false - The LiveJState feature is not supported on the device

Remarks

None.

5.6.26.6.17.25 PLIB_USB_ExistsLiveSingleEndedZero Function**C**

```
bool PLIB_USB_ExistsLiveSingleEndedZero(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the LiveSingleEndedZero feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_SE0InProgress](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The LiveSingleEndedZero feature is supported on the device
- false - The LiveSingleEndedZero feature is not supported on the device

Remarks

None.

5.6.26.6.17.26 PLIB_USB_ExistsModuleBusy Function**C**

```
bool PLIB_USB_ExistsModuleBusy(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the ModuleBusy feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_ModuleIsBusy](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ModuleBusy feature is supported on the device
- false - The ModuleBusy feature is not supported on the device

Remarks

None.

5.6.26.6.17.27 PLIB_USB_ExistsModulePower Function**C**

```
bool PLIB_USB_ExistsModulePower(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the ModulePower feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_Enable](#)
- [PLIB_USB_Disable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ModulePower feature is supported on the device
- false - The ModulePower feature is not supported on the device

Remarks

None.

5.6.26.6.17.28 PLIB_USB_ExistsNextTokenSpeed Function**C**

```
bool PLIB_USB_ExistsNextTokenSpeed(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the NextTokenSpeed feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_TokenSpeedSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The NextTokenSpeed feature is supported on the device

- false - The NextTokenSpeed feature is not supported on the device

Remarks

None.

5.6.26.6.17.29 PLIB_USB_ExistsOnChipPullup Function**C**

```
bool PLIB_USB_ExistsOnChipPullup(  
    USB_MODULE_ID index  
)  
;
```

Description

This function identifies whether the OnChipPullup feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_OnChipPullUpDisable](#)
- [PLIB_USB_OnChipPullUpEnable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OnChipPullup feature is supported on the device
- false - The OnChipPullup feature is not supported on the device

Remarks

None.

5.6.26.6.17.30 PLIB_USB_ExistsOnChipTransceiver Function**C**

```
bool PLIB_USB_ExistsOnChipTransceiver(  
    USB_MODULE_ID index  
)  
;
```

Description

This function identifies whether the OnChipTransceiver feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_TransceiverEnable](#)
- [PLIB_USB_TransceiverDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OnChipTransceiver feature is supported on the device
- false - The OnChipTransceiver feature is not supported on the device

Remarks

None.

5.6.26.6.17.31 PLIB_USB_ExistsOpModeSelect Function**C**

```
bool PLIB_USB_ExistsOpModeSelect(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OpModeSelect feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_OperatingModeSelect](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OpModeSelect feature is supported on the device
- false - The OpModeSelect feature is not supported on the device

Remarks

None.

5.6.26.6.17.32 PLIB_USB_ExistsOTG_ASessionValid Function**C**

```
bool PLIB_USB_ExistsOTG_ASessionValid(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_ASessionValid feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_OTG_VBusValid](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_ASessionValid feature is supported on the device
- false - The OTG_ASessionValid feature is not supported on the device

Remarks

None.

5.6.26.6.17.33 PLIB_USB_ExistsOTG_BSessionEnd Function**C**

```
bool PLIB_USB_ExistsOTG_BSessionEnd(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_BSessionEnd feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_OTG_BSessionHasEnded](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_BSessionEnd feature is supported on the device
- false - The OTG_BSessionEnd feature is not supported on the device

Remarks

None.

5.6.26.6.17.34 PLIB_USB_ExistsOTG_IDPinState Function**C**

```
bool PLIB_USB_ExistsOTG_IDPinState(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_IDPinState feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_OTG_IDPinStateIsTypeA](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_IDPinState feature is supported on the device
- false - The OTG_IDPinState feature is not supported on the device

Remarks

None.

5.6.26.6.17.35 PLIB_USB_ExistsOTG_Interrupt Function**C**

```
bool PLIB_USB_ExistsOTG_Interrupt(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_OTG_InterruptEnable](#)
- [PLIB_USB_OTG_InterruptDisable](#)
- [PLIB_USB_OTG_InterruptIsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_Interrupt feature is supported on the device
- false - The OTG_Interrupt feature is not supported on the device

Remarks

None.

5.6.26.6.17.36 PLIB_USB_ExistsOTG_InterruptStatus Function**C**

```
bool PLIB_USB_ExistsOTG_InterruptStatus(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_InterruptStatus feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_OTG_InterruptFlagSet](#)
- [PLIB_USB_OTG_InterruptFlagClear](#)
- [PLIB_USB_OTG_InterruptFlagGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_InterruptStatus feature is supported on the device
- false - The OTG_InterruptStatus feature is not supported on the device

Remarks

None.

5.6.26.6.17.37 PLIB_USB_ExistsOTG_LineState Function**C**

```
bool PLIB_USB_ExistsOTG_LineState(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_LineState feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_OTG_LineStateIsStable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_LineState feature is supported on the device
- false - The OTG_LineState feature is not supported on the device

Remarks

None.

5.6.26.6.17.38 PLIB_USB_ExistsOTG_PullUpPullDown Function**C**

```
bool PLIB_USB_ExistsOTG_PullUpPullDown(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_PullUpPullDown feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_OTG_PullUpPullDownSetup](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_PullUpPullDown feature is supported on the device
- false - The OTG_PullUpPullDown feature is not supported on the device

Remarks

None.

5.6.26.6.17.39 PLIB_USB_ExistsOTG_SessionValid Function**C**

```
bool PLIB_USB_ExistsOTG_SessionValid(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_SessionValid feature is available on the USB module. When this function returns true, this function is supported on the device:

- [PLIB_USB_OTG_SessionValid](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_SessionValid feature is supported on the device
- false - The OTG_SessionValid feature is not supported on the device

Remarks

None.

5.6.26.6.17.40 PLIB_USB_ExistsOTG_VbusCharge Function**C**

```
bool PLIB_USB_ExistsOTG_VbusCharge(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_VbusCharge feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_OTG_VBusChargeEnable](#)

- [PLIB_USB_OTG_VBusChargeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_VbusCharge feature is supported on the device
- false - The OTG_VbusCharge feature is not supported on the device

Remarks

None.

5.6.26.6.17.41 PLIB_USB_ExistsOTG_VbusDischarge Function**C**

```
bool PLIB_USB_ExistsOTG_VbusDischarge(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_VbusDischarge feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_OTG_VBusDischargeEnable](#)
- [PLIB_USB_OTG_VBusDischargeDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_VbusDischarge feature is supported on the device
- false - The OTG_VbusDischarge feature is not supported on the device

Remarks

None.

5.6.26.6.17.42 PLIB_USB_ExistsOTG_VbusPowerOnOff Function**C**

```
bool PLIB_USB_ExistsOTG_VbusPowerOnOff(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the OTG_VbusPowerOnOff feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_OTG_VBusPowerOff](#)
- [PLIB_USB_OTG_VBusPowerOn](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The OTG_VbusPowerOnOff feature is supported on the device
- false - The OTG_VbusPowerOnOff feature is not supported on the device

Remarks

None.

5.6.26.6.17.43 PLIB_USB_ExistsPacketTransfer Function**C**

```
bool PLIB_USB_ExistsPacketTransfer(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the PacketTransfer feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_PacketTransferIsDisabled](#)
- [PLIB_USB_PacketTransferEnable](#)
- [PLIB_USB_PacketTransferDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PacketTransfer feature is supported on the device
- false - The PacketTransfer feature is not supported on the device

Remarks

None.

5.6.26.6.17.44 PLIB_USB_ExistsPingPongMode Function**C**

```
bool PLIB_USB_ExistsPingPongMode(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the PingPongMode feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_PingPongModeSelect](#)
- [PLIB_USB_PingPongModeGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PingPongMode feature is supported on the device
- false - The PingPongMode feature is not supported on the device

Remarks

None.

5.6.26.6.17.45 PLIB_USB_ExistsResumeSignaling Function**C**

```
bool PLIB_USB_ExistsResumeSignaling(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the ResumeSignaling feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_ResumeSignalingEnable](#)
- [PLIB_USB_ResumeSignalingDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The ResumeSignaling feature is supported on the device
- false - The ResumeSignaling feature is not supported on the device

Remarks

None.

5.6.26.6.17.46 PLIB_USB_ExistsSleepEntryGuard Function

C

```
bool PLIB_USB_ExistsSleepEntryGuard(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the SleepEntryGuard feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_SleepGuardEnable](#)
- [PLIB_USB_SleepGuardDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SleepEntryGuard feature is supported on the device
- false - The SleepEntryGuard feature is not supported on the device

Remarks

None.

5.6.26.6.17.47 PLIB_USB_ExistsSOFThreshold Function

C

```
bool PLIB_USB_ExistsSOFThreshold(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the SOFThreshold feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_SOFThresholdGet](#)
- [PLIB_USB_SOFThresholdSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SOFThreshold feature is supported on the device
- false - The SOFThreshold feature is not supported on the device

Remarks

None.

5.6.26.6.17.48 PLIB_USB_ExistsSpeedControl Function**C**

```
bool PLIB_USB_ExistsSpeedControl(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the SpeedControl feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_FullSpeedEnable](#)
- [PLIB_USB_FullSpeedDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The SpeedControl feature is supported on the device
- false - The SpeedControl feature is not supported on the device

Remarks

None.

5.6.26.6.17.49 PLIB_USB_ExistsStartOfFrames Function**C**

```
bool PLIB_USB_ExistsStartOfFrames(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the StartOfFrames feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_SOFEnable](#)
- [PLIB_USB_SOFDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StartOfFrames feature is supported on the device
- false - The StartOfFrames feature is not supported on the device

Remarks

None.

5.6.26.6.17.50 PLIB_USB_ExistsStopInIdle Function**C**

```
bool PLIB_USB_ExistsStopInIdle(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the StopInIdle feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_StopInIdleEnable](#)
- [PLIB_USB_StopInIdleDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Remarks

None.

5.6.26.6.17.51 PLIB_USB_ExistsSuspend Function**C**

```
bool PLIB_USB_ExistsSuspend(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the Suspend feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_SuspendEnable](#)
- [PLIB_USB_SuspendDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The Suspend feature is supported on the device
- false - The Suspend feature is not supported on the device

Remarks

None.

5.6.26.6.17.52 PLIB_USB_ExistsTokenEP Function**C**

```
bool PLIB_USB_ExistsTokenEP(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the TokenEP feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB-TokenEPGet](#)
- [PLIB_USB-TokenEPSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TokenEP feature is supported on the device
- false - The TokenEP feature is not supported on the device

Remarks

None.

5.6.26.6.17.53 PLIB_USB_ExistsTokenPID Function**C**

```
bool PLIB_USB_ExistsTokenPID(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the TokenPID feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB-TokenPIDGet](#)
- [PLIB_USB-TokenPIDSet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TokenPID feature is supported on the device
- false - The TokenPID feature is not supported on the device

Remarks

None.

5.6.26.6.17.54 PLIB_USB_ExistsUOEMonitor Function

C

```
bool PLIB_USB_ExistsUOEMonitor(  
    USB_MODULE_ID index  
);
```

Description

This function identifies whether the UOEMonitor feature is available on the USB module. When this function returns true, these functions are supported on the device:

- [PLIB_USB_UOEMonitorEnable](#)
- [PLIB_USB_UOEMonitorDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The UOEMonitor feature is supported on the device
- false - The UOEMonitor feature is not supported on the device

Remarks

None.

5.6.26.7 Files

Files

Name	Description
plib_usb.h	USB PLIB Interface Header for common definitions

Description

5.6.26.7.1 plib_usb.h

USB Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB Peripheral Library.

Enumerations

	Name	Description
	<u>USB_BUFFER_DATA01</u>	Provides enumeration data toggle for a buffer.
	<u>USB_BUFFER_DIRECTION</u>	Provides enumeration transmit/receive direction for a buffer.
	<u>USB_BUFFER_PING_PONG</u>	Enumerates the ping-pong buffer (Even vs. Odd).
	<u>USB_BUFFER_SCHEDULE_DATA01</u>	Provides enumeration data toggle for a buffer.
	<u>USB_EP_TXRX</u>	Provides enumeration transmit/receive setup for an endpoint.
	<u>USB_OPMODES</u>	Provides enumeration of operating modes supported by USB.
	<u>USB_OTG_INTERRUPTS</u>	Provides enumeration of interrupts related to the USB On-The-Go (OTG) module.
	<u>USB_OTG_PULL_UP_PULL_DOWN</u>	USB OTG pull-Up and pull-Down resistors for D+ and D- .
	<u>USB_PID</u>	Legal PID values.
	<u>USB_PING_PONG_MODE</u>	Supports the four modes of ping-pong buffering for PIC18 and PIC24F devices.
	<u>USB_PING_PONG_STATE</u>	Decodes which buffer (Even vs. Odd) was used for the last transaction.
	<u>USB_TOKEN_SPEED</u>	Provides enumeration of available token speeds.

Functions

	Name	Description
	<u>PLIB_USB_ActivityPending</u>	Returns whether or not USB activity is pending.
	<u>PLIB_USB_AllInterruptEnable</u>	Configures the USB peripheral general interrupts, error interrupts and OTG interrupts.
	<u>PLIB_USB_AutoSuspendDisable</u>	Disables USB OTG Auto-suspend mode.
	<u>PLIB_USB_AutoSuspendEnable</u>	Enables USB Auto-suspend mode.
	<u>PLIB_USB_BDTBaseAddressGet</u>	Returns the base address of the Buffer Descriptor Table.
	<u>PLIB_USB_BDTBaseAddressSet</u>	Sets the base address for the Buffer Descriptor Table for PIC24 and PIC32 devices.
	<u>PLIB_USB_BufferAddressGet</u>	Gets the memory address of an endpoint buffer.
	<u>PLIB_USB_BufferAddressSet</u>	Sets the endpoint buffer address.
	<u>PLIB_USB_BufferAllCancelReleaseToUSB</u>	Cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.
	<u>PLIB_USB_BufferByteCountGet</u>	Returns the endpoint buffer byte count.
	<u>PLIB_USB_BufferByteCountSet</u>	Sets the buffer byte count.
	<u>PLIB_USB_BufferCancelReleaseToUSB</u>	Cancels release of the endpoint buffer by software, allowing software to again access the buffer.
	<u>PLIB_USB_BufferClearAll</u>	Clears (zeros out) entries in the Buffer Descriptor Table.
	<u>PLIB_USB_BufferClearAllDTSEnable</u>	Clears the endpoint descriptor entry and enables data toggle synchronization.
	<u>PLIB_USB_BufferDataToggleGet</u>	Returns data synchronization (DATA0 or DATA1) for the endpoint buffer.
	<u>PLIB_USB_BufferDataToggleSelect</u>	Sets the endpoint buffer to DATA0 or DATA1.

	PLIB_USB_BufferDataToggleSyncDisable	Disables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferDataToggleSyncEnable	Enables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferEP0RxStatusInitialize	Initializes the Endpoint 0 RX endpoint buffer descriptors.
	PLIB_USB_BufferIndexGet	Gets the Buffer Descriptor Table index for a buffer.
	PLIB_USB_BufferPIDBitsClear	Clears the Buffer Status bits in the Buffer Descriptor Table.
	PLIB_USB_BufferPIDGet	Returns the token packet ID (PID) from the endpoint buffer status.
	PLIB_USB_BufferReleasedToSW	Returns the boolean flag value of 'true' when the buffer has been released by the USB module.
	PLIB_USB_BufferReleaseToUSB	Releases the endpoint buffer by software, allowing the USB module access to the buffer.
	PLIB_USB_BufferSchedule	Hands over a buffer to the USB module along with the buffer address and byte count.
	PLIB_USB_BufferStallDisable	Disables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallEnable	Enables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallGet	Returns the buffer stall status for an endpoint/direction/ping-pong.
	PLIB_USB_DeviceAddressGet	Returns the address of the USB module in Device mode.
	PLIB_USB_DeviceAddressSet	Sets the USB Device's address.
	PLIB_USB_Disable	Disables (powers down) the USB module.
	PLIB_USB_Enable	Enables (powers up) the USB module.
	PLIB_USB_EP0HostSetup	Sends token to the specified address.
	PLIB_USB_EP0LSDirectConnectDisable	Disables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0LSDirectConnectEnable	Enables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0NakRetryDisable	Disables retrying of NAKed transactions.
	PLIB_USB_EP0NakRetryEnable	Enables retrying NAK'd transactions for Endpoint 0.
	PLIB_USB_EPnAttributesClear	Clears the set attributes of the specified endpoint.
	PLIB_USB_EPnAttributesSet	Configures attributes of the endpoint such as direction, handshake capability and direction.
	PLIB_USB_EPnControlTransferDisable	Disables endpoint control transfers.
	PLIB_USB_EPnControlTransferEnable	Enables endpoint control transfers.
	PLIB_USB_EPnDirectionDisable	Disables the specified endpoint direction.
	PLIB_USB_EPnHandshakeDisable	Disables endpoint handshaking.
	PLIB_USB_EPnHandshakeEnable	Enables endpoint handshaking.
	PLIB_USB_EPnIsStalled	Tests whether the endpoint epValue is stalled.
	PLIB_USB_EPnRxDisable	Disables an endpoint's ability to process IN tokens.
	PLIB_USB_EPnRxEnable	Enables an endpoint to process IN tokens.
	PLIB_USB_EPnRxSelect	Selects receive capabilities of an endpoint.
	PLIB_USB_EPnStallClear	Clears an endpoint's stalled flag.
	PLIB_USB_EPnTxDisable	Disables an endpoint's ability to process OUT tokens.
	PLIB_USB_EPnTxEnable	Enables an endpoint to process OUT tokens.
	PLIB_USB_EPnTxRxSelect	Selects transmit and/or receive capabilities of an endpoint.
	PLIB_USB_EPnTxSelect	Selects transmit capabilities of an endpoint.
	PLIB_USB_ErrorInterruptDisable	Disables an error interrupt for the USB module.
	PLIB_USB_ErrorInterruptEnable	Enables an error interrupt for the USB module.
	PLIB_USB_ErrorInterruptFlagAllGet	Returns a logically ORed bit map of active error interrupt flags.
	PLIB_USB_ErrorInterruptFlagClear	Clears an error interrupt flag for the USB module.

	PLIB_USB_ErrorInterruptFlagGet	Tests an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagSet	Sets an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptIsEnabled	Returns true if interrupts are enabled.
	PLIB_USB_ExistsActivityPending	Identifies whether the ActivityPending feature exists on the USB module.
	PLIB_USB_ExistsALL_Interrupt	Identifies whether the ALL_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsAutomaticSuspend	Identifies whether the AutomaticSuspend feature exists on the USB module.
	PLIB_USB_ExistsBDTBaseAddress	Identifies whether the BDTBaseAddress feature exists on the USB module.
	PLIB_USB_ExistsBDTFunctions	Identifies whether the BDTFunctions feature exists on the USB module.
	PLIB_USB_ExistsBufferFreeze	Identifies whether the BufferFreeze feature exists on the USB module.
	PLIB_USB_ExistsDeviceAddress	Identifies whether the DeviceAddress feature exists on the USB module.
	PLIB_USB_ExistsEP0LowSpeedConnect	Identifies whether the EP0LowSpeedConnect feature exists on the USB module.
	PLIB_USB_ExistsEP0NAKRetry	Identifies whether the EP0NAKRetry feature exists on the USB module.
	PLIB_USB_ExistsEPnRxEnable	Identifies whether the EPnRxEnableEnhanced feature exists on the USB module.
	PLIB_USB_ExistsEPnTxRx	Identifies whether the EPnTxRx feature exists on the USB module.
	PLIB_USB_ExistsERR_Interrupt	Identifies whether the ERR_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsERR_InterruptStatus	Identifies whether the ERR_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsEyePattern	Identifies whether the EyePattern feature exists on the USB module.
	PLIB_USB_ExistsFrameNumber	Identifies whether the FrameNumber feature exists on the USB module.
	PLIB_USB_ExistsGEN_Interrupt	Identifies whether the GEN_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsGEN_InterruptStatus	Identifies whether the GEN_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsHostBusyWithToken	Identifies whether the HostBusyWithToken feature exists on the USB module.
	PLIB_USB_ExistsHostGeneratesReset	Identifies whether the HostGeneratesReset feature exists on the USB module.
	PLIB_USB_ExistsLastDirection	Identifies whether the LastDirection feature exists on the USB module.
	PLIB_USB_ExistsLastEndpoint	Identifies whether the LastEndpoint feature exists on the USB module.
	PLIB_USB_ExistsLastPingPong	Identifies whether the LastPingPong feature exists on the USB module.
	PLIB_USB_ExistsLastTransactionDetails	Identifies whether the LastTransactionDetails feature exists on the USB module.
	PLIB_USB_ExistsLiveJState	Identifies whether the LiveJState feature exists on the USB module.
	PLIB_USB_ExistsLiveSingleEndedZero	Identifies whether the LiveSingleEndedZero feature exists on the USB module.

	<u>PLIB_USB_ExistsModuleBusy</u>	Identifies whether the ModuleBusy feature exists on the USB module.
	<u>PLIB_USB_ExistsModulePower</u>	Identifies whether the ModulePower feature exists on the USB module.
	<u>PLIB_USB_ExistsNextTokenSpeed</u>	Identifies whether the NextTokenSpeed feature exists on the USB module.
	<u>PLIB_USB_ExistsOnChipPullup</u>	Identifies whether the OnChipPullup feature exists on the USB module.
	<u>PLIB_USB_ExistsOnChipTransceiver</u>	Identifies whether the OnChipTransceiver feature exists on the USB module.
	<u>PLIB_USB_ExistsOpModeSelect</u>	Identifies whether the OpModeSelect feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_ASessionValid</u>	Identifies whether the OTG_ASessionValid feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_BSessionEnd</u>	Identifies whether the OTG_BSessionEnd feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_IDPinState</u>	Identifies whether the OTG_IDPinState feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_Interrupt</u>	Identifies whether the OTG_Interrupt feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_InterruptStatus</u>	Identifies whether the OTG_InterruptStatus feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_LineState</u>	Identifies whether the OTG_LineState feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_PullUpPullDown</u>	Identifies whether the OTG_PullUpPullDown feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_SessionValid</u>	Identifies whether the OTG_SessionValid feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_VbusCharge</u>	Identifies whether the OTG_VbusCharge feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_VbusDischarge</u>	Identifies whether the OTG_VbusDischarge feature exists on the USB module.
	<u>PLIB_USB_ExistsOTG_VbusPowerOnOff</u>	Identifies whether the OTG_VbusPowerOnOff feature exists on the USB module.
	<u>PLIB_USB_ExistsPacketTransfer</u>	Identifies whether the PacketTransfer feature exists on the USB module.
	<u>PLIB_USB_ExistsPingPongMode</u>	Identifies whether the PingPongMode feature exists on the USB module.
	<u>PLIB_USB_ExistsResumeSignaling</u>	Identifies whether the ResumeSignaling feature exists on the USB module.
	<u>PLIB_USB_ExistsSleepEntryGuard</u>	Identifies whether the SleepEntryGuard feature exists on the USB module.
	<u>PLIB_USB_ExistsSOFThreshold</u>	Identifies whether the SOFThreshold feature exists on the USB module.
	<u>PLIB_USB_ExistsSpeedControl</u>	Identifies whether the SpeedControl feature exists on the USB module.
	<u>PLIB_USB_ExistsStartOfFrames</u>	Identifies whether the StartOfFrames feature exists on the USB module.
	<u>PLIB_USB_ExistsStopInIdle</u>	Identifies whether the StopInIdle feature exists on the USB module.
	<u>PLIB_USB_ExistsSuspend</u>	Identifies whether the Suspend feature exists on the USB module.
	<u>PLIB_USB_ExistsTokenEP</u>	Identifies whether the TokenEP feature exists on the USB module.

	PLIB_USB_ExistsTokenPID	Identifies whether the TokenPID feature exists on the USB module.
	PLIB_USB_ExistsUOEMonitor	Identifies whether the UOEMonitor feature exists on the USB module.
	PLIB_USB_ExternalComparatorMode2Pin	Sets the 2-pin input configuration for VBUS comparators.
	PLIB_USB_ExternalComparatorMode3Pin	Sets the 3-pin input configuration for VBUS Comparators.
	PLIB_USB_EyePatternDisable	Disables the USB eye pattern test.
	PLIB_USB_EyePatternEnable	Enables USB eye pattern test.
	PLIB_USB_FrameNumberGet	Returns the USB frame number.
	PLIB_USB_FullSpeedDisable	Forces the USB module to operate at low speed.
	PLIB_USB_FullSpeedEnable	Enables the USB to operate at full speed.
	PLIB_USB_I2CInterfaceForExtModuleDisable	Specifies external module(s) are controlled via dedicated pins.
	PLIB_USB_I2CInterfaceForExtModuleEnable	Specifies external module(s) are controlled via the I2C interface.
	PLIB_USB_InterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USB_InterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USB_InterruptEnableGet	Returns the enable/disable status of general USB module interrupts
	PLIB_USB_InterruptFlagAllGet	Returns a logically ORed bit map of active general USB interrupt flags.
	PLIB_USB_InterruptFlagClear	Clears a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagGet	Tests a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagSet	Sets a general interrupt flag for the USB module.
	PLIB_USB_InterruptIsEnabled	Returns true if interrupts are enabled.
	PLIB_USB_IsBusyWithToken	Indicates whether there is a token being executed by the USB module as Host.
	PLIB_USB_JStatIsActive	Live differential receiver J State flag.
	PLIB_USB_LastTransactionDetailsGet	Returns the details of the last completed transaction.
	PLIB_USB_LastTransactionDirectionGet	Indicates the direction of the last transaction.
	PLIB_USB_LastTransactionEndPtGet	Returns the endpoint number of the last USB transfer.
	PLIB_USB_LastTransactionPingPongStateGet	Indicates whether the last transaction was to an EVEN buffer or an ODD buffer.
	PLIB_USB_ModuleIsBusy	Indicates if the USB module is not ready to be enabled.
	PLIB_USB_OnChipPullUpDisable	Disables on-chip pull-ups.
	PLIB_USB_OnChipPullUpEnable	Enables on-chip pull-ups.
	PLIB_USB_OperatingModeSelect	Selects the operating mode of the USB module.
	PLIB_USB_OTG_BSessionHasEnded	Returns the status of the B-Session End Indicator bit.
	PLIB_USB_OTG_IDPinStateIsTypeA	Returns the ID Pin state.
	PLIB_USB_OTG_InterruptDisable	Disables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptEnable	Enables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptFlagClear	Clears a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagGet	Tests a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagSet	Sets a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptIsEnabled	Returns whether or not interrupts are enabled.
	PLIB_USB_OTG_LineStateIsStable	Returns the status of the Line Stable Indicator bit.
	PLIB_USB_OTG_PullUpPullDownSetup	Enables or disables pull-up and pull-down resistors.
	PLIB_USB_OTG_SessionValid	Returns the status of the Session Valid Indicator bit.
	PLIB_USB_OTG_VBusChargeDisable	Disables VBUS line charge.
	PLIB_USB_OTG_VBusChargeEnable	Enables the VBUS line to be charged through a pull-up resistor.

	PLIB_USB_OTG_VBusChargeTo3V	Sets the VBUS line to charge to 3.3V.
	PLIB_USB_OTG_VBusChargeTo5V	Sets the VBUS line to charge to 5V.
	PLIB_USB_OTG_VBusDischargeDisable	Disables VBUS line discharge.
	PLIB_USB_OTG_VBusDischargeEnable	Enables VBUS line to be discharged through a resistor.
	PLIB_USB_OTG_VBusPowerOff	Turns off power on the VBUS Line.
	PLIB_USB_OTG_VBusPowerOn	Turns on power for the VBUS line.
	PLIB_USB_OTG_VBusValid	Returns the status of the A-VBUS valid indicator.
	PLIB_USB_PacketTransferDisable	Disables the Serial Interface Engine (SIE).
	PLIB_USB_PacketTransferEnable	Re-enables the Serial Interface Engine (SIE), allowing token and packet processing.
	PLIB_USB_PacketTransferIsDisabled	Indicates that a setup token has been received from the Host and that token/packet processing is disabled.
	PLIB_USB_PingPongFreeze	Resets all Ping-Pong buffer pointers to even buffers.
	PLIB_USB_PingPongModeGet	Returns the Ping-Pong Configuration setting.
	PLIB_USB_PingPongModeSelect	Selects the Ping-Pong Configuration setting.
	PLIB_USB_PingPongReset	Resets the USB peripheral internal Ping-Pong indicator to point to even buffers.
	PLIB_USB_PingPongUnfreeze	Enables Ping-Pong buffering.
	PLIB_USB_PWMCounterDisable	Disables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMCounterEnable	Enables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMDisable	Disables the PWM Generator.
	PLIB_USB_PWMEEnable	Enables the PWM Generator.
	PLIB_USB_PWMPolarityActiveLow	Sets the PWM output to active-high and resets low.
	PLIB_USB_PWMPolarityActiveHigh	Sets the PWM output to active-low and resets high.
	PLIB_USB_ResetSignalDisable	Disables reset signalling on the USB bus.
	PLIB_USB_ResetSignalEnable	Enables reset signalling on the USB bus.
	PLIB_USB_ResumeSignalingDisable	Disables resume signaling.
	PLIB_USB_ResumeSignalingEnable	Enables resume signaling.
	PLIB_USB_SE0InProgress	Returns whether a single-ended zero event is in progress.
	PLIB_USB_SleepGuardDisable	This function disables Sleep Guard. Entry into Sleep mode is immediate.
	PLIB_USB_SleepGuardEnable	Entry into Sleep mode is blocked if bus activity is detected or if an interrupt is pending.
	PLIB_USB_SOFDisable	Disables the automatic generation of the SOF token.
	PLIB_USB_SOFEnable	Enables the automatic generation of the SOF token every 1 ms.
	PLIB_USB_SOFThresholdGet	Returns the Start-of-Frame (SOF) Count bits.
	PLIB_USB_SOFThresholdSet	Sets the Start-of-Frame (SOF) threshold value.
	PLIB_USB_StopInIdleDisable	Allows the USB module to continue operation when the device enters Idle mode.
	PLIB_USB_StopInIdleEnable	Enables USB module operation to stop when the device enters Idle mode.
	PLIB_USB_SuspendDisable	Disables USB OTG Suspend mode.
	PLIB_USB_SuspendEnable	Enables USB Suspend mode.
	PLIB_USB-TokenEPGet	Returns the specified Endpoint address.
	PLIB_USB-TokenEPSet	Sets the Endpoint address for a host transaction.

	PLIB_USB-TokenPIDGet	Returns the token transaction type.
	PLIB_USB-TokenPIDSet	Sets the token transaction type to pidValue.
	PLIB_USB-TokenSend	Sends token to the specified address.
	PLIB_USB-TokenSpeedSelect	Selects low speed or full speed for subsequent token executions.
	PLIB_USB-TransceiverDisable	Disables the on-chip transceiver
	PLIB_USB-TransceiverEnable	Enables the on-chip transceiver.
	PLIB_USB-UOEMonitorDisable	Disables the OE signal output.
	PLIB_USB-UOEMonitorEnable	Enables the OE signal output.
	PLIB_USB-VBoostDisable	Disables the On-Chip 5V Boost Regulator Circuit Disabled bit.
	PLIB_USB-VBoostEnable	Enables the On-Chip 5V Boost Regulator Circuit Enabled bit.
	PLIB_USB-VBUSComparatorDisable	Disables the on-chip VBUS Comparator.
	PLIB_USB-VBUSComparatorEnable	Enables the on-chip VBUS Comparator.
	PLIB_USB-VBUSPullUpDisable	Disables the pull-up on the VBUS pin.
	PLIB_USB-VBUSPullUpEnable	Enables the pull-up on the VBUS pin.

Macros

	Name	Description
	USB_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

File Name

plib_usb.h

Company

Microchip Technology Inc.

5.6.27 Watchdog Timer Peripheral Library

5.6.27.1 Introduction

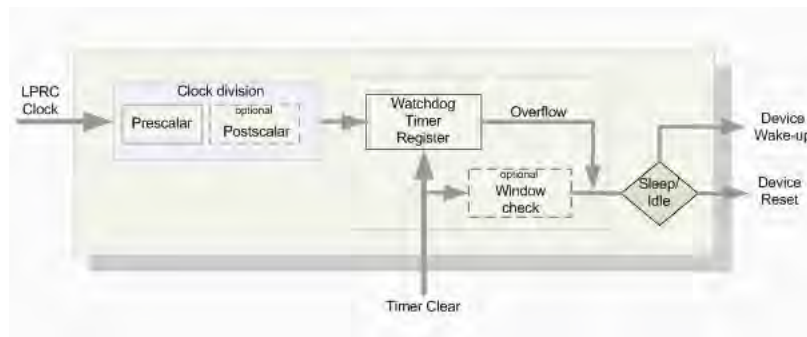
Watchdog Timer (WDT) Peripheral Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the WDT Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description



The primary function of the Watchdog Timer (WDT) is to reset the microcontroller, in the event of a software malfunction, by resetting the device if it has not been cleared by software. To ensure that application doesn't hang, the application is supposed to reset the timer periodically. It can also be used to wake the device from Sleep or Idle mode. The WDT is a free-running timer, which uses the Low-Power RC (LPRC) Oscillator and requires no external components. Therefore, the WDT will continue to operate even if the system's primary clock source is stopped.

5.6.27.2 Release Notes

MPLAB Harmony Version: v0.70b **Watchdog Timer Peripheral Library Version :** 0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.6.27.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.6.27.4 How the Library Works

This section describes how the WDT Peripheral Library works.

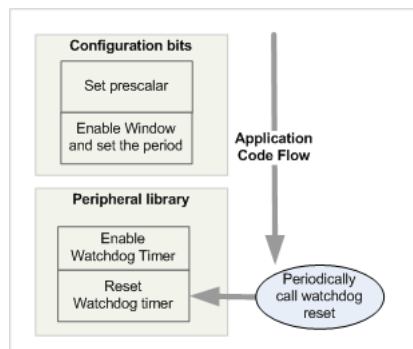
Description

Initializing the WDT module involves these processes.

- Selecting the postscaler
- Enabling the WDT using [PLIB_WDT_Enable](#)
- Selecting the window size if the application wants the WDT in Windowed mode using [PLIB_WDT_WindowEnable](#)

Note: Selecting the postscaler and window size should be done through the Configuration bits

The application should periodically clear the timer once it is enabled; otherwise, a time-out will lead to a device Reset. Use [PLIB_WDT_TimerClear](#) to clear the WDT. A timer clear before a time-out may also lead to a reset in Windowed mode. The user application can clear the window only in the allowed window. Refer to the specific device data sheet to determine the allowed window period for your device.



Example

```

/*****
 * This code fragment assumes the WDT is not enabled through
 * device configuration bits. The Postscaler value must be set
 * through the device configuration and window size should be
 * set if applicable
 *****/

/* Initializing the watchdog involves */
PLIB_WDT_Enable(WDT_ID_0);

//Application loop
while(1)
{
    PLIB_WDT_TimerClear(WDT_ID_0);

    //user code
}
  
```

Note: Refer to the specific device data sheet for information on the allowed window periods for your device and other Configuration bit settings.

5.6.27.4.1 Using the Library

This topic describes the basic architecture of the WDT Peripheral Library and provides information and examples on how to use it.

Interface Header File: [plib_wdt.h](#)

The interface to the WDT Peripheral Library is defined in the "[plib_wdt.h](#)" header file. Any C language source (.c) file that uses the WDT Peripheral Library should include "peripheral.h".

Library File:

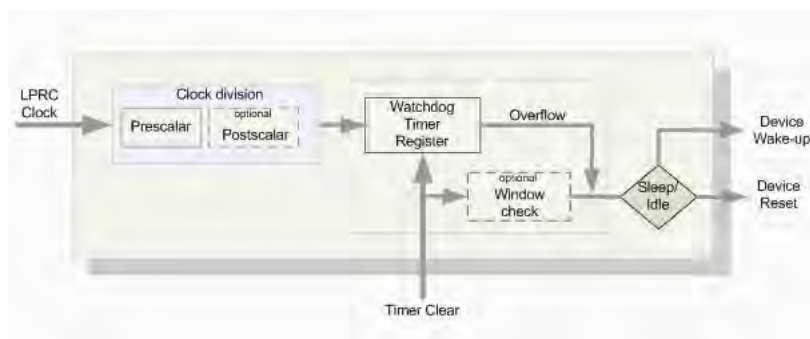
The WDT Peripheral Library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the library interacts with the framework.

5.6.27.4.1.1 Hardware Abstraction Model

This library provides the low-level abstraction of the *Watchdog timer* module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all of the functionality of the available Watchdog Timer modules on the device. Refer to the specific device data sheet or related family reference manual section to determine the set of functions that are supported for each Watchdog Timer module on your device.

Description**Watchdog Timer Software Abstraction Block Diagram**

The WDT uses the internal Low-Power RC (LPRC) Oscillator as the clock source. The clock is divided by the configured prescaler value. There may be one more postscaler divisors. The divided clock is then used to increment a counter. The WDT module uses the watchdog register as a timer. If there is no reset signal from the software and if the counter overflows, this results in a reset in normal mode. In the case of Sleep or Idle mode, the overflow will result in a device wake-up.

For Windowed mode, resetting the counter when the count is not in the specified window will also lead to a reset. If the device is in Sleep or Idle mode, the reset signal will be used to wake-up the device.

5.6.27.4.1.2 Library Overview

Library Interface Section	Description
General Configuration Functions	This includes module enable, module disable, watchdog timer window enable, watchdog timer window disable and the timer reset routines.
General Status Functions	Status routines for the WDT.

5.6.27.5 Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB IDE.

5.6.27.6 Library Interface

Data Types and Constants

	Name	Description
	WDT_MODULE_ID	Identifies the supported WDT modules.

Feature Existence Functions

	Name	Description
	PLIB_WDT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the WDT module.
	PLIB_WDT_ExistsPostscalarValue	Identifies whether the PostscalarValue feature exists on the WDT module.
	PLIB_WDT_ExistsTimerClear	Identifies whether the TimerClear feature exists on the WDT module.
	PLIB_WDT_ExistsWindowEnable	Identifies whether the WindowEnable feature exists on the WDT module.

General Configuration Functions

	Name	Description
	PLIB_WDT_Disable	Disables the WDT module.
	PLIB_WDT_Enable	Enables the WDT module.
	PLIB_WDT_WindowDisable	Disables the WDT Windowed mode.
	PLIB_WDT_WindowEnable	Enables the WDT Window mode.
	PLIB_WDT_TimerClear	Resets the WDT module.

General Status Functions

	Name	Description
	PLIB_WDT_IsEnabled	Returns the watchdog timer on/off(enable/disable) status.
	PLIB_WDT_PostscalarValueGet	Returns the WDT postscalar value.

Description

This section describes the functions, data types, and constants available in the WDT Peripheral Library.

5.6.27.6.1 General Configuration Functions

5.6.27.6.1.1 PLIB_WDT_Disable Function

C

```
void PLIB_WDT_Disable(
    WDT_MODULE_ID index
);
```

Description

This function disables the WDT module if it is enabled in software.

Preconditions

The WDT module must be enabled through software.

Parameters

Parameters	Description
index	Identifies the desired WDT module

Returns

None.

Remarks

This function will not disable the WDT module if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_WDT_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_WDT_Disable ( WDT_ID_0 );
```

5.6.27.6.1.2 PLIB_WDT_Enable Function**C**

```
void PLIB_WDT_Enable(  
    WDT_MODULE_ID index  
);
```

Description

This function enables the WDT module. If it is already enabled through the Configuration bits, it will keep it enabled.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired WDT module

Returns

None.

Remarks

Calling this function is not necessary if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_WDT_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
PLIB_WDT_Enable ( WDT_ID_0 );
```

5.6.27.6.1.3 PLIB_WDT_WindowDisable Function**C**

```
void PLIB_WDT_WindowDisable(  
    WDT_MODULE_ID index  
);
```

Description

This function disables the WDT Windowed mode.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired WDT module

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_WDT_ExistsWindowEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_WDT_WindowDisable ( WDT_ID_0 );
```

5.6.27.6.1.4 PLIB_WDT_WindowEnable Function**C**

```
void PLIB_WDT_WindowEnable(  
    WDT_MODULE_ID index  
);
```

Description

This function enables the WDT Windowed mode.

Preconditions

The window size must be set through the Configuration bits.

Parameters

Parameters	Description
index	Identifies the desired WDT module

Returns

None.

Remarks

The window size must be set through the Configuration bits.

The example code doesn't include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_WDT_ExistsWindowEnable](#) in your application to determine whether this feature is available.

Example

```
PLIB_WDT_WindowEnable ( WDT_ID_0 );  
PLIB_WDT_Enable ( WDT_ID_0 );
```

5.6.27.6.1.5 PLIB_WDT_TimerClear Function**C**

```
void PLIB_WDT_TimerClear(  
    WDT_MODULE_ID index  
);
```

Description

This function resets the WDT module. The WDT module should be cleared periodically before the count crosses and forces the

device to reset.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired WDT module

Returns

None.

Remarks

Resetting the device before the count reaches the window will cause a reset in Windowed mode.

The example code doesn't include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_WDT_ExistsTimerClear](#) in your application to determine whether this feature is available.

Example

```
PLIB_WDT_Enable ( WDT_ID_0 );  
  
//Application loop  
while(1)  
{  
    PLIB_WDT_TimerClear ( WDT_ID_0 );  
    //user code  
}
```

5.6.27.6.2 General Status Functions

5.6.27.6.2.1 PLIB_WDT_IsEnabled Function

C

```
bool PLIB_WDT_IsEnabled(  
    WDT_MODULE_ID index  
);
```

Description

Returns the 'true', if the watchdog timer is already ON. Otherwise returns 'false'.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired WDT module

Returns

true - If the watchdog timer is on false - If the watchdog timer is off

Remarks

This function returns 'true' if the device is enabled either through the Configuration bits or in the software.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

[PLIB_WDT_ExistsEnableControl](#) in your application to determine whether this feature is available.

Example

```
if ( PLIB_WDT_IsEnabled ( WDT_ID_0 ) )
{
    //Do some action
}
```

5.6.27.6.2.2 PLIB_WDT_PostscalarValueGet Function

C

```
char PLIB_WDT_PostscalarValueGet(
    WDT_MODULE_ID index
);
```

Description

This function returns the WDT postscalar value. The value will correspond to a division factor.

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired WDT module

Returns

The postscalar value.

Remarks

The value returned will be right-aligned.

Refer the datasheet of the specific device to get the division factor corresponding to the value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use [PLIB_WDT_ExistsPostscalarValue](#) in your application to determine whether this feature is available.

Example

```
uint8_t value;

PLIB_WDT_Enable ( WDT_ID_0 );
value = PLIB_WDT_PostscalarValueGet ( WDT_ID_0 );
```

5.6.27.6.3 Data Types and Constants

5.6.27.6.3.1 WDT_MODULE_ID Enumeration

C

```
typedef enum {
    WDT_ID,
    WDT_NUMBER_OF_MODULES
} WDT_MODULE_ID;
```

Description

WDT Module ID

This enumeration identifies the available WDT modules.

Members

Members	Description
WDT_ID	WDT Module 1 ID
WDT_NUMBER_OF_MODULES	Number of available WDT modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

5.6.27.6.4 Feature Existence Functions

5.6.27.6.4.1 PLIB_WDT_ExistsEnableControl Function

C

```
bool PLIB_WDT_ExistsEnableControl(  
    WDT_MODULE_ID index  
);
```

Description

This function identifies whether the EnableControl feature is available on the WDT module. When this function returns true, these functions are supported on the device:

- [PLIB_WDT_Enable](#)
- [PLIB_WDT_Disable](#)
- [PLIB_WDT_IsEnabled](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

Existence of the EnableControl feature:

- true - When EnableControl feature is supported on the device
- false - When EnableControl feature is not supported on the device

Remarks

None.

5.6.27.6.4.2 PLIB_WDT_ExistsPostscalarValue Function

C

```
bool PLIB_WDT_ExistsPostscalarValue(  
    WDT_MODULE_ID index  
);
```

Description

This function identifies whether the PostscalarValue feature is available on the WDT module. When this function returns true, this

function is supported on the device:

- [PLIB_WDT_PostscalarValueGet](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The PostscalarValue feature is supported on the device
- false - The PostscalarValue feature is not supported on the device

Remarks

None.

5.6.27.6.4.3 PLIB_WDT_ExistsTimerClear Function

C

```
bool PLIB_WDT_ExistsTimerClear(  
    WDT_MODULE_ID index  
);
```

Description

This function identifies whether the TimerClear feature is available on the WDT module. When this function returns true, this function is supported on the device:

- [PLIB_WDT_TimerClear](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The TimerClear feature is supported on the device
- false - The TimerClear feature is not supported on the device

Remarks

None.

5.6.27.6.4.4 PLIB_WDT_ExistsWindowEnable Function

C

```
bool PLIB_WDT_ExistsWindowEnable(  
    WDT_MODULE_ID index  
);
```

Description

This function identifies whether the WindowEnable feature is available on the WDT module. When this function returns true, these functions are supported on the device:

- [PLIB_WDT_WindowEnable](#)
- [PLIB_WDT_WindowDisable](#)

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Returns

- true - The WindowEnable feature is supported on the device
- false - The WindowEnable feature is not supported on the device

Remarks

None.

5.6.27.7 Files

Files

Name	Description
plib_wdt.h	Watchdog Timer (WDT) Peripheral Library interface header for Watchdog Timer common definitions.

Description

5.6.27.7.1 plib_wdt.h

WDT Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Watchdog Timer Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Watchdog Timer peripheral.

Functions

	Name	Description
≡	PLIB_WDT_Disable	Disables the WDT module.
≡	PLIB_WDT_Enable	Enables the WDT module.
≡	PLIB_WDT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the WDT module.
≡	PLIB_WDT_ExistsPostscalarValue	Identifies whether the PostscalarValue feature exists on the WDT module.
≡	PLIB_WDT_ExistsTimerClear	Identifies whether the TimerClear feature exists on the WDT module.
≡	PLIB_WDT_ExistsWindowEnable	Identifies whether the WindowEnable feature exists on the WDT module.
≡	PLIB_WDT_IsEnabled	Returns the watchdog timer on/off(enable/disable) status.
≡	PLIB_WDT_PostscalarValueGet	Returns the WDT postscalar value.
≡	PLIB_WDT_TimerClear	Resets the WDT module.
≡	PLIB_WDT_WindowDisable	Disables the WDT Windowed mode.
≡	PLIB_WDT_WindowEnable	Enables the WDT Window mode.

File Name

plib_wdt.h

Company

Microchip Technology Inc.

5.7 System Service Library Help

5.7.1 System Service Overview

System services provide common functionality that would otherwise need to be duplicated or would force other drivers and libraries to interact in complex and difficult to manage ways. System services have two primary types of implementations:

- System Service Libraries
- Low-Level Support

Most system service libraries follow the same basic model as a device driver (directly using a peripheral library to access hardware) or a middleware library (using a device driver to access hardware) as the rest of the system.

Some low-level support libraries provide common (cross-micro/cross-board) interfaces, but require system-specific implementation as part of a Board Support Package (BSP) or application configuration (for example, the low-level "raw" interrupt support necessary for interrupt-driven configurations).

5.7.1.1 Library Interface

Data Types and Constants

	Name	Description
	<u>_SYS_TASKS_PRIORITIES</u>	Defines system tasks priorities.
	<u>MAIN_RETURN_CODES</u>	Predefined list of return codes for "main".
	<u>SYS_MODULE_DATA</u>	Contains module-interface info used to define a table of system modules
	<u>SYS_MODULE_DEINITIALIZE_ROUTINE</u>	Pointer to a routine that de-initializes a system module (driver, library, or system-maintained application)
	<u>SYS_MODULE_INDEX</u>	Identifies which instance of a system module should be initialized or opened.
	<u>SYS_MODULE_INIT</u>	Initializes a module (including device drivers) in a current power status as requested by the system or power manager.
	<u>SYS_MODULE_INITIALIZE_ROUTINE</u>	Pointer to a routine that initializes a system module (driver, library, or system-maintained application).
	<u>SYS_MODULE_INTERFACE</u>	Structure contains pointers to module-interface routines for a single system module.
	<u>SYS_MODULE_OBJ</u>	Handle to an instance of a system module.
	<u>SYS_MODULE_REINITIALIZE_ROUTINE</u>	Pointer to a routine that reinitializes a system module (driver, library, or system-maintained application)
	<u>SYS_MODULE_STATUS_ROUTINE</u>	Pointer to a routine that gets the current status of a system module (driver, library, or system-maintained application).
	<u>SYS_MODULE_TASKS_DATA</u>	Structure contains pointers to module's tasks routine.

	<u>SYS_MODULE_TASKS_ROUTINE</u>	Pointer to a routine that performs the tasks necessary to maintain a state machine in a module system module (driver, library, or system-maintained application).
	<u>SYS_STATUS</u>	Identifies the current status/state of a system module (including device drivers).
	<u>SYS_TASKS_PRIORITY</u>	Defines system tasks priorities.
	<u>MAIN_RETURN</u>	Defines the correct return type for the "main" routine.
	<u>MAIN_RETURN_CODE</u>	Provides the correct value for the return code from "main".
	<u>SYS_MODULE_OBJ_INVALID</u>	Object handle value returned if unable to initialize the requested instance of a system module.
	<u>SYS_MODULE_OBJ_STATIC</u>	Object handle value returned by static modules.
	<u>SYS_MODULE_POWER_IDLE_RUN</u>	Module power-state idle-run state code.
	<u>SYS_MODULE_POWER_IDLE_STOP</u>	Module power-state idle-stop state code.
	<u>SYS_MODULE_POWER_OFF</u>	Module power-state power off state code.
	<u>SYS_MODULE_POWER_RUN_FULL</u>	Module power-state run-full state code.
	<u>SYS_MODULE_POWER_SLEEP</u>	Module power-state sleep state code.

System Functions

	Name	Description
	<u>SYS_Tasks</u>	Function that performs all polled system tasks.

Description

5.7.1.1.1 System Functions

5.7.1.1.1.1 SYS_Tasks Function

C

```
void SYS_Tasks();
```

Description

System Tasks Function

This function performs all polled system tasks by calling the state machine "tasks" functions for all polled modules in the system, including drivers, services, middleware and applications.

Preconditions

The [SYS_Initialize](#) function must have been called and completed.

Returns

None.

Remarks

If the module is interrupt driven, the system will call this routine from an interrupt context.

Example

```
SYS_Initialize ( NULL );

while ( true )
{
    SYS_Tasks ( );
}
```

```
}
```

5.7.1.1.2 Data Types and Constants

5.7.1.1.2.1 MAIN_RETURN_CODES Enumeration

C

```
typedef enum {
    MAIN_RETURN_FAILURE = -1,
    MAIN_RETURN_SUCCESS = 0
} MAIN_RETURN_CODES;
```

Description

Main Routine Codes

This enumeration provides a predefined list of return codes for "main".

Remarks

These codes can be passed into the [MAIN_RETURN_CODE](#) macro to convert them to the appropriate type (or discard them if not needed) for the Microchip microcontroller in use.

Example

```
MAIN_RETURN main ( void )
{
    // Initialize the system
    SYS_Initialize(...);

    // Main Loop
    while(true)
    {
        SYS_Tasks();
    }

    return MAIN_RETURN_CODE(MAIN_RETURN_SUCCESS);
}
```

5.7.1.1.2.2 SYS_MODULE_DATA Structure

C

```
typedef struct {
    SYS_MODULE_INTERFACE interface;
    SYS_MODULE_OBJ object;
    SYS_TASKS_PRIORITY priority;
    unsigned int count;
} SYS_MODULE_DATA;
```

Description

System Module Data

This structure contains module-interface info used to define a table of system modules.

Members

Members	Description
SYS_MODULE_INTERFACE interface;	Module interface (function pointers, index, etc)
SYS_MODULE_OBJ object;	Object handle obtained from initialization call
SYS_TASKS_PRIORITY priority;	Requested module priority (identifies "Tasks" interval)
unsigned int count;	Number of intervals (of the module priority) before calling this module's "Tasks" routine.

Remarks

Used internally by the system services. Not to be accessed by user code.

5.7.1.1.2.3 SYS_MODULE_DEINITIALIZE_ROUTINE Type**C**

```
typedef void (* SYS_MODULE_DEINITIALIZE_ROUTINE)(SYS_MODULE_OBJ object);
```

Description

System Module De-initialization Routine Pointer

This data type is a pointer to a routine that deinitializes a system module (driver, library, or system-maintained application).

Preconditions

The low-level board initialization must have (and will be) completed and the module's initialization routine will have been called before the system will call the deinitialization routine for any modules.

Parameters

Parameters	Description
object	Handle to the module instance

Returns

None.

Remarks

If the module instance has to be used again, the module's "initialize" function must first be called.

Example

TBD

5.7.1.1.2.4 SYS_MODULE_INDEX Type**C**

```
typedef unsigned short int SYS_MODULE_INDEX;
```

Description

System Module Index

This data type identifies to which instance of a system module a call to that module's "Initialize" and "Open" routines refers.

Remarks

Each individual module will usually define macro names for the index values it supports (ex. [DRV_TMR_INDEX_1](#), [DRV_TMR_INDEX_2](#), ...).

5.7.1.1.2.5 SYS_MODULE_INIT Union**C**

```
typedef union {  
    uint8_t value;  
    struct {  
        uint8_t powerState : 4;  
        uint8_t reserved : 4;  
    } sys;  
} SYS_MODULE_INIT;
```

Description

System Module Init

This structure provides the necessary data to initialize or reinitialize a module (including device drivers) into a requested power state. The structure can be extended in a module specific way as to carry module specific initialization data.

Members

Members	Description
uint8_t powerState : 4;	Requested power state
uint8_t reserved : 4;	Module-definable field, module-specific usage

Remarks

This structure is used in the device driver routines DRV__Initialize and DRV__Reinitialize that are defined by each device driver.

The "powerState" member has several pre-defined values (shown below). All other available values (within the 4-bit field) are available for module-specific meaning.

Pre-defined powerState Values:

0. [SYS_MODULE_POWER_OFF](#) - Module power-state power off state code
1. [SYS_MODULE_POWER_SLEEP](#) - Module power-state sleep state code
2. [SYS_MODULE_POWER_IDLE_STOP](#) - Module power-state idle-stop state code
3. [SYS_MODULE_POWER_IDLE_RUN](#) - Module power-state idle-run state code
- 4-14. - Module-specific meaning
15. [SYS_MODULE_POWER_RUN_FULL](#) - Module power-state run-full state code

5.7.1.1.2.6 SYS_MODULE_INITIALIZE_ROUTINE Type

C

```
typedef SYS_MODULE_OBJ (* SYS_MODULE_INITIALIZE_ROUTINE)(const SYS_MODULE_INDEX index, const  
SYS_MODULE_INIT * const init);
```

Description

System Module Initialization Routine Pointer

This data type is a pointer to a routine that initializes a system module (driver, library, or system-maintained application).

Preconditions

The low-level board initialization must have (and will be) completed before the system will call the initialization routine for any modules.

Parameters

Parameters	Description
index	Identifier for the module instance to be initialized

init	Pointer to the data structure containing any data necessary to initialize the module. This pointer may be null if no data is required and default initialization is to be used.

Returns

A handle to the instance of the system module that was initialized. This handle is a necessary parameter to all of the other system-module level routines for that module.

Remarks

This function will only be called once during system initialization.

5.7.1.1.2.7 SYS_MODULE_INTERFACE Structure

C

```
typedef struct {
    SYS_MODULE_INITIALIZE_ROUTINE pModuleInitialize;
    SYS_MODULE_REINITIALIZE_ROUTINE pModuleReinitialize;
    SYS_MODULE_DEINITIALIZE_ROUTINE pModuleDeinitialize;
    SYS_MODULE_STATUS_ROUTINE pModuleStatus;
    SYS_MODULE_TASKS_DATA * pTasksData;
    SYS_MODULE_INIT * pModuleInitData;
    SYS_MODULE_INDEX index;
    unsigned int tasksNum;
} SYS_MODULE_INTERFACE;
```

Description

System Module Interface

This structure contains pointers to module-interface routines for a single system module. The system will initialize these modules (using their respective initialization routines) and (potentially) monitor and maintain their states using the reinitialize, deinitialize, and status routines.

Members

Members	Description
SYS_MODULE_INITIALIZE_ROUTINE pModuleInitialize;	Pointer to the module's initialization routine
SYS_MODULE_REINITIALIZE_ROUTINE pModuleReinitialize;	Pointer to the module's re-initializtion routine
SYS_MODULE_DEINITIALIZE_ROUTINE pModuleDeinitialize;	Pointer to the module's de-initialization routine
SYS_MODULE_STATUS_ROUTINE pModuleStatus;	Pointer to the module's status routine
SYS_MODULE_TASKS_DATA * pTasksData;	Pointer to the module's "Tasks" routine
SYS_MODULE_INIT * pModuleInitData;	Pointer to the module's init data table
SYS_MODULE_INDEX index;	Index to the module instance
unsigned int tasksNum;	Number of tasks supported by this module

Remarks

A pointer to this structure is passed to the SYS_ModuleRegister routine.

Example

TBD

5.7.1.1.2.8 SYS_MODULE_OBJ Type

C

```
typedef uintptr_t SYS_MODULE_OBJ;
```

Description

System Module Object

This data type is a handle to a specific instance of a system module (such as a device driver).

Remarks

Code outside of a specific module should consider this as an opaque type (much like a void *). Do not make any assumptions about base type as it may change in the future or about the value stored in a variable of this type.

5.7.1.1.2.9 SYS_MODULE_REINITIALIZE_ROUTINE Type

C

```
typedef void (* SYS_MODULE_REINITIALIZE_ROUTINE)(SYS_MODULE_OBJ object, const SYS_MODULE_INIT  
* const init);
```

Description

System Module Re-initialization Routine Pointer

This data type is a pointer to a routine that re-initializes a system module (driver, library, or system-maintained application).

Preconditions

The low-level board initialization must have (and will be) completed and the module's initialization routine will have been called before the system will call the reinitialization routine for any modules.

Parameters

Parameters	Description
object	Handle to the module instance
init	Pointer to the data structure containing any data necessary to initialize the module. This pointer may be null if no data is required and default initialization is to be used.

Returns

None.

Remarks

This operation uses the same initialization data structure as the Initialize operation.

This operation can be used to change the power state of a module.

This operation can also be used to refresh the hardware state as defined by the initialization data, thus it must guarantee that all hardware state has been refreshed.

This function can be called multiple times to reinitialize the module.

Example

TBD

5.7.1.1.2.10 SYS_MODULE_STATUS_ROUTINE Type

C

```
typedef SYS_STATUS (* SYS_MODULE_STATUS_ROUTINE)(SYS_MODULE_OBJ object);
```

Description

System Module Status Routine Pointer

This data type is a pointer to a routine that gets the current status of a system module (driver, library, or system-maintained application).

Preconditions

The low-level board initialization must have (and will be) completed and the module's initialization routine will have been called before the system will call the status routine for any modules.

Parameters

Parameters	Description
object	Handle to the module instance

Returns

One of the possible status codes from [SYS_STATUS](#)

Remarks

A module's status operation can be used to determine when any of the other module level operations has completed as well as to obtain general status of the module. The value returned by the status routine will be checked after calling any of the module operations to find out when they have completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). A module may define module-specific error values of less or equal SYS_STATUS_ERROR_EXTENDED (-10).

The status function must NEVER block.

If the status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Example

TBD

5.7.1.1.2.11 SYS_MODULE_TASKS_DATA Structure

C

```
typedef struct {  
    SYS_MODULE_TASKS_ROUTINE pTasksFunction;  
    bool intModule;  
} SYS_MODULE_TASKS_DATA;
```

Description

System Module Tasks data

This structure contains pointers to module's tasks routine. A module can have an array of such structures depending on number of tasks routines it supports. The system will either directly/indirectly call the tasks routine depending on the value of intModule.

Members

Members	Description
SYS_MODULE_TASKS_ROUTINE pTasksFunction;	Pointer to the module's "Tasks" routine
bool intModule;	Tasks type (polled/interrupt)

Remarks

A pointer to this structure is passed to the SYS_ModuleRegister routine.

Example

TBD

5.7.1.1.2.12 SYS_MODULE_TASKS_ROUTINE Type**C**

```
typedef void (* SYS_MODULE_TASKS_ROUTINE)(SYS_MODULE_OBJ object);
```

Description

System Module Tasks Routine Pointer

This data type is a pointer to a routine that performs the tasks necessary to maintain a state machine in a module system module (driver, library, or system-maintained application).

Preconditions

The low-level board initialization must have (and will be) completed and the module's initialization routine will have been called before the system will call the deinitialization routine for any modules.

Parameters

Parameters	Description
object	Handle to the module instance

Returns

None.

Remarks

If the module is interrupt driven, the system will call this routine from an interrupt context.

Example

TBD

5.7.1.1.2.13 SYS_STATUS Enumeration**C**

```
typedef enum {  
    SYS_STATUS_ERROR_EXTENDED = -10,  
    SYS_STATUS_ERROR = -1,  
    SYS_STATUS_UNINITIALIZED = 0,  
    SYS_STATUS_BUSY = 1,  
    SYS_STATUS_READY = 2,  
    SYS_STATUS_READY_EXTENDED = 10  
} SYS_STATUS;
```

Description

System Module Status

This enumeration identifies the current status/state of a system module (including device drivers).

Members

Members	Description
SYS_STATUS_ERROR_EXTENDED = -10	Indicates that a non-system defined error has occurred. The caller must call the extended status routine for the module in question to identify the error.
SYS_STATUS_ERROR = -1	An un-specified error has occurred.
SYS_STATUS_UNINITIALIZED = 0	The module has not yet been initialized
SYS_STATUS_BUSY = 1	An operation is currently in progress
SYS_STATUS_READY = 2	Any previous operations have succeeded and the module is ready for additional operations
SYS_STATUS_READY_EXTENDED = 10	Indicates that the module is in a non-system defined ready/run state. The caller must call the extended status routine for the module in question to identify the state.

Remarks

This enumeration is the return type for the system-level status routine defined by each device driver or system module (for example, DRV_I2C_Status).

5.7.1.1.2.14 SYS_TASKS_PRIORITY Enumeration**C**

```
typedef enum SYS_TASKS_PRIORITIES {
    SYS_TASKS_PRIORITY_INVALID = 0,
    SYS_TASKS_PRIORITY_HIGH,
    SYS_TASKS_PRIORITY_MEDIUM,
    SYS_TASKS_PRIORITY_LOW
} SYS_TASKS_PRIORITY;
```

Description

System Tasks Priority

This enumeration defines the available system tasks priorities.

Members

Members	Description
SYS_TASKS_PRIORITY_INVALID = 0	Invalid priority (can be used as a sentinel value)
SYS_TASKS_PRIORITY_HIGH	High priority tasks are called every time through the loop
SYS_TASKS_PRIORITY_MEDIUM	Called at the medium priority interval.
SYS_TASKS_PRIORITY_LOW	Called at the low priority interval.

Remarks

To use medium priority tasks, a medium priority interval must be defined by defining SYS_TASKS_CONFIG_MEDIUM_INTERVAL to the desired interval (in milliseconds).

To use low priority tasks, a low priority interval must be defined by defining SYS_TASKS_CONFIG_LOW_INTERVAL to the desired interval (in milliseconds).

5.7.1.1.2.15 MAIN_RETURN Macro**C**

```
#define MAIN_RETURN void
```

Description

Main Routine Return Type

This macro defines the correct return type for the "main" routine for the selected Microchip microcontroller family.

Remarks

This type changes depending upon which family of Microchip microcontrollers is chosen. Most Microchip microcontrollers do not return any value from "main".

Example

```
MAIN_RETURN main ( void )
{
    // Initialize the system
    SYS_Initialize(...);

    // Main Loop
    while(true)
    {
        SYS_Tasks();
    }

    return MAIN_RETURN_CODE(MAIN_RETURN_SUCCESS);
}
```

5.7.1.1.2.16 MAIN_RETURN_CODE Macro

C

```
#define MAIN_RETURN_CODE(c)
```

Description

Main Routine Code Macro

This macro provides the correct value for the return code from "main".

Remarks

Most Microchip microcontrollers do not provide a return value from "main". Therefore, this macro discards the code it is given unless it is needed.

Example

```
MAIN_RETURN main ( void )
{
    // Initialize the system
    SYS_Initialize(...);

    // Main Loop
    while(true)
    {
        SYS_Tasks();
    }

    return MAIN_RETURN_CODE(MAIN_RETURN_SUCCESS);
}
```

5.7.1.1.2.17 SYS_MODULE_OBJ_INVALID Macro

C

```
#define SYS_MODULE_OBJ_INVALID ((SYS_MODULE_OBJ) -1 )
```

Description

System Module Object Invalid

This is the object handle value returned if unable to initialize the requested instance of a system module.

Remarks

Do not rely on the actual value of this constant. It may change in future implementations.

5.7.1.1.2.18 SYS_MODULE_OBJ_STATIC Macro**C**

```
#define SYS_MODULE_OBJ_STATIC ((SYS_MODULE_OBJ) 0 )
```

Description

System Module Object Static

This is the object handle value returned by static system modules.

Remarks

Do not rely on the actual value of this constant. It may change in future implementations.

5.7.1.1.2.19 SYS_MODULE_POWER_IDLE_RUN Macro**C**

```
#define SYS_MODULE_POWER_IDLE_RUN 3
```

Description

System Module Power Idle-Run State

This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter an idle-run state.

Remarks

An idle-run state indicates that the core CPU clock may be stopped, but the module's peripheral clock may continue running and peripheral operations may continue as long as no code needs to be executed. If code needs to execute, the module must cause an interrupt.

This value is passed in the powerState field of the [SYS_MODULE_INIT](#) structure that takes part in all modules initialization and reinitialization.

The power state codes between [SYS_MODULE_POWER_IDLE_RUN](#) (with a value of 3) and [SYS_MODULE_POWER_RUN_FULL](#) (with a value of 15) are available for module-specific definition and usage.

5.7.1.1.2.20 SYS_MODULE_POWER_IDLE_STOP Macro**C**

```
#define SYS_MODULE_POWER_IDLE_STOP 2
```

Description

System Module Power Idle-Stop State

This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter an idle-stop state.

Remarks

An idle-stop state indicates that the core CPU clock may be stopped, but the module's peripheral clock may continue running. However, the peripheral should prepare to stop operations when the idle state is entered.

This value is passed in the powerState field of the [SYS_MODULE_INIT](#) structure that takes part in all modules initialization and reinitialization.

The power state codes between [SYS_MODULE_POWER_IDLE_RUN](#) (with a value of 3) and [SYS_MODULE_POWER_RUN_FULL](#) (with a value of 15) are available for module-specific definition and usage.

5.7.1.1.2.21 SYS_MODULE_POWER_OFF Macro

C

```
#define SYS_MODULE_POWER_OFF 0
```

Description

System Module Power Off State

This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter a full power-off state.

Remarks

A power off state indicates that power may be completely removed (0 Volts).

This value is passed in the powerState field of the [SYS_MODULE_INIT](#) structure that takes part in all modules initialization and reinitialization.

The power state codes between [SYS_MODULE_POWER_IDLE_RUN](#) (with a value of 3) and [SYS_MODULE_POWER_RUN_FULL](#) (with a value of 15) are available for module-specific definition and usage.

5.7.1.1.2.22 SYS_MODULE_POWER_RUN_FULL Macro

C

```
#define SYS_MODULE_POWER_RUN_FULL 15
```

Description

System Module Power Run-Full State

This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter an run-full state.

Remarks

An run-full state indicates that the core CPU and peripheral clocks are operational at their normal configured speed and the module should be ready for normal operation.

This value is passed in the powerState field of the [SYS_MODULE_INIT](#) structure that takes part in all modules initialization and reinitialization.

The power state codes between [SYS_MODULE_POWER_IDLE_RUN](#) (with a value of 3) and [SYS_MODULE_POWER_RUN_FULL](#) (with a value of 15) are available for module-specific definition and usage.

5.7.1.1.2.23 SYS_MODULE_POWER_SLEEP Macro

C

```
#define SYS_MODULE_POWER_SLEEP 1
```

Description

System Module Power Sleep State

This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter a sleep state.

Remarks

A Sleep state indicates that the core CPU and peripheral clocks may be stopped and no code will execute and any module hardware will be stopped.

This value is passed in the powerState field of the `SYS_MODULE_INIT` structure that takes part in all modules initialization and reinitialization.

The power state codes between `SYS_MODULE_POWER_IDLE_RUN` (with a value of 3) and `SYS_MODULE_POWER_RUN_FULL` (with a value of 15) are available for module-specific definition and usage.

5.7.1.2 Files

Files

Name	Description
<code>sys_common.h</code>	Common definitions and declarations required for the system.
<code>sys_module.h</code>	Defines definitions and declarations related to system modules.
<code>system.h</code>	Aggregates all of the system services library interface headers.

Description

5.7.1.2.1 sys_common.h

System Services Common Library Header

This file defines the common definitions and declarations required for the system.

Remarks

This file is included by "`system.h`".

Enumerations

	Name	Description
	<code>_SYS_TASKS_PRIORITIES</code>	Defines system tasks priorities.
	<code>MAIN_RETURN_CODES</code>	Predefined list of return codes for "main".
	<code>SYS_TASKS_PRIORITY</code>	Defines system tasks priorities.

Macros

	Name	Description
	<code>MAIN_RETURN</code>	Defines the correct return type for the "main" routine.
	<code>MAIN_RETURN_CODE</code>	Provides the correct value for the return code from "main".

File Name

`sys_common.h`

Company

Microchip Technology Inc.

5.7.1.2.2 sys_module.h

System Module Header

This file defines definitions and interfaces related to system modules.

Remarks

This file is included via "[system.h](#)" and does not normally need to be included directly.

Enumerations

	Name	Description
	SYS_STATUS	Identifies the current status/state of a system module (including device drivers).

Functions

	Name	Description
	SYS_Initialize	Function that initializes all modules in the system.
	SYS_Tasks	Function that performs all polled system tasks.

Macros

	Name	Description
	SYS_MODULE_OBJ_INVALID	Object handle value returned if unable to initialize the requested instance of a system module.
	SYS_MODULE_OBJ_STATIC	Object handle value returned by static modules.
	SYS_MODULE_POWER_IDLE_RUN	Module power-state idle-run state code.
	SYS_MODULE_POWER_IDLE_STOP	Module power-state idle-stop state code.
	SYS_MODULE_POWER_OFF	Module power-state power off state code.
	SYS_MODULE_POWER_RUN_FULL	Module power-state run-full state code.
	SYS_MODULE_POWER_SLEEP	Module power-state sleep state code.

Structures

	Name	Description
	SYS_MODULE_DATA	Contains module-interface info used to define a table of system modules
	SYS_MODULE_INTERFACE	Structure contains pointers to module-interface routines for a single system module.
	SYS_MODULE_TASKS_DATA	Structure contains pointers to module's tasks routine.

Types

	Name	Description
	SYS_MODULE_DEINITIALIZE_ROUTINE	Pointer to a routine that de-initializes a system module (driver, library, or system-maintained application)
	SYS_MODULE_INDEX	Identifies which instance of a system module should be initialized or opened.
	SYS_MODULE_INITIALIZE_ROUTINE	Pointer to a routine that initializes a system module (driver, library, or system-maintained application).
	SYS_MODULE_OBJ	Handle to an instance of a system module.
	SYS_MODULE_REINITIALIZE_ROUTINE	Pointer to a routine that reinitializes a system module (driver, library, or system-maintained application)
	SYS_MODULE_STATUS_ROUTINE	Pointer to a routine that gets the current status of a system module (driver, library, or system-maintained application).

	<u>SYS_MODULE_TASKS_ROUTINE</u>	Pointer to a routine that performs the tasks necessary to maintain a state machine in a module system module (driver, library, or system-maintained application).

Unions

	Name	Description
	<u>SYS_MODULE_INIT</u>	Initializes a module (including device drivers) in a current power status as requested by the system or power manager.

File Name

sys_module.h

Company

Microchip Technology Inc.

5.7.1.2.3 system.h

System Services Library Interface Header

This file aggregates all of the system services library interface headers so that client code only needs to include this one single header to obtain prototypes and definitions for the interface to all system service libraries. System services provide common functionality that would otherwise need to be duplicated by multiple other modules or would force other drivers and libraries to interact in complex and hard to manage ways. System services eliminate conflicts by controlling access shared resources.

Remarks

The directory in which this file resides should be added to the compiler's search path for header files. Many Harmony libraries assume that "system.h" is available in the "include" path.

File Name

system.h

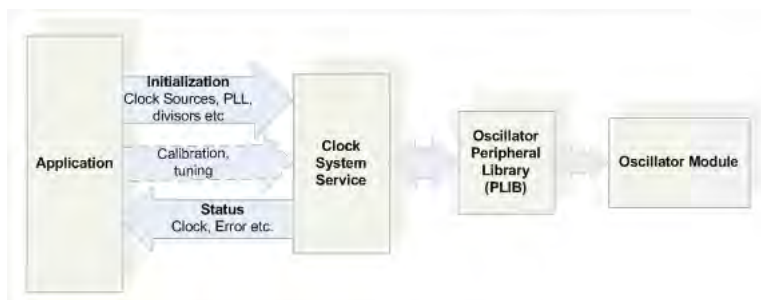
Company

Microchip Technology Inc.

5.7.2 Clock System Service Library

5.7.2.1 Introduction

Clock System Service Library
for
Microchip Microcontrollers



This library provides an interface to manage the Oscillator module on Microchip family of micro controllers during different modes of operation.

Overview of Oscillator

The Oscillator is the heart of the microcontroller which provides the clock on which the core and the peripherals run. For all of the oscillators present on a Microchip microcontroller, two kinds of configurations exist:

- Through Configuration bits
- At Execution time

The first is through 'Configuration bits' which is a one-time configuration and it is done during the programming of the device. These one-time configurations are programmed in the code memory. The other is 'Execution time' configuration, which deals with features that are allowed to change during the code execution. This library provides functions which deals only with the 'Execution time' configurable features of the Oscillator module.

Oscillator is a device which supports the CPU and other peripherals by providing clock. So there is more of initialization and a little or no run time operations. The run time operations would be getting the System clock or any other 'output clocks'. If the application want to change any of the initialization, at run time, that can be done.

The Oscillator module as a whole is a group of oscillators. These are provided for the application to choose right oscillator for his application. It is important to wisely choose the oscillator and the frequency at which the CPU and the other peripheral should run at in a very power critical environments.

5.7.2.2 Release Notes

MPLAB Harmony Version: v0.70b **Oscillator System Service Library Version :** 0.01a **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.7.2.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED AS IS MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.7.2.4 Using the Library

This topic describes the basic architecture of the Oscillator System Service and provides information and examples on how to use it.

Interface Header File: sys_osc.h

The interface to the Oscillator System Service is defined in the "sys_osc.h" header file, which is included by the "sys.h" header file.

Any C language source (.c) file that uses the Oscillator system service must include "sys.h".

5.7.2.4.1 Abstraction Model

To understand the oscillator module and how each of its feature is mapped in this library, it is important to understand all these terminologies.

Oscillators:

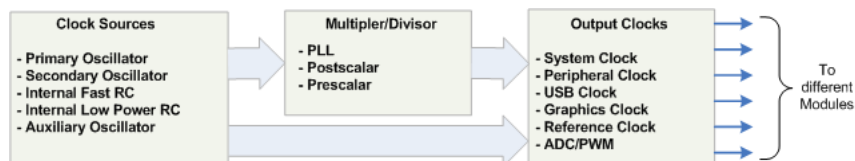
A clock source is hardware which generates oscillations. This may be internal or external.

Divisor and Multiplier/PLL:

These are hardware modules which can scale the clock. The rate at which the scaling is done may be fixed or configurable.

Clocks:

Clock outputs means output lines from the oscillator module which may route to different modules of the device or to the CPU(the system clock).



The diagram above gives a simplified explanation and the relationship between the above mentioned terms. In most of the cases, there are multiple clock source options available for each of the clock outputs. But not all the clock sources are available for all the output clocks. Scaling is an optional feature in most of the cases.

5.7.2.4.2 Library Overview

The diagram below shows the flow of an application call to Oscillator System service. Refer to the section [System Services Overview](#) for how the system services operates in a system.

Configuration
- Configuration bit divisors - Configuration bit multipliers - Primary oscillator clock input - Secondary oscillator clock input
Initialization
- General - Action on Wait instruction - Enable Secondary - Clock Sources - System Clock - USB Clock - Graphics Clock - Reference clock - PLL and Clock divisors - Doze Mode
Oscillator Tuning
- Internal FRC oscillator Tuning
Status routines
- Get clock - Clock Fail Status

The library interface routines are divided into various subsections, each of the subsection addresses one of the blocks or the overall operation of the Oscillator System Service module.

Library Interface Section	Description
System Initialization Functions	This section provides an initialization routine to configure the module at the startup of the system and Re-initialization if the application want to change any of the initialization settings
System Status Functions	Provides function to read different clock outputs from the oscillator module.
Optional System Functions	FRC oscillator tuning and clock fail detection APIs

5.7.2.4.3 How the Library Works

5.7.2.4.3.1 Initialization

Oscillator is a module with a little or no run time operations. So the application must do most of the configurations at the time of System initialization even though it is allowed to change the clock frequency at run time.

Not all the devices will need all the initialization parameters of the structure. Refer the datasheet of the specific device to know the features available in each device.

Call the initialization API by passing the initialization data required.

```

generalInit.auxOscMode = CLK_AUX_HS;
generalInit.auxPllEnable = true;
generalInit.refOscEnable = true;
generalInit.trimValue = 0;
generalInit.secondaryOscKeepEnabled = true;
generalInit.onWaitOperation = CLK_ON_WAIT_IDLE;

generalInit.sysClockSource = SYS_CLK_SYSCLOCK_PRIMARY_PLL;
generalInit.usbClockSource = SYS_CLK_USBCLK_PRIMARY_PLL;
generalInit.auxClockSource = CLK_AUX_CLOCK_PRIMARY_PLL;
generalInit.refOscBaseClock = CLK_REF_PB_CLOCK;
generalInit.pll_multiplier = SYS_CLK_PLL_SYSCLOCK_BY_20;
generalInit.frcClockDivisor = CLK_FRC_DIVISOR_4;
generalInit.peripheralBusClkDiv = CLK_PB_CLOCK_DIV_8;
generalInit.auxPllMultiplier = CLK_PLL_MUL_18;
generalInit.refOscDivisor = 1024;

```

```
generalInit.dozeRatio = CLK_DOZE_RATIO_4;
generalInit.dozeRecoverOnInterrupt = true;
```

```
SYS_CLK_Initialize (&generalInit);
```

Oscillator needs some time to settle. So it is recommended to do the oscillator initialization before initializing other modules. Check for the oscillator status once all the initializations are complete and before application task routine takes control.

5.7.2.4.3.2 Changing the Clock

It is allowed to change any of the clock frequency at run time. There is a single API provided to set the clock source and configure it to the clock frequency requested. User is supposed to look at the data sheet and find out one of the possible values of the output clock. The API, internally, adjusts the multipliers/divisors necessary.

Setting the System Clock:

```
SYS_CLK_ClockFrequencySet(CLK_SYSTEM, SYS_CLK_SOURCE_FRC, 8000000);
```

Setting the USB Clock:

```
SYS_CLK_ClockFrequencySet(CLK_USB, SYS_CLK_SOURCE_FRCPLL, 48000000);
```

5.7.2.4.3.3 Using Status Functions

Once the initialization is done, oscillator needs some time to settle down and get into its normal operation. The application can add a delay, do other initialization or wait in the status check routine. To minimize the waiting time it is highly recommended to do the initialization before all other initializations.

```
// Populate the oscillator initialization structure
// Call the initialization routines

// Do other initializations

// Check whether all the oscillators are ready
while((!SYS_OSC_IsReady(OSC_OUTPUT_SYSCLOCK)) &&
      (!SYS_OSC_IsReady(OSC_OUTPUT_PBCLOCK)) &&
      (!SYS_OSC_IsReady(OSC_OUTPUT_USBCLOCK)) &&
      (!SYS_OSC_IsReady(OSC_OUTPUT_GFXCLOCK)));

while(1)
{
    // Application
}
```

The application can read the frequency at which the system is running at any time after the initialization is complete.

```
uint32_t sysClockOutputHz, usbClockOutputHz, pbClockOutputHz, gfxClockOutputHz;
```

```
// Get the system clock
sysClockOutputHz = SYS_OSC_ClockFrequencyGet(OSC_OUTPUT_SYSCLOCK);
// Get the peripheral clock
pbClockOutputHz = SYS_OSC_ClockFrequencyGet(OSC_OUTPUT_PBCLOCK);
// Get the USB clock
usbClockOutputHz = SYS_OSC_ClockFrequencyGet(OSC_OUTPUT_USBCLOCK);
// Get the graphics clock
gfxClockOutputHz = SYS_OSC_ClockFrequencyGet(OSC_OUTPUT_GFXCLOCK);
```

5.7.2.4.3.4 Oscillator Tuning

Tuning the Oscillator:

Oscillator Tuning will help compensate for temperature effects on the FRC frequency over a wide range of temperatures. The tuning step size is an approximation, the application is supposed to try different values to achieve the best result. In some of the

devices there are different tuning modes available.

Direct Number method:

```
//Software must unlock before the start of the tuning.
SYS_OSC_FRCTUNING_DATA tuningInput;
OSC_FRC_TUNING_VALUE tuningValue = OSC_TUNE_TO_CENTRAL_FREQ_PLUS_0_86;

tuningInput.tuningMode = OSC_TUNING_USING_NUMBER;
tuningInput.tuningData = &tuningValue;
SYS_OSC_FRCTuningSet(&tuningInput);
```

Sequential Dithering:

To get the Sequential Dithering working, the application is suppose to set the value in 7 sequencers and also in the tuning register. Then configure the PWM module, set the period and pulse width. The Oscillator module generates frequencies corresponding the value specified in these registers in every 8th PWM cycle.

```
//Software must unlock before the start of the tuning.
SYS_OSC_FRCTUNING_DATA tuningInput;
OSC_FRC_TUNING_VALUE tuningValue[8];

//Initialize with tuning values
tuningValue[0] = OSC_TUNE_TO_CENTRAL_MINUS_2_25_PERC;
tuningValue[1] = OSC_TUNE_TO_CENTRAL_MINUS_1_5_PERC;
tuningValue[2] = OSC_TUNE_TO_CENTRAL_MINUS_0_375_PERC;
tuningValue[3] = OSC_TUNE_TO_CENTRAL_PLUS_0_43_PERC;
tuningValue[4] = OSC_TUNE_TO_CENTRAL_PLUS_1_29_PERC;
tuningValue[5] = OSC_TUNE_TO_CENTRAL_PLUS_2_54_PERC;
tuningValue[6] = OSC_TUNE_TO_CENTRAL_MINUS_3_PERC;
tuningValue[7] = OSC_TUNE_TO_CENTRAL_MINUS_3_PERC;

tuningInput.tuningMode = OSC_TUNING_SEQ_DITHER;
tuningInput.tuningData = tuningValue;
SYS_OSC_FRCTuningSet(&tuningInput);
```

Pseudo-Random Number:

Select the tuning mode. Then configure the PWM module, set the period and pulse width. The Oscillator system generates a 4-bit number based on Pseudo-Random number generation algorithm. It then uses this value to tune the FRC oscillator. The module will generate different frequencies corresponding to the generated Pseudo-Random numbers every 8th PWM cycle.

```
//Software must unlock before the start of the tuning.
SYS_OSC_FRCTUNING_DATA tuningInput;
OSC_FRC_TUNING_VALUE tuningValue = 0x7FFF;

tuningInput.tuningMode = OSC_TUNING_PSEUDO_RANDOM;
tuningInput.tuningData = &tuningValue;
SYS_OSC_FRCTuningSet(&tuningInput);
```

5.7.2.5 Configuring the Library

The configuration of the Oscillator System Service Library is based on the file `sys_config.h`

This header file contains the configuration selection for the Oscillator System Service Library. Based on the selections made, the Oscillator System Service Library will support or not support selected features. These configuration settings will apply to all instances of the Oscillator System Service Library.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.7.2.6 Library Interface

Data Types and Constants

	Name	Description
	SYS_CLK_FRCTUNING_DATA	Defines the data required for tuning the FRC Oscillator.
	SYS_CLK_INIT	Defines the data required to initialize the Oscillator for the Clock System Service.
	SYS_CLK_OUTPUT	Lists the available clock outputs.
	SYS_CLK_SOURCE	Lists the available clock outputs.
	CLK_PERIPH_BUS_MZ	Lists the available peripheral clocks.
	CLK_SOURCE_PERIPHERAL	Lists the available clock sources for the peripheral clock.

Optional System Functions

	Name	Description
	SYS_CLK_FRCTuningSet	Starts the FRC tuning.

System Initialization Functions

	Name	Description
	SYS_CLK_ClockFailureCallbackRegister	Allows registration of a call back function that will be triggered on a clock failure.
	SYS_CLK_Initialize	Initializes hardware and internal data structure of the System Clock.

System Status Functions

	Name	Description
	SYS_CLK_ClockFrequencyGet	Gets the selected clock frequency in Hz.
	SYS_CLK_ClockFrequencySet	Configures the clock multiplier and divisor values.
	SYS_CLK_ClockIsReady	Checks whether the selected clock output is stable and ready to use.
	SYS_CLK_FRCTune	This function is used for direct value based FRC oscillator tuning.
	SYS_CLK_MZPeriphBusFreqGet	Gets the selected peripheral clock frequency in Hz (PIC32MZ only).

Description

5.7.2.6.1 System Initialization Functions

5.7.2.6.1.1 SYS_CLK_ClockFailureCallbackRegister Function

C

```
void SYS_CLK_ClockFailureCallbackRegister(
    void * callback
);
```

Description

This function allows registration of a callback function that will be triggered on a clock failure. In addition, this function monitors the Fail-Safe Clock Monitor (FSCM).

Preconditions

None.

Parameters

Parameters	Description
index	Identifies the desired System Clock

Returns

None.

Remarks

This function is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
SYS_CLK_ClockFailureCallbackRegister(&ErrorHandle);

void ErrorHandle (void)
{
    //Handle the error.
}
```

5.7.2.6.1.2 SYS_CLK_Initialize Function**C**

```
void SYS_CLK_Initialize(
    const SYS_CLK_INIT const * clkInit
);
```

Description

This function initializes the hardware and internal data structure of the System Clock.

Preconditions

None.

Parameters

Parameters	Description
clkInit	Pointer to a data structure containing any data necessary to initialize the System Clock. This pointer can be NULL if no data is required as static overrides have been provided.

Returns

None.

Example

```
SYS_CLK_INIT generalInit;

// Populate the oscillator initialization structure
generalInit.systemClockSource = SYS_CLK_SOURCE_FRC_SYSPLL;
generalInit.systemClockFrequency = 30000000;
generalInit.usbClockEnable = true;
generalInit.usbClockSource = SYS_CLK_SOURCE_PRIMARY_USBPLL;
generalInit.usbClockFrequency = 48000000;
generalInit.peripheralClockEnable = true;
generalInit.peripheralClockSource = SYS_CLK_SOURCE_SYSClk_OUT;
generalInit.peripheralClockFrequency = 15000000;

generalInit.referenceClockEnable = true;
generalInit.referenceClockSource = SYS_CLK_SOURCE_FRC;
generalInit.referenceClockFrequency = 8000000;

SYS_CLK_Initialize (&generalInit);
```

Remarks: This could be called at the time of system initialization to initialize the oscillator or after initialization to change any of the initialization settings.

5.7.2.6.2 System Status Functions

5.7.2.6.2.1 SYS_CLK_ClockFrequencyGet Function

C

```
unsigned long SYS_CLK_ClockFrequencyGet(  
    SYS_CLK_OUTPUT clockOutput  
);
```

Description

This function gets the selected clock frequency in Hz.

Preconditions

The selected clock output must be configured and enabled.

Returns

Clock frequency in Hz.

Example

```
unsigned long sysClockOutputHz;  
  
sysClockOutputHz = SYS_CLK_ClockFrequencyGet(CLK_SYSTEM);
```

5.7.2.6.2.2 SYS_CLK_ClockFrequencySet Function

C

```
unsigned long SYS_CLK_ClockFrequencySet(  
    SYS_CLK_OUTPUT outputClock,  
    SYS_CLK_SOURCE clockSource,  
    unsigned long clockFrequency  
);
```

Description

This function configures the clock multiplier and divisor values. This function covers all of the configuration for an oscillator clock output. All of the arguments in the API may not be applicable in all cases. The API discards the arguments that are not relevant.

Preconditions

The [SYS_CLK_Initialize](#) function should be called before calling this API.

Parameters

Parameters	Description
outputClock	Clock for which the source is to be selected
clockSource	Select the clock source
clockFrequency	Select the clock clock frequency

Returns

None.

Example

```
SYS_CLK_ClockFrequencySet(CLK_SYSTEM, SYS_CLK_SOURCE_FRC, 8000000);  
Remarks: This API returns a zero if it cannot achieve the value requested. The user
```

must recheck the hardware and enter the possible value.

5.7.2.6.2.3 SYS_CLK_ClockIsReady Function

C

```
bool SYS_CLK_ClockIsReady(  
    SYS_CLK_OUTPUT clockOutput  
);
```

Description

This function checks whether the selected clock output is stable. This should be checked at system start-up, after the initialization is complete and also each time when there is a change in the clock setup configuration. This function checks the PLL lock status to determine whether the PLL is enabled.

Preconditions

None.

Returns

- true - The selected clock output is stable
- false - The selected clock output is not stable

Remarks

This function should be called at system start-up.

This API is non-blocking in nature, so the application must call this API in a loop to wait until the initial settling time of the oscillator expires and the oscillator becomes ready.

Example

```
//Wait until the selected clock output is getting stabilized.  
while(!SYS_CLK_ClockIsReady(CLK_SYSTEM));
```

5.7.2.6.2.4 SYS_CLK_FRCTune Function

C

```
void SYS_CLK_FRCTune(  
    OSC_FRC_DIV tuningData  
);
```

Description

This function triggers the FRC tuning based on the selected mode. FRC tuning functionality has been provided to help customers compensate for temperature effects on the FRC frequency over a wide range of temperatures.

Preconditions

The device selected must support the oscillator tuning feature.

Parameters

Parameters	Description
tuningData	One of the possible value of OSC_FRC_TUNE enum

Returns

None.

Remarks

The tuning step size is an approximation, and is neither characterized, nor tested.

This API may can be only used with devices that support direct value based FRC tuning. Refer to the specific device data sheet

to determine whether this feature exists for your device.

Example

```
SYS_CLK_FRCTune( OSC_FRC_TUNE_MINUS_1_875_PERCENT);
```

5.7.2.6.2.5 SYS_CLK_MZPeriphBusFreqGet Function

C

```
unsigned long SYS_CLK_MZPeriphBusFreqGet(  
    CLK_PERIPH_BUS_MZ periphBus  
);
```

Description

This function gets the selected clock frequency in Hz (PIC32MZ only).

Preconditions

The selected clock output must be configured and enabled.

Returns

Clock frequency in Hz.

Example

```
unsigned long sysClockOutputHz;  
  
sysClockOutputHz = SYS_CLK_ClockFrequencyGet(CLK_PERIPH_BUS_2);
```

5.7.2.6.3 Optional System Functions

5.7.2.6.3.1 SYS_CLK_FRCTuningSet Function

C

```
void SYS_CLK_FRCTuningSet(  
    SYS_CLK_FRCTUNING_DATA * tuningData  
);
```

Description

This function triggers the FRC tuning based on the selected mode.

Preconditions

The device selected must support the oscillator tuning feature.

Parameters

Parameters	Description
tuningData	An instance of the structure SYS_CLK_FRCTUNING_DATA Data used for tuning
tuningMode	FRC oscillator tuning mode

Returns

None.

Remarks

This function is not applicable for devices that do not have an FRC oscillator and devices that do have an FRC oscillator, but not FRC tuning. Refer to the specific device data sheet to determine whether this feature exists for your device.

Example

```
SYS_CLK_FRCTUNING_DATA tuningInput;
```

```
CLK_FRC_TUNING_VALUE tuningValue;

tuningData.tuningMode = CLK_TUNING_USING_NUMBER;
tuningInput.tuningData = &tuningValue;

SYS_CLK_FRCTuningSet(&tuningInput);
```

5.7.2.6.4 Data Types and Constants

5.7.2.6.4.1 SYS_CLK_FRCTUNING_DATA Structure

C

```
typedef struct {
    OSC_TUNING_MODE tuningMode;
    uint8_t * tuningData;
} SYS_CLK_FRCTUNING_DATA;
```

Description

Clock module FRC tuning modes

This structure defines the necessary data required to tune the FRC Oscillator.

Members

Members	Description
OSC_TUNING_MODE tuningMode;	Oscillator tuning mode
uint8_t * tuningData;	The data required may vary depending on the mode. This could be pointed to a variable or an array of data

Remarks

None.

5.7.2.6.4.2 SYS_CLK_INIT Structure

C

```
typedef struct {
    SYS_CLK_SOURCE systemClockSource;
    unsigned long systemClockFrequency;
    bool usbClockEnable;
    SYS_CLK_SOURCE usbClockSource;
    unsigned long usbClockFrequency;
    bool peripheralClockEnable;
    SYS_CLK_SOURCE peripheralClockSource;
    unsigned long peripheralClockFrequency;
    bool referenceClockEnable;
    SYS_CLK_SOURCE referenceClockSource;
    unsigned long referenceClockFrequency;
    bool secondaryOscKeepEnabled;
    OSC_OPERATION_ON_WAIT onWaitInstruction;
    bool referenceStopInIdle;
    bool referenceSuspendInSleep;
} SYS_CLK_INIT;
```

Description

Clock System Service Reference Oscillator initialization data

This structure defines the data required to initialize the Oscillator for the Clock System Service.

Members

Members	Description
SYS_CLK_SOURCE systemClockSource;	Initializations for System clock
unsigned long systemClockFrequency;	Set the System clock frequency
bool usbClockEnable;	Initializations for USB clock Enable the USB clock. If the device doesn't need USB clock, set this variable to 'false'. In that case, user need not set next two variables.
SYS_CLK_SOURCE usbClockSource;	Select the USB clock source
unsigned long usbClockFrequency;	Set the USB clock frequency
bool peripheralClockEnable;	Initializations for Peripheral clock Enable the Peripheral clock. If the device doesn't need Peripheral clock, set this variable to 'false'. In that case, user need not set next two variables.
SYS_CLK_SOURCE peripheralClockSource;	Select the Peripheral clock source
unsigned long peripheralClockFrequency;	Set the Peripheral clock frequency
bool referenceClockEnable;	Initializations for Reference clock out Set this variable to 'true' to enable the reference clock out. The system service will discard this either if this variable is set to 'false' or if the device doesn't has 'reference clock out' feature.
SYS_CLK_SOURCE referenceClockSource;	Base clock for the reference clock out. This will be divided and tuned to get the actual Reference clock output
unsigned long referenceClockFrequency;	Use this to fine tune the divided reference oscillator clock out.
bool secondaryOscKeepEnabled;	Even though the secondary oscillator is not used, keeping the oscillator running, allows a fast switch to the lower system clock for low-power operation
OSC_OPERATION_ON_WAIT onWaitInstruction;	System action on a 'Wait' instruction
bool referenceStopInIdle;	Stop reference clock out in Idle mode
bool referenceSuspendInSleep;	Stop reference clock out in sleep mode

Remarks

Calling this API may not be essential in some cases. Some of the elements in this structure are part of the library configuration file. Setting them here will override the configuration file setting.

5.7.2.6.4.3 SYS_CLK_OUTPUT Enumeration**C**

```
typedef enum {
    CLK_SYSTEM,
    CLK_PERIPHERAL,
    CLK_USB,
    CLK_REFERENCE,
    CLK_GRAPHIX,
    CLK_ADCPWM
} SYS_CLK_OUTPUT;
```

Description

Clocks enumeration

This enumeration lists all of the available clock outputs. This is used by the [SYS_CLK_ClockFrequencyGet](#) and [SYS_CLK_ClockIsReady](#) functions.

Members

Members	Description
CLK_SYSTEM	Selects System clock output. This is used by the CPU and some of the peripherals

CLK_PERIPHERAL	Selects peripheral clock output. This is used by most of the peripherals
CLK_USB	Selects USB clock output. This is used by the USB module
CLK_REFERENCE	Selects referece clock output. This to check any of the clock or it could be used as a clock source for other devices
CLK_GRAPHIX	Selects Graphics clock output. This is used by the display module
CLK_ADCPWM	Selects ADC/PWM clock

Remarks

'Clock output' means output from the clock source.

5.7.2.6.4.4 SYS_CLK_SOURCE Enumeration**C**

```
typedef enum {
    SYS_CLK_TOTAL_SOURCES,
    SYS_CLK_SOURCE_EXTERNAL,
    SYS_CLK_SOURCE_USBPPLL_OUT,
    SYS_CLK_SOURCE_SYSPPLL_OUT,
    SYS_CLK_SOURCE_PBCLK_OUT,
    SYS_CLK_SOURCE_SYSCLK_OUT,
    SYS_CLK_SOURCE_AUXILIARY,
    SYS_CLK_SOURCE_FRC_BY_DIV,
    SYS_CLK_SOURCE_FRC_BY_16,
    SYS_CLK_SOURCE_LPRC,
    SYS_CLK_SOURCE_SECONDARY,
    SYS_CLK_SOURCE_PRIMARY_USBPPLL,
    SYS_CLK_SOURCE_PRIMARY_SYSPPLL,
    SYS_CLK_SOURCE_PRIMARY,
    SYS_CLK_SOURCE_FRC_SYSPPLL,
    SYS_CLK_SOURCE_FRC
} SYS_CLK_SOURCE;
```

Description

Clock clocks enumeration

This enumeration lists all the available clock outputs. This is used by the [SYS_CLK_ClockFrequencyGet](#) and [SYS_CLK_ClockIsReady](#) functions.

Members

Members	Description
SYS_CLK_TOTAL_SOURCES	This gives the total number of sources present
SYS_CLK_SOURCE_EXTERNAL	Source of clock is external(from the pin) This is used only for the Reference clock.
SYS_CLK_SOURCE_USBPPLL_OUT	Source of clock is output of USB PLL This is used only for the Reference clock.
SYS_CLK_SOURCE_SYSPPLL_OUT	Source of clock is the output of System PLL. This is used only for the Reference clock.
SYS_CLK_SOURCE_PBCLK_OUT	Source of clock is peripheral clock. This is used only for the Reference clock.
SYS_CLK_SOURCE_SYSCLK_OUT	Source of clock is System clock. This is used for Peripheral clock and the Reference clock.
SYS_CLK_SOURCE_AUXILIARY	Source of clock is Auxiliary clock.
SYS_CLK_SOURCE_FRC_BY_DIV	Source of clock is internal fast RC divided by the divisor configured in software. This is used only for the System clock.
SYS_CLK_SOURCE_FRC_BY_16	Source of clock is internal fast RC divided by 16. This is used only for the System clock.
SYS_CLK_SOURCE_LPRC	Source of clock is internal low power RC. This is used for System clock and the Reference clock.

SYS_CLK_SOURCE_SECONDARY	Source of clock is secondary oscillator. This is used for System clock and the Reference clock.
SYS_CLK_SOURCE_PRIMARY_USBPLL	Source of clock is primary oscillator multiplied by the USB PLL value and divided by the divisor configured by software. This is used only for the USB clock
SYS_CLK_SOURCE_PRIMARY_SYSPLL	Source of clock is primary oscillator multiplied by the System PLL value and divided by the divisor configured by software. This is used for System clock and the Reference clock.
SYS_CLK_SOURCE_PRIMARY	Source of clock is primary oscillator. This is used for System clock, Reference clock and the USB clock
SYS_CLK_SOURCE_FRC_SYSPLL	Source of clock is internal fast RC multiplied by system PLL. This is used for System clock and the Reference clock.
SYS_CLK_SOURCE_FRC	Source of clock is internal fast RC This is used for System clock, Reference clock and the USB clock

Remarks

'Clock output' means output from the clock source. Not all clock sources are available for all devices. Refer to the specific device data sheet to determine the available clock sources.

5.7.2.6.4.5 CLK_PERIPH_BUS_MZ Enumeration**C**

```
typedef enum {
    CLK_PERIPH_BUS_1,
    CLK_PERIPH_BUS_2,
    CLK_PERIPH_BUS_3,
    CLK_PERIPH_BUS_4,
    CLK_PERIPH_BUS_5,
    CLK_PERIPH_BUS_6,
    CLK_PERIPH_BUS_7,
    CLK_PERIPH_BUS_8
} CLK_PERIPH_BUS_MZ;
```

Description

Peripherals Clocks enumeration (PIC32MZ only)

This enumeration lists all of the available peripheral clocks. This is used by the [SYS_CLK_ClockFrequencyGet](#) function to return the peripheral clock frequency to the caller.

Members

Members	Description
CLK_PERIPH_BUS_6	not used

Remarks

None.

5.7.2.6.4.6 CLK_SOURCE_PERIPHERAL Enumeration**C**

```
typedef enum {
    SYS_CLK_SOURCE_SYSTEMCLK
} CLK_SOURCE_PERIPHERAL;
```

Description

Peripheral clock sources enumeration

This enumeration lists all the available peripheral clock sources. This is used by the [SYS_CLK_ClockFrequencySet](#) function.

Members

Members	Description
SYS_CLK_SOURCE_SYSTEMCLK	Source of clock is system clock

Remarks

None.

5.7.2.7 Files

Files

Name	Description
sys_clk.h	API for the Clock System Service.

Description

5.7.2.7.1 sys_clk.h

Clock System Service Interface Definition

This file contains the interface definition for the Clock System Service. It provides a way to interact with the Clock subsystem to manage the timing requests supported by the system.

Enumerations

	Name	Description
	CLK_PERIPH_BUS_MZ	Lists the available peripheral clocks.
	CLK_SOURCE_PERIPHERAL	Lists the available clock sources for the peripheral clock.
	SYS_CLK_OUTPUT	Lists the available clock outputs.
	SYS_CLK_SOURCE	Lists the available clock outputs.

Functions

	Name	Description
	SYS_CLK_ClockFailureCallbackRegister	Allows registration of a call back function that will be triggered on a clock failure.
	SYS_CLK_ClockFrequencyGet	Gets the selected clock frequency in Hz.
	SYS_CLK_ClockFrequencySet	Configures the clock multiplier and divisor values.
	SYS_CLK_ClockIsReady	Checks whether the selected clock output is stable and ready to use.
	SYS_CLK_FRCTune	This function is used for direct value based FRC oscillator tuning.
	SYS_CLK_FRCTuningSet	Starts the FRC tuning.
	SYS_CLK_Initialize	Initializes hardware and internal data structure of the System Clock.
	SYS_CLK_MZPeriphBusFreqGet	Gets the selected peripheral clock frequency in Hz (PIC32MZ only).

Structures

	Name	Description
	SYS_CLK_FRCTUNING_DATA	Defines the data required for tuning the FRC Oscillator.
	SYS_CLK_INIT	Defines the data required to initialize the Oscillator for the Clock System Service.

File Name

sys_clk.h

Company

Microchip Technology Inc.

5.7.3 Device Control System Service Library

5.7.3.1 Introduction

Device Control System Service Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Device Control System Service Library that is available on the Microchip family of PIC32 microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Device Control System Service provides the developer a simple API to configure their device for maximum performance. It does this by minimizing the number of clock cycles required to fetch instructions and data from Program Flash Memory (PFM), and SRAM. For PIC32MX devices, it also sets the peripheral clock bus to the fastest speed available for the current system clock frequency.

5.7.3.2 Release Notes

MPLAB Harmony Version: v0.70b **Device Control System Service Library Version :** 0.1 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.7.3.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER

EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.7.3.4 Using the Library

This topic describes the basic architecture of the Device Control System Service Library and provides information and examples on how to use it.

Interface Header File: [sys_devcon.h](#)

The interface to the Device Control System Service library is defined in the "[sys_devcon.h](#)" header file. This file is included by the "[system.h](#)" file. Any C language source (.c) file that uses the Device Control System Service library should include "[system.h](#)".

Library File:

The Device Control System Service library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the System Service interacts with the framework.

5.7.3.4.1 Abstraction Model

This topic provides a description of the software abstraction for the Device Control System Service.

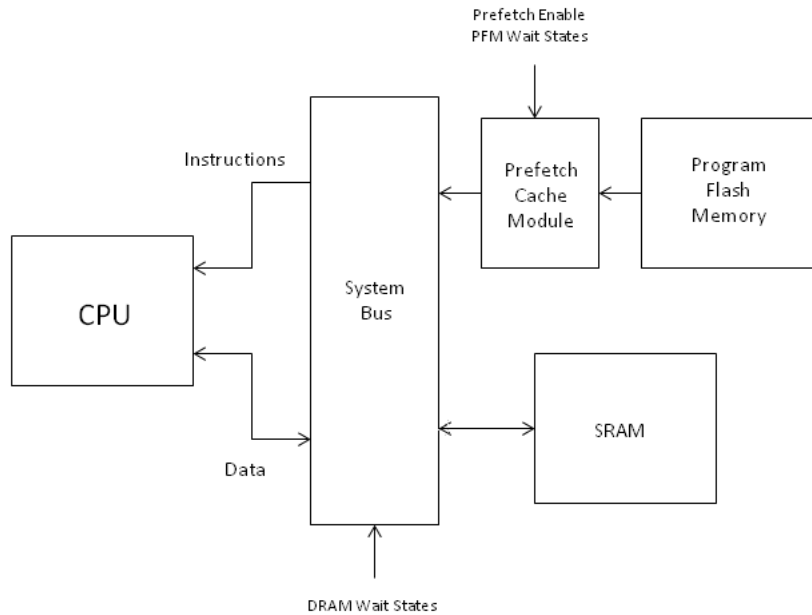
Description

Device Control Software Abstraction Block Diagram

The Device Control System Service configures the device for maximum performance given the current system clock speed. The system clock is retrieved using the System Clock service when the Device Control service is initialized. For this reason, the System Clock Service must be initialized prior to the Device Control Service.

The Device Control Service increases performance by:

- Minimizing the number of wait states inserted in an instruction fetch from Program Flash Memory (PFM).
- Minimizing the number of wait states inserted in a CPU access to Data RAM Memory (DRM) (PIC32MX only).
- Enabling the prefetch logic in the Prefetch Cache module.
- Setting the peripheral bus clock to the maximum speed (PIC32MX only).



5.7.3.4.2 Library Overview

Refer to the section [System Services Overview](#) for how the system services operates in a system.

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Device Control module.

Library Interface Section	Description
System Interaction Functions	Provides system module APIs. Device initialization, de-initialization, re-initialization and status functions.
Core Functionality	Configures Device based on current system clock

5.7.3.4.3 How the Library Works

5.7.3.4.3.1 System Interaction

Initialization and Reinitialization

Initialization of the Device Control System Service initializes an internal data structure that stores the current system clock frequency. Any time the system clock is changed, the Device Control system service should be reinitialized.

The system clock is read using the Clock System Service. Therefore, the Clock System Service must be initialized prior to initializing the Device Control System Service.

The Device Control Status and Tasks routines are provided as stubs for future enhancement. They are not currently required for this system service.

5.7.3.4.3.2 Core Functionality

The Device Control System Service currently provides a single API to configure the device for maximum performance. It uses the current clock frequency stored in an internal data structure to tune the wait states, enable the prefetch buffer and set the peripheral clock (PIC32MX only) for maximum performance. The API takes no arguments and returns void.

```
SYS_DEVCON_PerformanceConfig()
```

5.7.3.5 Configuring the Library

Macros

	Name	Description
	SYS_DEVCON_PIC32MX_MAX_PB_FREQ	This is macro SYS_DEVCON_PIC32MX_MAX_PB_FREQ.
	SYS_DEVCON_PRIMARY_CLOCK	This is macro SYS_DEVCON_PRIMARY_CLOCK.
	SYS_DEVCON_SYSTEM_CLOCK	This is macro SYS_DEVCON_SYSTEM_CLOCK.

Description

The configuration of the Device Control system service is based on the file sys_config.h

This header file contains the configuration selection for the Device Control system service. Based on the selections made, the Device Control system service will support or not support selected features. These configuration settings will apply to all instances of the Device Control system service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.7.3.5.1 Others

Three configuration options exist for the Device Control System Service.

[SYS_DEVCON_PRIMARY_CLOCK](#): Primary oscillator frequency. Not currently used.

[SYS_DEVCON_SYSTEM_CLOCK](#): System clock frequency. Retrieved using the System Clock Service. Not currently used.

[SYS_DEVCON_PIC32MX_MAX_PB_FREQ](#): Maximum peripheral bus clock frequency. Required for PIC32MX devices only.

5.7.3.5.2 Examples - Sample Functionality

```
/* Primary Oscillator clock frequency
```

Summary:

Sets up the Primary Oscillator clock frequency.

Description:

This macro sets up the Primary Oscillator clock frequency.

Remarks:

None.

```
*/
```

```
#define SYS\_DEVCON\_PRIMARY\_CLOCK 24000000L
```

```
// *****
```

```
/* System Clock frequency
```

Summary:

Sets up the System clock frequency.

Description:

This macro sets up the System clock frequency.

Remarks:

None.

```
*/
```

```
#define SYS\_DEVCON\_SYSTEM\_CLOCK 200000000L
```

```
// *****
```

```
/* PIC32MX Max Peripheral Bus Frequency
```

Summary:

Defines the maximum peripheral bus clock frequency for PIC32MX.

Description:

This macro defines the maximum peripheral bus clock frequency for PIC32MX.

Remarks:

None.

```
*/
```

```
#define SYS\_DEVCON\_PIC32MX\_MAX\_PB\_FREQ 80000000L
```

5.7.3.5.3 SYS_DEVCON_PIC32MX_MAX_PB_FREQ Macro

C

```
#define SYS_DEVCON_PIC32MX_MAX_PB_FREQ 80000000L
```

Description

This is macro SYS_DEVCON_PIC32MX_MAX_PB_FREQ.

5.7.3.5.4 SYS_DEVCON_PRIMARY_CLOCK Macro

C

```
#define SYS_DEVCON_PRIMARY_CLOCK 24000000L
```

Description

This is macro SYS_DEVCON_PRIMARY_CLOCK.

5.7.3.5.5 SYS_DEVCON_SYSTEM_CLOCK Macro

C

```
#define SYS_DEVCON_SYSTEM_CLOCK 200000000L
```

Description

This is macro SYS_DEVCON_SYSTEM_CLOCK.

5.7.3.6 Building the Library

This section list the files that are available in the \src of the Device Control System Service. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

sys_devcon.c

5.7.3.7 Library Interface

Data Types and Constants

	Name	Description
	SYS_DEVCON_HANDLE	Identifies a particular registered event instance.
	SYS_DEVCON_INIT	Identifies the system timer initialize structure.
	SYS_DEVCON_INDEX_0	Device Control System Service index definitions.

System Interaction Functions

	Name	Description
	SYS_DEVCON_Deinitialize	Deinitializes the specific module instance of the DEVCON module
	SYS_DEVCON_Initialize	Initializes data for the instance of the Device Control module and opens the specific module instance.
	SYS_DEVCON_PerformanceConfig	Configures the PFM wait states and prefetch (cache) module for maximum performance.

	SYS_DEVCON_Reinitialize	Reinitializes and refreshes the hardware for the instance of the Device Control module.
	SYS_DEVCON_Status	Returns status of the specific instance of the Device Control module.
	SYS_DEVCON_Tasks	Maintains the system Device Control state machine.
	SYS_Initialize	Function that initializes all modules in the system.

Description

5.7.3.7.1 System Interaction Functions

5.7.3.7.1.1 SYS_DEVCON_Deinitialize Function

C

```
void SYS_DEVCON_Deinitialize(
    SYS_MODULE_OBJ object
);
```

Description

This function deinitializes the specific module instance disabling its operation (and any hardware for driver modules). Resets all of the internal data structures and fields for the specified instance to the default settings.

Preconditions

The [SYS_DEVCON_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
object	SYS DEVCON object handle, returned from SYS_DEVCON_Initialize

Returns

None.

Example

```
SYS_MODULE_OBJ    object;    // Returned from SYS_DEVCON_Initialize
SYS_STATUS        status;

SYS_DEVCON_Deinitialize (object);
```

```
status = SYS_DEVCON_Status (object);
if (SYS_MODULE_DEINITIALIZED == status)
{
    // Check again later if you need to know
    // when the SYS DEVCON is deinitialized.
}
```

Remarks: Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

5.7.3.7.1.2 SYS_DEVCON_Initialize Function

C

```
SYS_MODULE_OBJ SYS_DEVCON_Initialize(
    const SYS_MODULE_INDEX index,
    const SYS_MODULE_INIT * const init
);
```

Description

This function initializes the instance of the Device Control module, using the specified initialization data. It also initializes any internal data structures.

Preconditions

None.

Parameters

Parameters	Description
index	Index for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the Device Control module. This pointer may be null if no data is required because static overrides have been provided.

Returns

If successful, returns a valid handle to an object. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#). The returned object must be passed as argument to [SYS_DEVCON_Reinitialize](#), [SYS_DEVCON_Deinitialize](#), [SYS_DEVCON_Tasks](#) and [SYS_DEVCON_Status](#) routines.

Example

```

SYS_MODULE_OBJ  objectHandle;
SYS_DEVCON_INIT  initConfig;

// Populate the devcon initialization structure
initConfig.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
initConfig.sysOscFreq           = CONFIG_OSC_FREQ;

objectHandle = SYS_DEVCON_Initialize (SYS_DEVCON_INDEX_0, (SYS_MODULE_INIT*)&initConfig);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}

```

Remarks: This routine should only be called once during system initialization unless [SYS_DEVCON_Deinitialize](#) is first called to deinitialize the device instance before reinitializing it. If the system was already initialized it safely returns without causing any disturbance.

5.7.3.7.1.3 SYS_DEVCON_PerformanceConfig Function**C**

```
void SYS_DEVCON_PerformanceConfig();
```

Description

This function configures the PFM wait states and prefetch (cache) module for maximum performance.

Preconditions

The [SYS_DEVCON_Initialize](#) function should have been called before calling this function.

Returns

None.

Example

```

SYS_DEVCON_PerformanceConfig();
Remarks:    None.

```

5.7.3.7.1.4 SYS_DEVCON_Reinitialize Function

C

```
void SYS_DEVCON_Reinitialize(
    SYS_MODULE_OBJ object,
    const SYS_MODULE_INIT * const init
);
```

Description

This function reinitializes the instance of the Device Control module using the supplied data. It modifies the internal data structure.

Preconditions

The [SYS_DEVCON_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
object	Identifies the SYS DEVCON Object returned by the Initialize interface
init	Pointer to the data structure containing any data necessary to initialize the hardware

Returns

None

Example

```
SYS_MODULE_OBJ  objectHandle;
SYS_DEVCON_INIT  initConfig;
SYS_STATUS      devconStatus;

// Populate the timer initialization structure
initConfig.moduleInit.value    = SYS_MODULE_POWER_RUN_FULL;
initConfig.sysOscFreq         = CONFIG_OSC_FREQ;

SYS_DEVCON_Reinitialize (objectHandle, (SYS_MODULE_INIT*)&initConfig);

devconStatus = SYS_DEVCON_Status (object);
if (SYS_STATUS_ERROR >= devconStatus)
{
    // Handle error
}
```

Remarks: This operation uses the same initialization data structure as the SYS_DEVCON_Initialize operation. This operation can be used to change the power state of a DEVCON module. This function can be called multiple times to reinitialize the module. This operation uses the same initialization data structure as the Initialize operation. This operation can also be used to refresh the hardware registers as defined by the initialization data.

5.7.3.7.1.5 SYS_DEVCON_Status Function

C

```
SYS_STATUS SYS_DEVCON_Status(
    SYS_MODULE_OBJ object
);
```

Description

This function returns the status of the specific module instance disabling its operation (and any hardware for driver modules).

Preconditions

The [SYS_DEVCON_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
object	SYS DEVCON object handle, returned from SYS_DEVCON_Initialize

Returns

SYS_STATUS_READY - Indicates that any previous operations have succeeded and the module is ready for additional operations.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to **accept** new operations

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been de-initialized

This value is less than SYS_STATUS_ERROR.

Example

```
SYS_MODULE_OBJ    object;    // Returned from SYS_DEVCON_Initialize
SYS_STATUS        tmrStatus;
```

```
devconStatus = SYS_DEVCON_Status (object);
else if (SYS_STATUS_ERROR >= devconStatus)
{
    // Handle error
}
```

Remarks: Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

5.7.3.7.1.6 SYS_DEVCON_Tasks Function**C**

```
void SYS_DEVCON_Tasks(
    SYS_MODULE_OBJ object
);
```

Description

This function is used to maintain the system Device Control internal state machine.

Preconditions

The [SYS_DEVCON_Initialize](#) function must have been called for the specified DEVCON driver instance.

Parameters

Parameters	Description
object	SYS DEVCON object handle, returned from SYS_DEVCON_Initialize

Returns

None

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)) or by the appropriate raw ISR.

Example

```
SYS_MODULE_OBJ      object;      // Returned from SYS_DEVCON_Initialize

while (true)
{
    SYS_DEVCON_Tasks (object);

    // Do other tasks
}
```

5.7.3.7.1.7 SYS_Initialize Function

C

```
void SYS_Initialize(
    void * data
);
```

Description

System Initialization Function

This function initializes all modules in the system, including any drivers, services, middleware, and applications.

Preconditions

None.

Parameters

Parameters	Description
data	Pointer to the data structure containing any data necessary to initialize the module. This pointer may be null if no data is required and default initialization is to be used.

Returns

None.

Remarks

This function will only be called once, after system reset.

Example

```
SYS_Initialize ( NULL );

while ( true )
{
    SYS_Tasks ( );
}
```

5.7.3.7.2 Data Types and Constants

5.7.3.7.2.1 SYS_DEVCON_HANDLE Type

C

```
typedef int8_t SYS_DEVCON_HANDLE;
```

Description

SYS DEVCON Handle

This event handle identifies a registered instance of an event. Every time the application that tries to access the paramateres with respect to a particular event, shall used this event handle to refer to that event.

Remarks

None.

5.7.3.7.2.2 SYS_DEVCON_INIT Structure

C

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    uint32_t sysOscFreq;
} SYS_DEVCON_INIT;
```

Description

SYS DEVCON Initialize structure

This structure identifies the system timer initialize structure.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
uint32_t sysOscFreq;	System Oscillator Frequency in Hertz

Remarks

None.

5.7.3.7.2.3 SYS_DEVCON_INDEX_0 Macro

C

```
#define SYS_DEVCON_INDEX_0 0
```

Description

SYS Device Control Module Index Numbers

These constants provide Device Control System Service index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

5.7.3.8 Files

Files

Name	Description
sys_devcon.h	Device Control System Service interface definition.
sys_devcon_config.h	Device Control System Service configuration templates.

Description

5.7.3.8.1 sys_devcon.h

Device Control System Service Interface Definition

This file contains the interface definition for the Device Control System Service. It provides a way to interact with the Device Control subsystem to manage the device control requests supported by the system.

Functions

	Name	Description
	SYS_DEVCON_Deinitialize	Deinitializes the specific module instance of the DEVCON module
	SYS_DEVCON_Initialize	Initializes data for the instance of the Device Control module and opens the specific module instance.
	SYS_DEVCON_PerformanceConfig	Configures the PFM wait states and prefetch (cache) module for maximum performance.
	SYS_DEVCON_Reinitialize	Reinitializes and refreshes the hardware for the instance of the Device Control module.
	SYS_DEVCON_Status	Returns status of the specific instance of the Device Control module.
	SYS_DEVCON_Tasks	Maintains the system Device Control state machine.

Macros

	Name	Description
	SYS_DEVCON_INDEX_0	Device Control System Service index definitions.

Structures

	Name	Description
	SYS_DEVCON_INIT	Identifies the system timer initialize structure.

Types

	Name	Description
	SYS_DEVCON_HANDLE	Identifies a particular registered event instance.

File Name

sys_devcon.h

Company

Microchip Technology Inc.

5.7.3.8.2 sys_devcon_config.h

Device Control System Service Configuration Templates

This file contains constants to configure the Device Control System Service.

Macros

	Name	Description
	SYS_DEVCON_PIC32MX_MAX_PB_FREQ	This is macro SYS_DEVCON_PIC32MX_MAX_PB_FREQ.
	SYS_DEVCON_PRIMARY_CLOCK	This is macro SYS_DEVCON_PRIMARY_CLOCK.
	SYS_DEVCON_SYSTEM_CLOCK	This is macro SYS_DEVCON_SYSTEM_CLOCK.

File Name

sys_devcon_config_template.h

Company

Microchip Technology Inc.

5.7.4 File System Service Library

5.7.4.1 Introduction

Introduction to the MPLAB Harmony File System (FS).

Description

The MPLAB Harmony File System (FS) provides file system services to MPLAB Harmony based applications. Figure 1 shows the architecture of the File System Service.

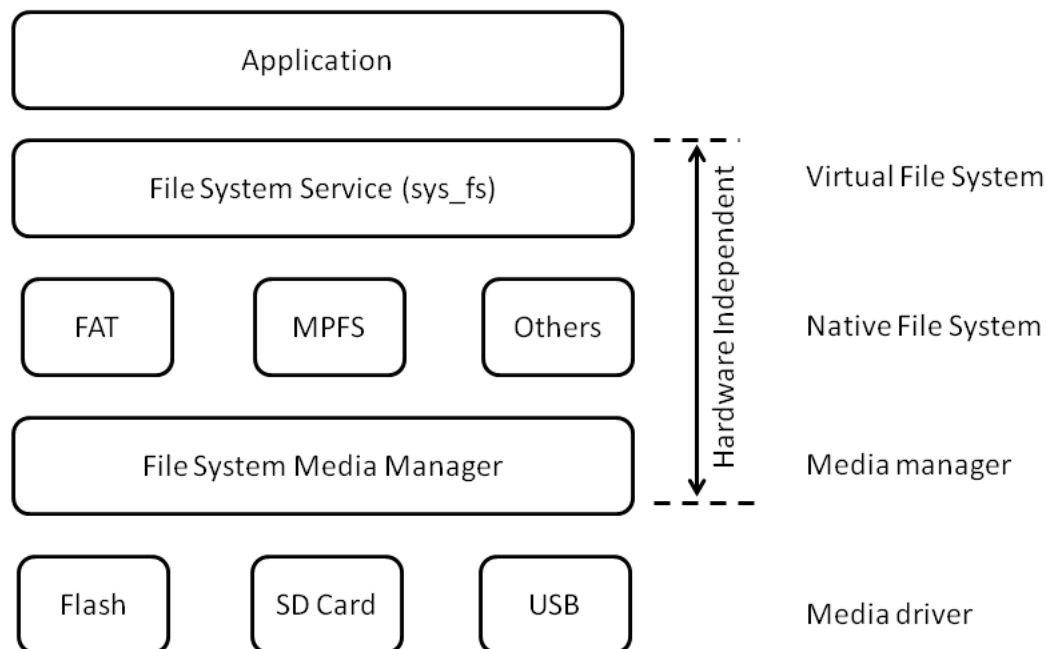


Figure 1: File System Architecture

The File System Service provides an application programming interface (API) through which a utility or user program requests services of a file system. Some file system APIs may also include interfaces for maintenance operations, such as creating or initializing a file system and verifying the file system for integrity.

The File System service is really a framework designed to support multiple file systems (native file system) and multiple media in the same application. Examples of native file systems are FAT12, FAT32, MPFS, and JFS, among others. Each of these native file systems have a common set of APIs that can be used to access the files of that particular native file system.

The File System Service abstracts the native file system calls and provides a common interface to the user/application layer. For example, while the application layer requests for a file read or write from a disk, due to the presence of this abstraction, the application need not be bothered about the native file system implemented on that disk. Instead, the application can call the

read/write API of the File System, which in turn translates to the read/write command of the native file system used on the required disk.

This simplifies the implementation of the higher application layer and also provides a mechanism to add more native file system to the File System framework in the future.

Note: Throughout this document, "File System Service" and "sys_fs" are used interchangeably.

5.7.4.2 Release Notes

MPLAB Harmony Version: v0.70b **File System Service Library Version :** 0.1 **Release Date:** 18Nov2013

New in This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

- Supports multiple file systems. Presently, FAT FS and MPFS are the two native file systems that are supported.
- Supports multiple media.
- Supports both static media (SD card, non-volatile memory, etc.) and dynamic media (mass storage device).
- Supports long file names for files.
- Supports media with multiple partitions.

Known Issues:

- Does not work in MIPS16 mode.
- For some SD cards, reading or writing data of more than 1 sector (512Bytes) causes random junk characters to be inserted in the data. Hence, it is advised to read or write 512Bytes or lesser at a time.
- SD cards (size 2 GB or less) do not work with the present version of File system. Only SDHC cards work.
- Hardware switch-based write protection feature is not implemented for SD cards.
- Since the FS demonstration codes do not have an integrated real-time clock module, the time is set internally to a fixed value of 9th August 2013, 3:06pm (in the file diskio.c).
- Not all functions (such as directory related functions) from native FAT FS are supported in the present version of the MPLAB Harmony FS. For the list of implemented functions, please refer to the Library Interface section.
- SD cards do not support dynamic card insertion. To use an SD card, the card has to be inserted before running the code execution.
- Testing on multi-partitioned media was not carried out extensively.

5.7.4.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR

CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.7.4.4 Getting Started

This topic provides an introduction to the MPLAB Harmony File System.

Description

The MPLAB Harmony File System (FS) provides embedded application developers with a file system framework for retrieving and storing data from various media.

The MPLAB Harmony file system is designed to support multiple file systems (native file systems) and multiple media at the same time. Examples of native file systems are FAT12, FAT32, MPFS, and JFS, among others. Each of these native file systems has a common set of APIs that can be used to access the files of that particular native file system. The FS is a part of the MPLAB Harmony installation and is accompanied by demonstration applications that highlight usage. These demonstrations can also be modified or updated to build custom applications.

FS features include the following:

- Support for multiple file system (FAT, MPFS)
- Supports multiple physical media (NVM, SD card)
- More physical media can be interfaced with the FS, once the driver is available for the media
- Modular and Layered architecture

5.7.4.4.1 Architecture

This topic describes the architecture of the MPLAB Harmony File System.

Description

The FS framework features a modular and layered architecture, as shown in Figure 1.

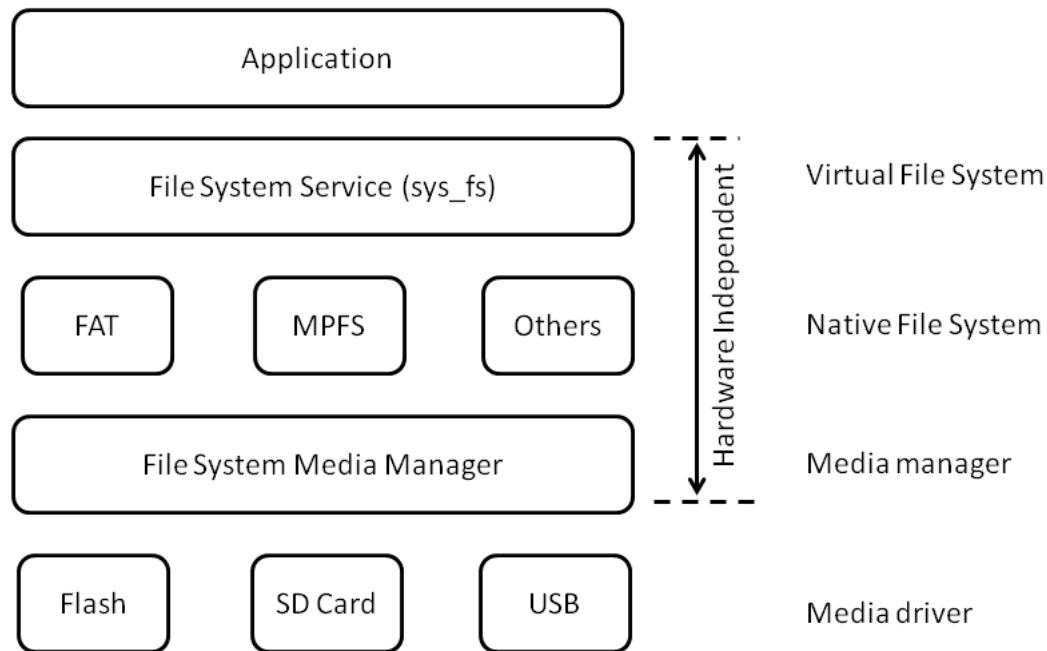


Figure 1: FS Framework Architecture

As seen in Figure 1, the FS Framework consists of the following major blocks:

- The **Driver** for the physical media has to be included as a part of the FS Framework. This layer provides a low-level interface to access the physical media. This layer also enables multiple instances of media. Examples of drivers are:
 - NVM driver – To access files using NVM (Flash memory)
 - SPI driver – To access files from SD card, which interfaces using the SPI peripheral
- The **Media driver** provides a mechanism to access the media as “sectors”. Sectors are the smallest storage element accessed by a file system and are contiguous memory locations. Typically, each sector has 512 bytes. Depending on the requirement, in some cases, the driver and media driver could be combined as one layer.
- The **Media manager** implements a disk and sector based media access mechanism. It also performs disk allocated/deallocated on media attach/detach. Due to the implementation of this layer, the FS Framework can support multiple disks.
- The **Native file system** implements support for the media file system format. Examples of native file systems are: FAT12, FAT32, and MPFS, among other. At present, only the FAT and MPFS file systems are supported by the FS framework; however, more native file systems can be included.
- The **Virtual file system (or SYS_FS)** layer provides a file system independent file system operation interface. This layer translates virtual file systems calls to native file system calls. Due to this layer, applications can now support multiple file systems. Interfaces provided by this layer, but not limited to, include:
 - SYS_FS_mount
 - SYS_FS_open
 - SYS_FS_read
 - SYS_FS_write
 - SYS_FS_close

5.7.4.4.2 Application Interaction

This topic describes how an application must interact with the File System.

Description

The interaction of various layers is shown in Figure 1.

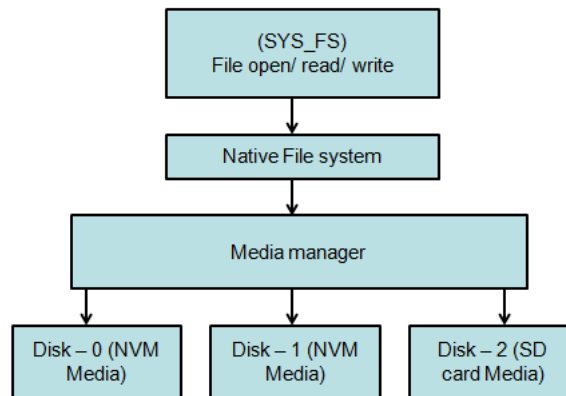


Figure 1: Application Interaction with FS Framework

In the process of using the FS Framework, the application must first **mount** the media drive for the FS Framework to access the media. Unless the mounting process returns successfully, the application should continue trying to mount the drive. If the drive is not attached, the mounting process will fail. In such a situation, the application should not proceed further unless the mounting is success.

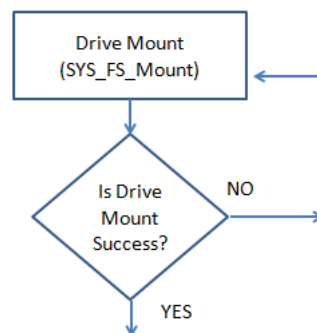
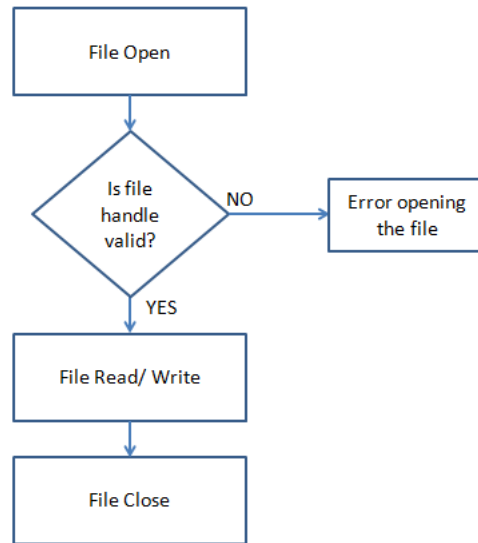


Figure 2: Application Mounts a Drive

Once the drive is mounted, the application code can then **open the file** from the drive with different attributes (such as read-only or write). If the file open returns a valid handle, the application can proceed further. Otherwise, the application will enter an error state. The reason for an invalid handle could be that the application was trying to read a file from the drive that does not exist. Another reason for an invalid handle is when the application tries to write to a drive that is write-protected.

Once the file is opened, the valid file handle is further used to **read/write** data to the file. Once the required operation is performed on the file, the file can then be **closed** by the application by passing the file handle. Figure 3 illustrates the process.

**Figure 3: Further File System Operations**

5.7.4.4.3 Using the File System

This topic describes how to use the File System.

Description

Use the Available Library Demonstration Applications

The FS framework release package contains a set of demonstration applications that are representative of common scenario (single/multi-media and single/multi-native file systems). These demonstrations can be easily modified to include application-specific initialization and application logic. The application logic must be non-blocking and could be implemented as a state machine.

- The application specific initialization can be called in the [SYS_Initialize\(\)](#) function (in the sys_init.c file). The [SYS_Initialize](#) is called when the device comes out of Power-on Reset (POR).
- The application logic can be called in the [SYS_Tasks\(\)](#) function (in the sys_tasks.c file). The application logic can interact with the FS layer by using relevant API calls, as provided in the [APP_Tasks\(\)](#) (in the app.c file)

Building a FS Application from Scratch

In a case where the available demonstration applications do not meet the end application requirements, an application to use the FS framework can be created by from scratch. Figure 1 shows a flowchart for the steps that need to be performed.

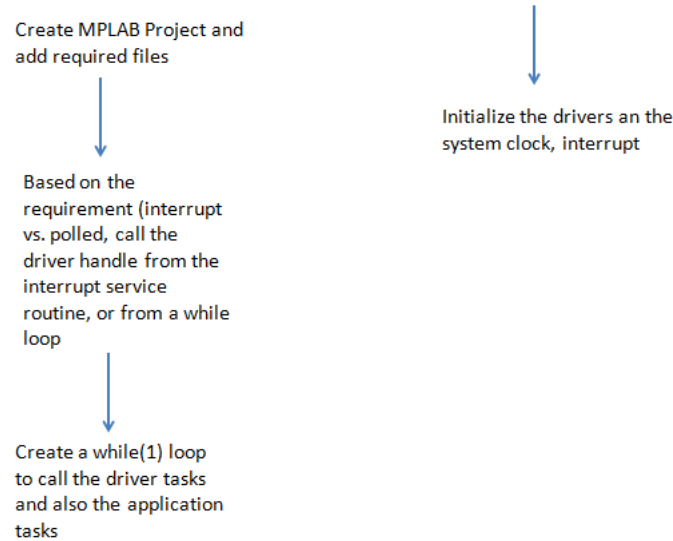


Figure 1: Steps to create a FS Application

Step 1:

Create a MPLAB Project and add the required FS framework files to the project. The following files are needed to build the a FS project

- system_config.h - This file should contain the compile time configuration macros for the Driver, Media Layer, and FS layer. The file also contains the clock speed setting, which is set for the microcontroller.
- system_init.c – This file should contain the initial settings for each driver. It should also call the functions required to initialize different drivers to be used by the FS.
- ff.c, diskio.c, mpfs.c, sys_fs.c, sys_fs_media_manager.c – These files are part of the FS, which must be included in the project
- sys_int_pic32.c and plib_int_pic32.c - These files implement the system interrupt service that is required by the FS
- sys_ports.c – If the FS is using a SD card as media, this file needs to be included (for Chip Select)
- Driver – The driver for media to be used by FS should also be included
- Application specific files - These file will implement the application logic

Step 2:

Since the MPLAB Harmony drivers included with the File System operate in interrupt mode, a driver Handler should be defined as follows:

```

/* Use this for PIC32MX */
void __ISR ( _SPI1_VECTOR, ipl4 ) _InterruptHandler_SPI_stub ( void )
{
    DRV_SPI_Tasks((SYS_MODULE_OBJ)appDrvObjects.drvSPIObject);
}

/* Use this for PIC32MZ */
void __ISR ( _SPI2_RX_VECTOR, ipl4 ) _InterruptHandler_SPI_RX_stub ( void )
{
    DRV_SPI_Tasks((SYS_MODULE_OBJ)appDrvObjects.drvSPIObject);
}

void __ISR ( _SPI2_TX_VECTOR, ipl4 ) _InterruptHandler_SPI_TX_stub ( void )
{
    DRV_SPI_Tasks((SYS_MODULE_OBJ)appDrvObjects.drvSPIObject);
}
  
```

Step 3:

The application should create a while(1) loop that continuously updates the driver layer State Machine and the application state machine. This requires the application state machine to be non-blocking.

/ This while(1) loop will continuously update the driver layer state machine and the application state machine */*

```
while(1)
{
    /* Task routine for sys_fs */
    SYS_FS_Tasks();

    /* Call the SDCARD Task */
    DRV_SDCARD_Tasks(appDrvObjects.drvSDCARDObject);

    /* Call the application's tasks routine */
    APP_Tasks ( );
}
```

Step 4:

If interrupt-based operation is needed, the interrupts need to be enabled first. The application should then initialize the driver layer. Refer to the driver specific documents regarding usage.

/ Initialize the interrupt system */*

```
SYS_INT_Initialize();
```

/ Initialize the global interrupts */*

```
SYS_INT_Enable();
```

/ set priority for SPI interrupt source */*

```
SYS_INT_VectorPrioritySet(INT_VECTOR_SPI1, INT_PRIORITY_LEVEL3);
```

/ set sub-priority for SPI interrupt source */*

```
SYS_INT_VectorSubprioritySet(INT_VECTOR_SPI1, INT_SUBPRIORITY_LEVEL3);
```

/ Initialize the global interrupts */*

```
SYS_INT_Enable();
```

/ Initialize the SPI driver */*

```
appDrvObjects.drvSPIObject = DRV_SPI_Initialize(DRV_SPI_INDEX_0,
    (SYS_MODULE_INIT *)&drvSPIInit);
```

/ Initialize the SDCARD driver*/*

```
appDrvObjects.drvSDCARDObject = DRV_SDCARD_Initialize(DRV_SDCARD_INDEX_0,
    (SYS_MODULE_INIT *)&drvSDCARDInit);
```

Step 5:

The application code can be implemented as a non-blocking state machine inside the APP_TASKS() function, as shown below for an application to read a file and write the content into another newly created file.

The input file "TEST.JPG" is not provided with the release package. It could be any arbitrary JPEG file chosen by the user, and then suitably renamed to "TEST.JPG". The reason for choosing a JPEG file for test purposes is that the duplicate file "TEST1.JPG" created by the FS demonstration could be easily verified for correctness by inserting the SD card in the computer and opening the "TEST1.JPG" file.

```
void APP_Tasks ( void )
{
    /* The application task state machine */

    switch(appData.state)
    {
```

```

case APP_MOUNT_DISK:
    if(SYS_FS_Mount("/dev/mmcblk1", "/mnt/myDrive", FAT, 0, NULL) != 0)
    {
        /* The disk could not be mounted. Try
         * mounting again untill success. */

        appData.state = APP_MOUNT_DISK;
    }
    else
    {
        /* Mount was successful. Open a file.
         * Let the switch case fall through. */
        appData.state = APP_OPEN_FILE;
    }
    break;

case APP_OPEN_FILE:

    appData.fileHandle = SYS_FS_FileOpen("/mnt/myDrive/TEST.JPG",
        (FA_READ));
    if(appData.fileHandle == SYS_FS_HANDLE_INVALID)
    {
        /* Could not open the file. Error out*/
        appData.state = APP_ERROR;
    }
    else
    {
        appData.fileHandle1 = SYS_FS_FileOpen("/mnt/myDrive/TEST1.JPG",
            (FA_WRITE|FA_CREATE_ALWAYS));
        if(appData.fileHandle == SYS_FS_HANDLE_INVALID)
        {
            /* Could not open the file. Error out*/
            appData.state = APP_ERROR;
        }
        else
        {
            /* Check the file to be read */
            appData.state = APP_CHECK_FILE;
        }
    }
    break;

case APP_CHECK_FILE:
    /* check the size of file */
    fileSize = SYS_FS_FileSize(appData.fileHandle);
    /* since, we will read 512 bytes at a time, find the number of times, the read has
    to be performed */
    sectorCounter = integralSector = (fileSize/512);
    /* find the remaining bytes */
    balanceSector = (fileSize%512);

    appData.state = APP_READ_WRITE_TO_FILE;

    break;

case APP_READ_WRITE_TO_FILE:

    if(SYS_FS_FileRead((void *)appData.data, 512, appData.fileHandle) == -1)
    {
        /* There was an error while reading the file.
         * Close the file and error out. */

        SYS_FS_FileClose(appData.fileHandle);
        appData.state = APP_ERROR;
    }

```



```

-1)    else if(SYS_FS_FileWrite((const void *)appData.data, 512, appData.fileHandle1) ==
      {
        /* Write was not successful. Close the file
         * and error out.*/
        SYS_FS_FileClose(appData.fileHandle1);
        appData.state = APP_ERROR;
      }
      else
      {
        sectorCounter--;
        /* if entire integral sectors are written, then write the balance sector*/
        if(sectorCounter == 0)
        {
          if(SYS_FS_FileRead((void *)appData.data, balanceSector,
appData.fileHandle) == -1)
          {
            /* There was an error while reading the file.
             * Close the file and error out. */

            SYS_FS_FileClose(appData.fileHandle);
            appData.state = APP_ERROR;
          }
          else if(SYS_FS_FileWrite((const void *)appData.data, balanceSector,
appData.fileHandle1) == -1)
          {
            /* Write was not successful. Close the file
             * and error out.*/
            SYS_FS_FileClose(appData.fileHandle1);
            appData.state = APP_ERROR;
          }
          else
          {
            appData.state = APP_CLOSE_FILE;
          }
        }
      }
      break;

case APP_CLOSE_FILE:
  /* Close both files */
  SYS_FS_FileClose(appData.fileHandle);
  SYS_FS_FileClose(appData.fileHandle1);
  /* The test was successful. Lets idle. */
  appData.state = APP_IDLE;
  break;

case APP_IDLE:
  /* The application comes here when the demo
   * has completed successfully. Switch on
   * green LED. */
  BSP_SwitchONLED(LED_2);
  break;

case APP_ERROR:
  /* The application comes here when the demo
   * has failed. Switch on the red LED.*/
  BSP_SwitchONLED(LED_1);
  break;

default:
  break;
}

} //End of APP_Tasks

```

5.7.4.5 Migration Guide

This section provides information for migrating an existing File System (FS) application that uses legacy MLA code to MPLAB Harmony.

Description

The topics covered include:

- [Initialization](#)
- [System Configuration](#)
- [Mounting a Volume](#)
- [Opening a File](#)
- [Reading a File](#)
- [Writing a File](#)
- [Closing a File](#)
- [File EOF](#)
- [File Tell](#)
- [File Seek](#)
- [SYS_FS_Tasks](#)

In each topic, differences between the legacy MLA code and MPLAB Harmony are described and examples are provided to assist with the migration.

5.7.4.5.1 Comparison of API Names

This topic tabulates the API names for MLA vs. the name for MPLAB Harmony.

Description

API Name in MLA	API Name in MPLAB Harmony
N/A	SYS_FS_Mount
N/A	SYS_FS_Unmount
N/A	SYS_FS_FileError
FSInit	SYS_FS_Initialize
N/A	SYS_FS_Tasks
Fsfclose	SYS_FS_FileClose
Fsfeof	SYS_FS_FileEOF
FSfread	SYS_FS_FileRead
FSftell	SYS_FS_FileTell
FSfwrite	SYS_FS_FileWrite
N/A	SYS_FS_FileSize
wFSfopen	SYS_FS_FileOpen
N/A	SYS_FS_FileStat
FSfseek	SYS_FS_FileSeek

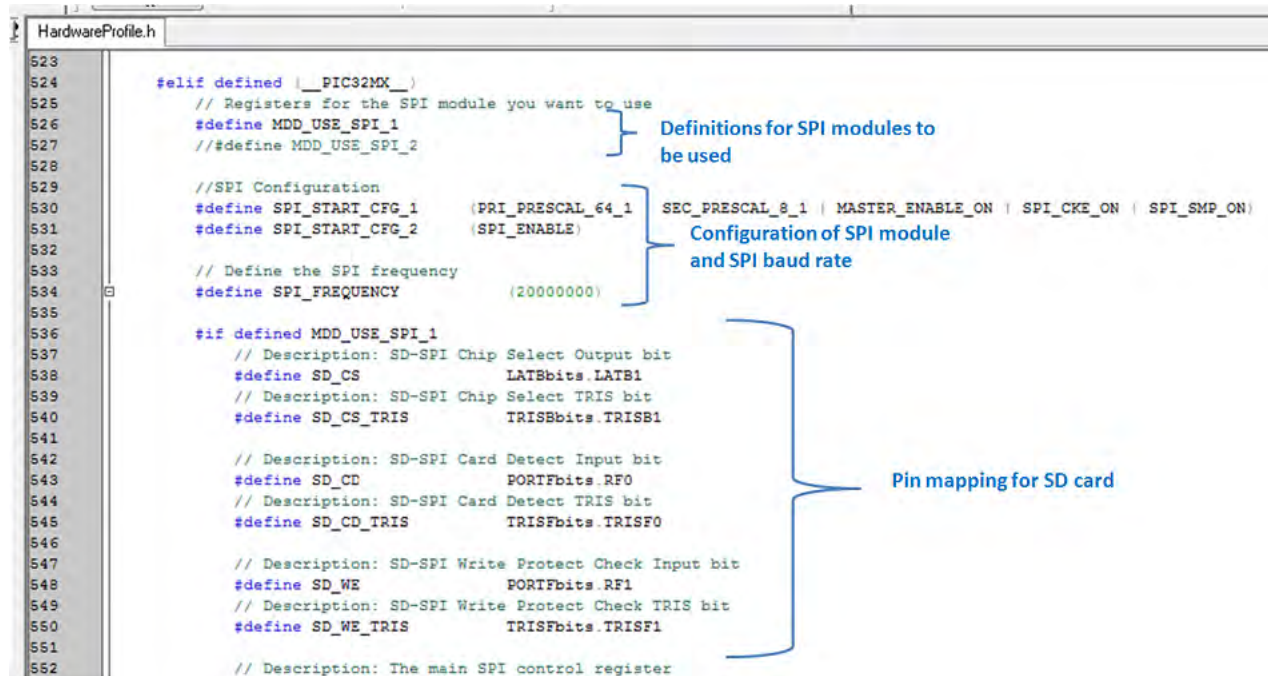
5.7.4.5.2 Initialization

This topic describes the differences in initialization.

Description

MLA Initialization (Legacy code):

In legacy MLA code, the initialization involves modification of the header file "HardwareProfile.h". The header file has definitions for the SPI modules used (SPI1 or SPI2), configuration for SPI module, clock frequency, pin mapping for SD card, and pin remapping for SPI (SDI, SDO and SCLK pins). The following two images illustrate these modifications.



```

HardwareProfile.h
566 // Description: The definition for the SPI baud rate generator register (PIC32)
567 #define SPIBRG          SPI1BRG
568
569 // Tris pins for SCK/SDI/SDO lines
570
571 // Tris pins for SCK/SDI/SDO lines
572 #if defined(__32MX460F512L__)
573 // Description: The TRIS bit for the SCK pin
574 #define SPICLOCK          TRISDbits.TRISD10
575 // Description: The TRIS bit for the SDI pin
576 #define SPIIN             TRISCbits.TRISC4
577 // Description: The TRIS bit for the SDO pin
578 #define SPIOUT            TRISDbits.TRISD0
579 #else // example: PIC32MX360F512L
580 // Description: The TRIS bit for the SCK pin
581 #define SPICLOCK          TRISFbits.TRISF6
582 // Description: The TRIS bit for the SDI pin
583 #define SPIIN             TRISFbits.TRISF7
584 // Description: The TRIS bit for the SDO pin
585 #define SPIOUT            TRISFbits.TRISF8
586 #endif
587
588 //SPI library functions
589 #define putcSPI            putcSPI1
590 #define getcSPI            getcSPI1
591 #define OpenSPI(config1, config2) OpenSPI1(config1, config2)
592
593 #elif defined MDD_USE_SPI_2
594 // Description: SD-SPI Chip Select Output bit
595 #define SD_CS             LATBbits.LATB9

```

Device specific SPI pin remapping based on "#define"

SPI module number specific functions based on "#define"

Inside the main() function, the initialization of interrupts and the clock occurs in a series of function calls. Then, the media detect function is called for the media used. Later, the FS init function is called, which internally performs the disk mount, and if the mount process and FS initialization is successful, the functions returns as "1".

```

Demonstration3.c
177
178 int main(void)
179 {
180     FSFILE * pointer;
181     #if defined(SUPPORT_LFN)
182     char count = 80;
183     #endif
184     char * pointer2;
185     SearchRec rec;
186     unsigned char attributes;
187     unsigned char size = 0, i;
188
189     // Turn on the interrupts
190     INTEnableSystemMultiVectoredInt();
191     SYSTEMConfigPerformance(GetSystemClock());
192     mOSCSetPBDIV(OSC_PB_DIV_2);
193     //Initialize the RTCC
194     RtccInit();
195     while(RtccGetClkStat() != RTCC_CLK_ON); // wait for the SOSC to be actually running and RTCC to have its clock source
196     // could wait here at most 32ms
197     RtccOpen(0x10073000, 0x07011602, 0);
198
199     while (!MDD_MediaDetect());
200
201     // Initialize the library
202     while (!FSInit());
203
204     #ifdef ALLOW_WRITES
205     // Create a file
206     pointer = wFSfopen ((const unsigned short int *) &str1[0], "w");

```

main() function

Clock and interrupt initialization

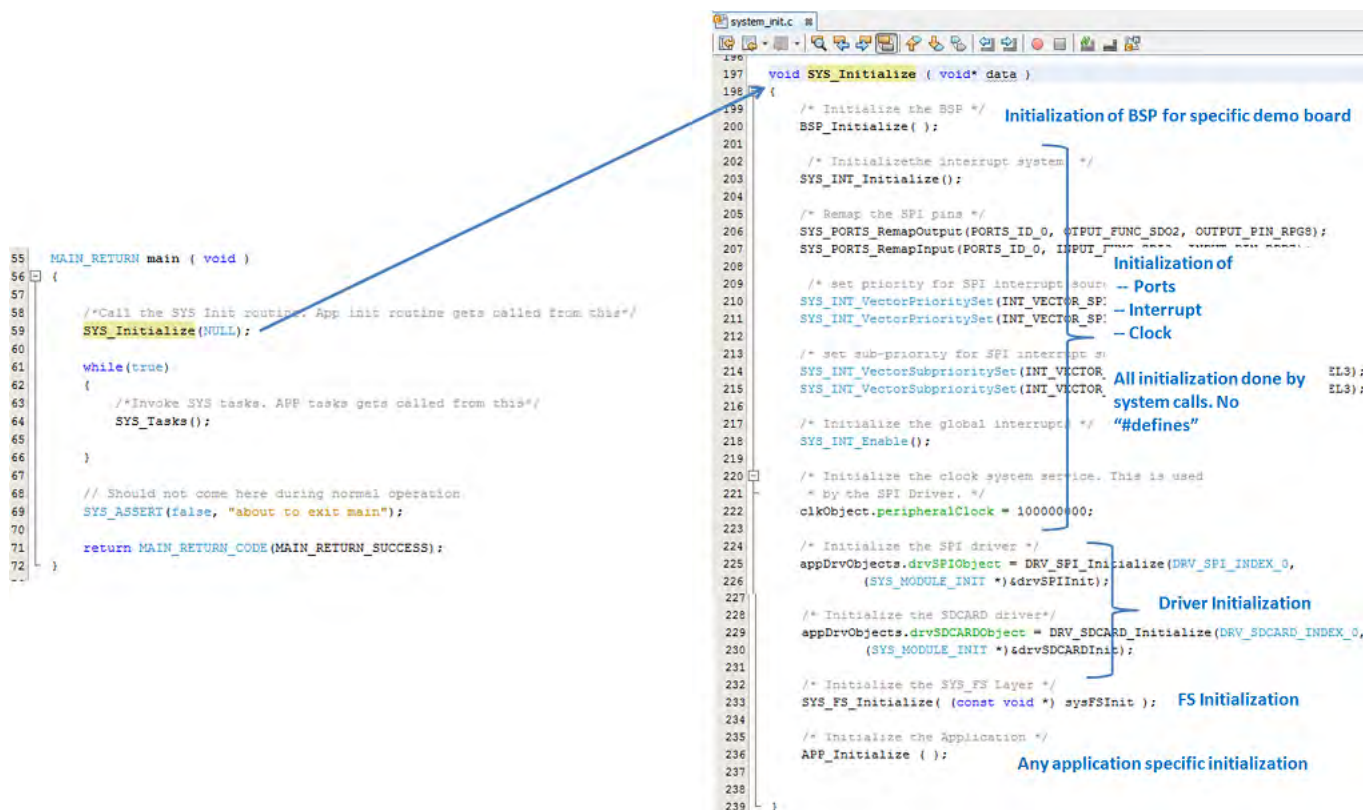
Media detection and FS initialization

MPLAB Harmony Initialization:

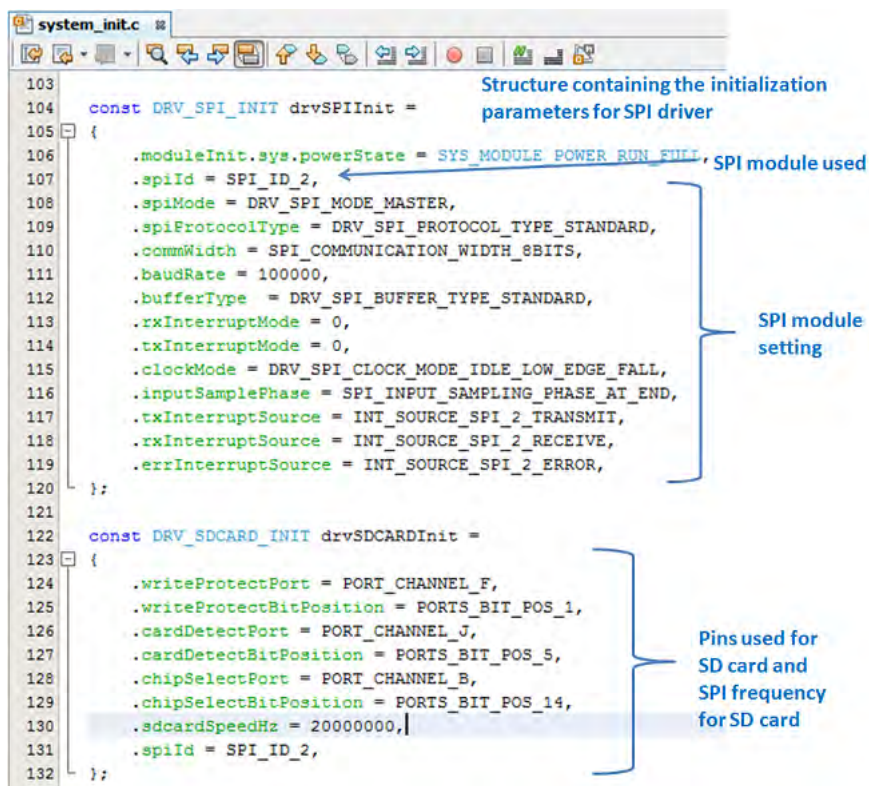
In the case of MPLAB Harmony, the initialization is done by the sys_initialize() function, which is called from the main() function.

Inside the sys_initialize() function, the BSP, interrupt, clock, port remapping, driver initialization, FS initialization and application-specific initialization is done. All initialization is done by calling specific system function calls, as shown in the following image. For more information, refer to the demonstration code provided as a part of the MPLAB Harmony release

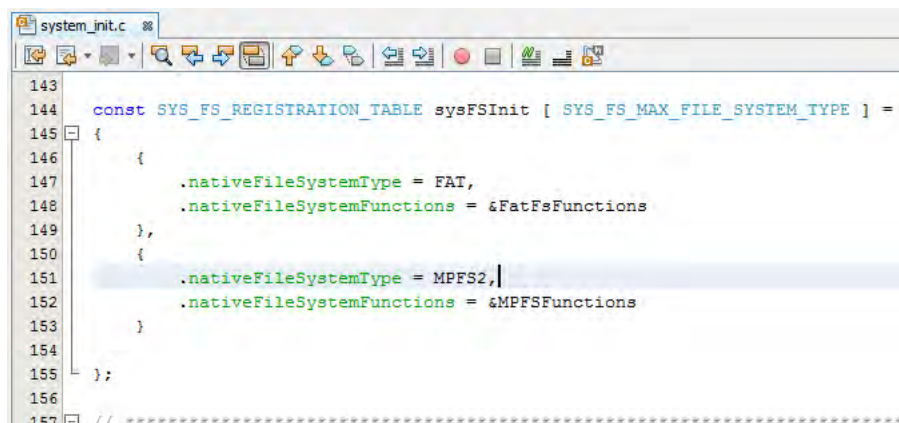
package.



The SPI module selection and configuration of the SPI module is done through the SPI driver initialization structure. Similarly, the selection of pins related to SD card functionality and the selection of the SPI clock frequency is done through the SD card driver initialization structure. The initialization structures are passed as input parameters during driver initialization function calls. There are no #define used, as was done in the case of legacy MLA code.



Since MPLAB Harmony supports multiple file systems, there is a structure which needs to be defined and then passed as an input parameter to the `SYS_FS_Init()` function.



5.7.4.5.3 System Configuration

This topic describes differences in system configuration.

Description

system_config.h file in MPLAB Harmony:

The file "system_config.h" contains the various system configurations required to run an application. Examples of system configurations related to, but not limited to drivers are:

- SPI driver configuration: such as `DRV_SPI_INSTANCES_NUMBER`, `DRV_SPI_CLIENTS_NUMBER`,

[DRV_SPI_INTERRUPT_MODE](#), [DRV_SPI_BUFFER_SIZE](#), etc.

- SD Card driver configuration: such as [DRV_SDCARD_INSTANCES_NUMBER](#), [DRV_SDCARD_CLIENTS_NUMBER](#), [DRV_SDCARD_QUEUE_POOL_SIZE](#), etc.

These configurations are required to configure the drivers used for the FS. Since the driver concept was not present in the legacy MLA code, this is a new addition to MPLAB Harmony. To get detailed information about each of the configuration parameter, please refer to the driver help.

Media manager configuration:

- [SYS_FS_MEDIA_NUMBER](#) - Number of media that will be used in the application. For example, if the application uses SD card and Mass storage device the [SYS_FS_MEDIA_NUMBER](#) should be defined as 2.
- [SYS_FS_VOLUME_NUMBER](#) - Number of volumes that will be used in the application. MPLAB Harmony supports multi-partitioned media. If the application uses a SD card that has three partitions and a Mass storage device, which has one partition, the [SYS_FS_VOLUME_NUMBER](#) should be defined as 4.
- Clock related configurations are present in the file, which are used to set the clock for the device

File system related configuration:

- [SYS_FS_MAX_FILE_SYSTEM_TYPE](#) - Number of native file system that will be used in the application. MPLAB Harmony supports multiple native file systems. If the application uses two file systems, FAT FS and MPFS2, [SYS_FS_MAX_FILE_SYSTEM_TYPE](#) should be defined as 2.
- [SYS_FS_MAX_FILES](#) - Maximum number of files that will be opened by the application at a time
- Other application-specific configuration settings

[SYS_Tasks\(\)](#) in MPLAB Harmony:

In the case of MPLAB Harmony, certain tasks need to be executed periodically. These tasks are executed from a common function called [SYS_Tasks\(\)](#), which is in turn called from the main loop (while(1)) loop. While the concept of running certain functions from the main loop is not new, the legacy MLA demonstration code did not have any functions that had to run from the main loop. However, in case of MPLAB Harmony, it is mandatory that certain tasks should be running periodically from the main loop.

In the case of the FS application, tasks that have to run from [SYS_Tasks\(\)](#) function are:

- [SYS_FS_Tasks\(\)](#) - This task maintains the working of the [SYS_FS](#) layer and other file system related layers. It is extremely essential that this function runs periodically from the [SYS_Tasks\(\)](#) function.
- Driver task - Consider a case where the FS application uses a SD card -- there is no hardware module that controls the SD card. In such a case, the task routine for the SD card must run from [SYS_Tasks\(\)](#). This is required so that the internal driver mechanism for the SD card continues working.
- Any other application related task

```

88      Remarks:
89      When not using the dynamic system "Tasks" service, this routine must be
90      implemented by the application's system configuration (in the application's
91      configuration-specific "sys_tasks.c" file).
92      */
93
94      void SYS_Tasks ( void )
95      {
96          /* Task routine for sys_fs */
97          SYS_FS_Tasks();
98
99          /* Call the SDCARD Task */
100         DRV_SDCARD_Tasks(appDrvObjects.drvcSDCARDObject);
101
102         /* Call the application's tasks routine */
103         APP_Tasks ( );
104     }
105

```

5.7.4.5.4 Mounting a Volume

This topic describes differences when mounting a volume.

Description

Mounting a Volume in MLA vs. MPLAB Harmony:

In case of MLA, the complete mounting of a volume was achieved by calling the two functions, MDD_MediaDetect() and FSInit(), until the function returns success. The features of mounting a volume in MLA were:

- There are two functions that completely achieve the mounting process
- Both functions were of input type as "void" (no input argument)
- Both functions were called in blocking mode

```

193      //Initialize the RTCC
194      RtccInit();
195      while(RtccGetClkStat() != RTCC_CLK_ON); // wait for the SOSC to be actually runni
196                                          // could wait here at most 32ms
197      RtccOpen(0x10073000, 0x07011602, 0);
198
199      while (!MDD_MediaDetect());
200
201      // Initialize the library
202      while (!FSInit());
203
204      #ifdef ALLOW_WRITES
205      // Create a file
206      pointer = wFSfopen ((const unsigned short int * )sstr1[0], "w");
207      if (pointer == NULL)
208          while(1);
209

```

In the case of MPLAB Harmony, the mounting of a volume is achieved by calling the [SYS_FS_Mount\(\)](#) function, until the function returns success. The features of mounting a volume in MPLAB Harmony are:

- [SYS_FS_Mount\(\)](#) function includes both media detection and mounting the volume. Please note that, MPLAB Harmony FS still needs the function [SYS_FS_Initialize\(\)](#) to be called during system initialization. Though the name [SYS_FS_Initialize\(\)](#) seems similar to the MLA function FSInit(), the task achieved by calling these functions are very different.
- [SYS_FS_Initialize\(\)](#) (MPLAB Harmony function) - Just does the initialization of SYS_FS layer. It does not do any mounting of volumes.
- FSInit() (MLA function) - Did the initialization of FS and also did mount the disk.
- [SYS_FS_Mount\(\)](#) function accepts input parameter such as devName, mntName, fileType etc. To know each of the parameter in detail, please refer to the documentation of the [SYS_FS_Mount\(\)](#) function.

- `SYS_FS_Mount()` has to be running in a non-blocking mode. The following image shows an application where `SYS_FS_Mount()` is called from the state machine implementation. This is required as the implementation of the state machine allows the `SYS_Tasks()` function to be run periodically. This is quite different from the earlier implementations of MLA.

```

193 //Initialize the RTCC
194 RtccInit();
195 while(RtccGetClkStat()!=RTCC_CLK_ON); // wait for the SOSC to be actually runni
196 // could wait here at most 32ms
197 RtccOpen(0x10073000, 0x07011602, 0);
198
199 while (!MDD_MediaDetect());
200
201 // Initialize the library
202 while (!FSInit());
203
204 #ifndef ALLOW_WRITES
205 // Create a file
206 pointer = wFSfopen ((const unsigned short int * )&str1[0], "w");
207 if (pointer == NULL)
208     while(1);
209
210

```

5.7.4.5.5 Opening a File

This topic describes differences when opening a file.

Description

Opening a file in MLA vs. MPLAB Harmony:

Opening a file remains similar for both MLA and MPLAB Harmony. However, a critical difference in MPLAB Harmony is that, while opening the file, the complete path of the file must be specified. In addition, the path must be preceded by the string "/mnt/". The complete path is required because the MPLAB Harmony FS implements a multi-partition media support. Therefore, the file path should include the name of volume (assigned media partition), and from where the file is to be opened. For more details, please refer to the documentation of the `SYS_FS_FileOpen()` function. The following two figures illustrate opening a file in MLA vs. MPLAB Harmony.

In case of MLA, the error in opening a file was indicated by the file open function returning a NULL, while, in MPLAB harmony, the file open returns `SYS_FS_HANDLE_INVALID`.

```

244 #endif
245
246 // Open file 1 in read mode
247 pointer = wFSfopen ((const unsigned short int * )&str1[0], "r");
248 if (pointer == NULL)
249     while(1);
250
251 if (wFSrename ((const unsigned short int * )&str3[0], pointer))
252     while(1);
253
254 // Read one four-byte object
255 if (FSfread (receiveBuffer, 4, 1, pointer) != 1)
256     while(1);

```

Figure 1: Opening a File in MLA

```

173
174 case APP_OPEN_FILE:
175
176     appData.fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE_TOO_LONG_NAME_EXAMPLE_123.JPG",
177                                         (SYS_FS_FILE_OPEN_READ));
178     if(appData.fileHandle == SYS_FS_HANDLE_INVALID)
179     {
180         /* Could not open the file. Error out*/
181         appData.state = APP_ERROR;
182     }
183     else
184     {
185         appData.fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE_TOO_LONG_NAME_EXAMPLE_123.JPG",
186                                         (SYS_FS_FILE_OPEN_READ));

```

Figure 2: Opening a File in MPLAB Harmony

5.7.4.5.6 Reading a File

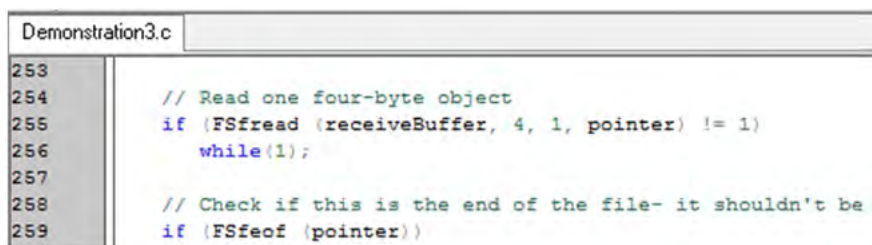
This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while reading a file.

Description

Reading a file in MLA vs. MPLAB Harmony:

Reading a file remains similar for both MLA and MPLAB Harmony, as depicted in the screen shots below. The smaller differences are that, the name of the function to read the file is different, the order of parameters passed are different and MPLAB Harmony only takes 3 parameters. The MPLAB Harmony function only enables byte based access.

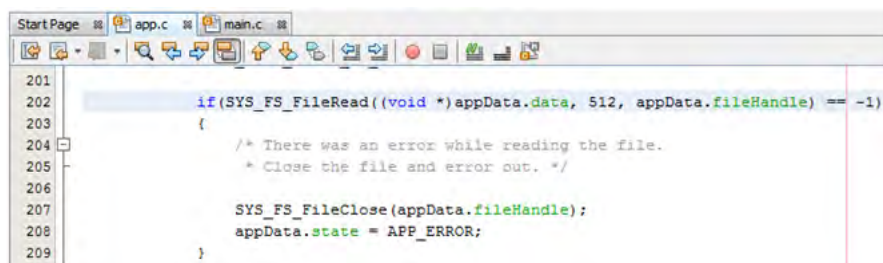
Though, both MLA and MPLAB harmony file read functions returns the number of bytes read, the MPLAB Harmony functions also returns “-1” if there were any errors while reading the file.



The screenshot shows a code editor window titled "Demonstration3.c". The code is as follows:

```
253
254 // Read one four-byte object
255 if (FSfread (receiveBuffer, 4, 1, pointer) != 1)
256     while(1);
257
258 // Check if this is the end of the file- it shouldn't be
259 if (FSfeof (pointer))
```

Figure: Reading a File in MLA



The screenshot shows a code editor window with tabs for "app.c" and "main.c". The code is as follows:

```
201
202 if(SYS_FS_FileRead((void *)appData.data, 512, appData.fileHandle) == -1)
203 {
204     /* There was an error while reading the file.
205      * Close the file and error out. */
206
207     SYS_FS_FileClose(appData.fileHandle);
208     appData.state = APP_ERROR;
209 }
```

Figure: Reading a File in MPLAB Harmony

5.7.4.5.7 Writing a File

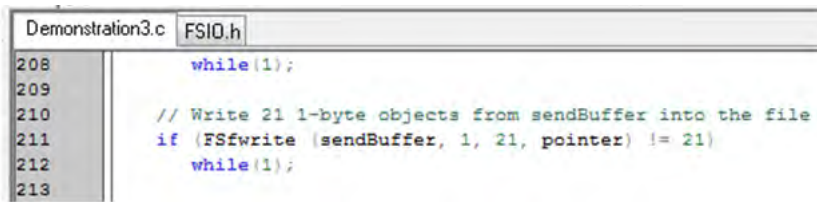
This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while writing to a file.

Description

Writing to a file in MLA vs. MPLAB Harmony:

Writing to a file remains similar for both MLA and MPLAB Harmony, as depicted in the screen shots below. The smaller differences are that, the name of the function to write to the file is different, the order of parameters passed are different and MPLAB Harmony only takes 3 parameters. The MPLAB Harmony function only enables byte based access.

Though, both MLA and MPLAB harmony file read functions returns the number of bytes written, the MPLAB Harmony functions also returns “-1” if there were any errors while writing to the file.

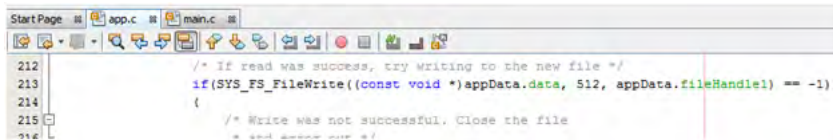


```

208     while(1);
209
210     // Write 21 1-byte objects from sendBuffer into the file
211     if (FSfwrite (sendBuffer, 1, 21, pointer) != 21)
212         while(1);
213

```

Figure: Writing to a File in MLA



```

212     /* If read was success, try writing to the new file */
213     if(SYS_FS_FileWrite((const void *)appData.data, 512, appData.fileHandle1) == -1)
214     {
215         /* Write was not successful. Close the file
216          * and attempt once more */

```

Figure: Writing to a File in MPLAB Harmony

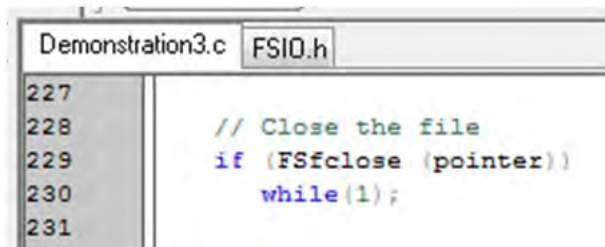
5.7.4.5.8 Closing a File

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while closing a file.

Description

Closing a file in MLA vs. MPLAB Harmony:

Closing a file remains similar for both MLA and MPLAB Harmony, as depicted in the screen shots below. The smaller difference is that, the name of the function is different.

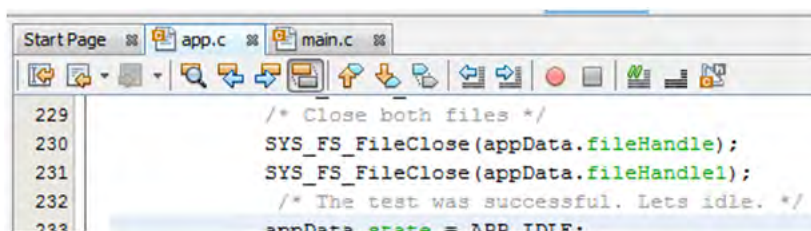


```

227
228     // Close the file
229     if (FSfclose (pointer))
230         while(1);
231

```

Figure: Closing a File in MLA



```

229     /* Close both files */
230     SYS_FS_FileClose (appData.fileHandle);
231     SYS_FS_FileClose (appData.fileHandle1);
232     /* The test was successful. Lets idle. */
233     appData.state = APP_IDLE;

```

Figure: Closing a File in MPLAB Harmony

5.7.4.5.9 File EOF

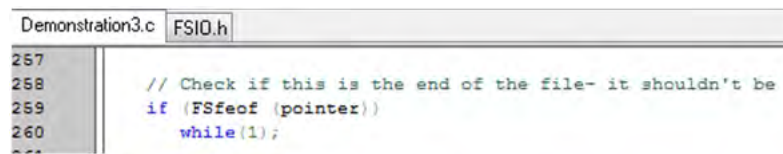
This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while checking for end of file.

Description

Checking for EOF in MLA vs. MPLAB Harmony:

The EOF function remains similar for both MLA and MPLAB Harmony, as depicted in the screen shots below. The smaller

difference is that, the name of the function is different.

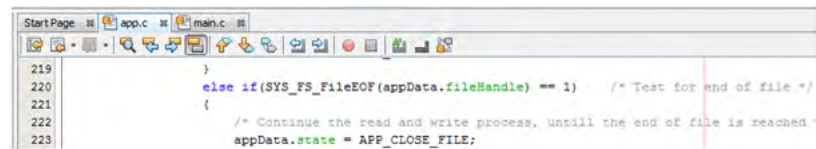


```

Demonstration3.c FSIO.h
257
258 // Check if this is the end of the file- it shouldn't be
259 if (FSfeof (pointer))
260 while(1);

```

Figure: EOF in MLA



```

Start Page app.c main.c
219 }
220 else if(SYS_FS_FileEOF(appData.fileHandle) == 1) /* Test for end of file */
221 {
222 /* Continue the read and write process, untill the end of file is reached */
223 appData.state = APP_CLOSE_FILE;

```

Figure: EOF in MPLAB Harmony

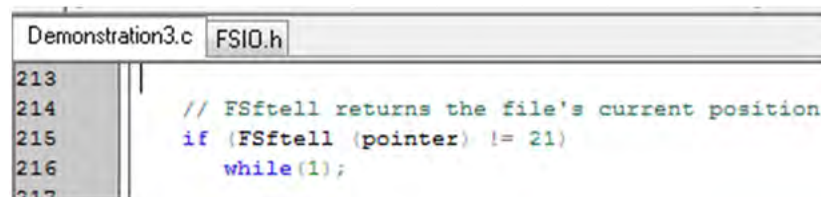
5.7.4.5.10 File Tell

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while checking for file pointer (tell).

Description

Checking for file pointer (tell) in MLA vs. MPLAB Harmony:

The function to check for file pointer (tell) remains similar for both MLA and MPLAB Harmony, as depicted in the screen shots below. The smaller difference is that, the name of the function is different.

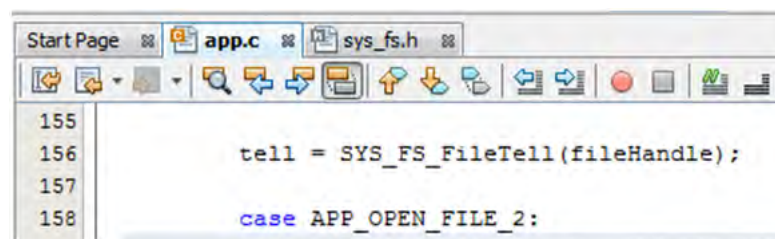


```

Demonstration3.c FSIO.h
213
214 // FSftell returns the file's current position
215 if (FSftell (pointer) != 21)
216 while(1);
217

```

Figure: Tell Function in MLA



```

Start Page app.c sys_fs.h
155
156 tell = SYS_FS_FileTell(fileHandle);
157
158 case APP_OPEN_FILE_2:

```

Figure: Tell Function in MPLAB Harmony

5.7.4.5.11 File Seek

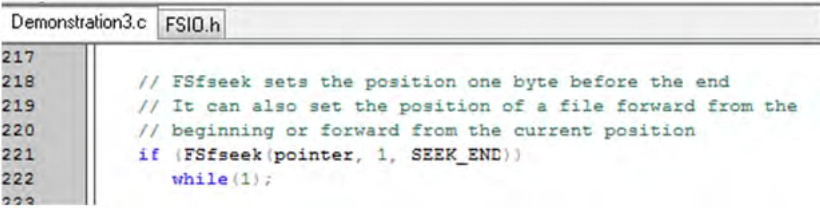
This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while performing a file seek.

Description

File seek in MLA vs. MPLAB Harmony:

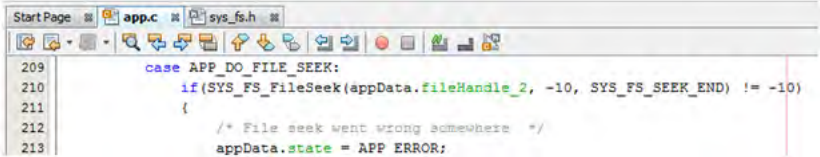
The function to perform file seek remains similar for both MLA and MPLAB Harmony, as depicted in the screen shots below. The

smaller difference is that, the name of the function is different.



```
217
218 // FSfseek sets the position one byte before the end
219 // It can also set the position of a file forward from the
220 // beginning or forward from the current position
221 if (FSfseek(pointer, 1, SEEK_END))
222     while(1);
223
```

Figure: File Seek in MLA



```
209 case APP_DO_FILE_SEEK:
210     if(SYS_FS_FileSeek(appData.fileHandle_2, -10, SYS_FS_SEEK_END) != -10)
211     {
212         /* File seek went wrong somewhere */
213         appData.state = APP_ERROR;
```

Figure: File Seek in MPLAB Harmony

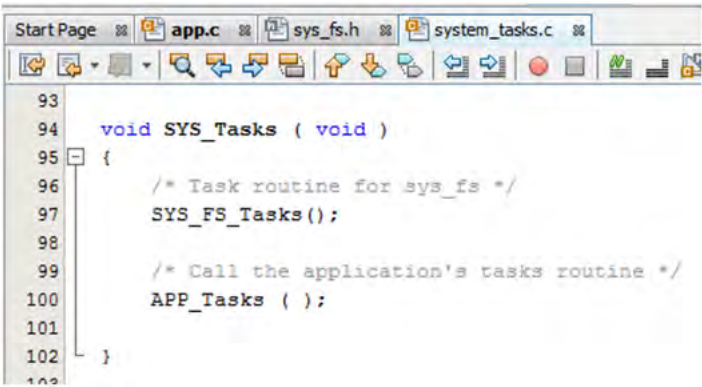
5.7.4.5.12 SYS_FS_Tasks

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS with respect to [SYS_FS_Tasks\(\)](#) function.

Description

[SYS_FS_Tasks\(\)](#) in MLA vs. MPLAB Harmony:

The MPLAB Harmony needs the [SYS_FS_Tasks\(\)](#) function to be running periodically from the [SYS_Tasks\(\)](#) function. The MLA code did not had any such function which had to be running periodically.



```
93
94 void SYS_Tasks ( void )
95 {
96     /* Task routine for sys_fs */
97     SYS_FS_Tasks();
98
99     /* Call the application's tasks routine */
100     APP_Tasks();
101
102 }
```

Figure: Running [SYS_FS_Tasks\(\)](#) Function from [SYS_Tasks\(\)](#)

5.7.4.6 Library Interface

Data Types and Constants

	Name	Description
	SYS_FS_ERROR	Lists the various error cases
	SYS_FS_FILE_OPEN_ATTRIBUTES	Lists the various attributes (modes) in which a file can be opened
	SYS_FS_FILE_SEEK_CONTROL	Lists the various modes of file seek

	<u>SYS_FS_FILE_SYSTEM_TYPE</u>	Enumerated data type identifying native file systems supported.
	<u>SYS_FS_FSTAT</u>	The structure to obtain the status of a file.
	<u>SYS_FS_FUNCTIONS</u>	SYS FS Function signature structure for native file systems
	<u>SYS_FS_HANDLE</u>	This type defines the file handle.
	<u>SYS_FS_REGISTRATION_TABLE</u>	The sys_fs layer has to be initialized by passing this structure with suitably initialized members.
	<u>SYS_FS_RESULT</u>	Lists the various results of a file operation
	<u>FAT_FS_MAX_LFN</u>	Lists the maximum length of file name during LFN selection
	<u>FAT_FS_MAX_SS</u>	Lists the definitions for FAT file system sector size.
	<u>FAT_FS_USE_LFN</u>	Lists the definitions for FAT file system LFN selection
	<u>SYS_FS_HANDLE_INVALID</u>	This definitions defines the SYS FS file invalid handle.

File Operation Functions

	Name	Description
	<u>SYS_FS_FileClose</u>	Close a file.
	<u>SYS_FS_FileEOF</u>	check handle status
	<u>SYS_FS_FileOpen</u>	Open a file
	<u>SYS_FS_FileRead</u>	Read specified bytes from a file
	<u>SYS_FS_FileSeek</u>	Move the file pointer.
	<u>SYS_FS_FileSize</u>	Returns the size of the file
	<u>SYS_FS_FileStat</u>	Get file status
	<u>SYS_FS_FileTell</u>	Obtains the file pointer position
	<u>SYS_FS_FileWrite</u>	Write on the file
	<u>SYS_FS_FileNameGet</u>	Reads the file name.

General Operation Functions

	Name	Description
	<u>SYS_FS_FileError</u>	check the type of error
	<u>SYS_FS_Initialize</u>	Initializes the File system Abstraction Layer (sys_fs layer).
	<u>SYS_FS_Mount</u>	Mount filesystems
	<u>SYS_FS_Tasks</u>	Tasks for the sys_fs layer
	<u>SYS_FS_Unmount</u>	unmount filesystems

Description

This section describes the Application Programming Interface (API) functions of the File System.

5.7.4.6.1 File Operation Functions

5.7.4.6.1.1 SYS_FS_FileClose Function

C

```
SYS_FS_RESULT SYS_FS_FileClose(
    SYS_FS_HANDLE handle
);
```

Description

The SYS_FS_FileClose() function closes an opened file

Preconditions

A valid file handle must be obtained before closing a file.

Parameters

Parameters	Description
handle	A valid handle, which was obtained while opening the file.

Returns

If Success - SYS_FS_RES_SUCCESS

If Failure - SYS_FS_RES_FAILURE

The reason for failure could be retrieved with SYS_FS_Error

Remarks

None

Example

```
SYS_FS_HANDLE fileHandle;  
fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",  
    (SYS_FS_FILE_OPEN_READ));  
  
if(fileHandle != SYS_FS_HANDLE_INVALID)  
{  
    // File open is successful  
}  
...  
...  
  
SYS_FS_FileClose(fileHandle);
```

5.7.4.6.1.2 SYS_FS_FileEOF Function**C**

```
bool SYS_FS_FileEOF(  
    SYS_FS_HANDLE handle  
);
```

Description

Checks whether or not the file position indicator is at the end of the file.

Preconditions

A valid file handle must be obtained before knowing a file eof.

Parameters

Parameters	Description
handle	file handle obtained during file Open.

Returns

If Success -

When file pointer has not reached the end of file - false

When file pointer has reached the end of file - true

If Failure - false

The reason for failure could be retrieved with SYS_FS_Error

Remarks

None

Example

```
SYS_FS_HANDLE fileHandle;
bool eof;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
                             (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

eof = SYS_FS_FileEOF(fileHandle);

if(eof == false)
{
    // could be not EOF or a failure
    // Check the error state using SYS_FS_FileError
}
```

5.7.4.6.1.3 SYS_FS_FileOpen Function

C

```
SYS_FS_HANDLE SYS_FS_FileOpen(
    const char* fname,
    SYS_FS_FILE_OPEN_ATTRIBUTES attributes
);
```

Description

The SYS_FS_FileOpen opens a requested file in a specific mode (attribute).

Preconditions

Prior to opening a file, the name of the volume on which the file resides should be known. Also, that volume should be already mounted. While opening the file, the name of the volume is to be passed along with the file name.

Parameters

Parameters	Description
path	Path to the file along with the volume name. The string of volume and file name has to be preceeded by "/mnt/". Also, the volume name and file name has to be separated by a slash "/".
attributes	Access mode of the file, of the type SYS_FS_FILE_OPEN_ATTRIBUTES

Returns

- If Success - Valid handle will be returned
- If Failure - Returned handle will be [SYS_FS_HANDLE_INVALID](#)
- The reason for failure could be retrieved with SYS_FS_Error

Remarks

None

Example

```
SYS_FS_HANDLE fileHandle;
```



```

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
                             (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}

```

5.7.4.6.1.4 SYS_FS_FileRead Function

C

```

size_t SYS_FS_FileRead(
    void * buf,
    size_t nbyte,
    SYS_FS_HANDLE handle
);

```

Description

The SYS_FS_FileRead() function shall attempt to read nbyte bytes from the file associated with the file handle into the buffer pointed to by buf.

Preconditions

A valid file handle must be obtained before reading a file.

Parameters

Parameters	Description
handle	File handle obtained during file open.
buf	Pointer to buffer into which data is read.
nbyte	Number of bytes to be read

Returns

If Success - The number of bytes successfully read (0 or positive number)

If Failure - (-1)

The reason for failure could be retrieved with SYS_FS_Error

Remarks

None

Example

```

...
char buf[20];
size_t nbytes;
size_t bytes_read;
SYS_FS_HANDLE fd;
...
nbytes = sizeof(buf);
bytes_read = SYS_FS_FileRead(buf, nbytes, fd);
...

```

5.7.4.6.1.5 SYS_FS_FileSeek Function

C

```

int32_t SYS_FS_FileSeek(
    SYS_FS_HANDLE fildes,
    int32_t offset,
    SYS_FS_FILE_SEEK_CONTROL whence

```

```
);
```

Description

The SYS_FS_FileSeek() function shall set the file pointer for a open file associated with the file handle, as follows:

If whence is SYS_FS_SEEK_SET, the file offset shall be set to offset bytes from the begining.

If whence is SYS_FS_SEEK_CUR, the file offset shall be set to its current location plus offset.

If whence is SYS_FS_SEEK_END, the file offset shall be set to the size of the file plus offset. If SYS_FS_SEEK_END is selected, then offset has to be a negative number, inorder for the file pointer to be valid.

Trying to move the file pointer using SYS_FS_FileSeek, beyond the range of file will only cause the pointer to be moved to the last location of the file.

Preconditions

A valid file handle must be obtained before seeking a file.

Parameters

Parameters	Description
handle	A valid file handle obtained during file open.
offset	The number of bytes which act as file offset. This value could be a positive or negative value.
whence	File seek control input of type SYS_FS_FILE_SEEK_CONTROL .

Returns

If Success - The number of bytes by which file pointer is moved (0 or positive number)

If Failure - (-1) If the choosen offset value was (-1), then the success or failure can be ascertained with SYS_FS_Error.

The reason for failure could be retrieved with SYS_FS_Error

Remarks

None

Example

```
SYS_FS_HANDLE fileHandle;
int status;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
    (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

status = SYS_FS_FileSeek(fileHandle, 5, SYS_FS_SEEK_CUR);

if((status != -1) && (status == 5))
{
    // Success
}
```

5.7.4.6.1.6 SYS_FS_FileSize Function

C

```
int32_t SYS_FS_FileSize(
```

```
    SYS_FS_HANDLE handle  
);
```

Description

Returns the size of the file as pointed by the handle.

Preconditions

A valid file handle must be obtained before knowing a file size.

Parameters

Parameters	Description
handle	File handle obtained during file Open.

Returns

If Success - file size

If Failure - (-1)

The reason for failure could be retrieved with SYS_FS_Error

Remarks

None

Example

```
SYS_FS_HANDLE fileHandle;  
long fileSize;  
  
fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",  
    (SYS_FS_FILE_OPEN_READ));  
  
if(fileHandle != SYS_FS_HANDLE_INVALID)  
{  
    // File open is successful  
}  
...  
...  
  
fileSize = SYS_FS_FileSize(fileHandle);  
  
if(fileSize != -1)  
{  
    // Success  
}
```

5.7.4.6.1.7 SYS_FS_FileStat Function**C**

```
SYS_FS_RESULT SYS_FS_FileStat(  
    const char* fname,  
    SYS_FS_FSTAT * buf  
);
```

Description

The SYS_FS_FileStat() function shall obtain information about a file associated with the file name, and shall write it to the structure pointed to by buf.

The buf argument is a pointer to a [SYS_FS_FSTAT](#) structure, into which information is placed concerning the file.

This function can read the status of file irrespective of a file is opened or not.

Preconditions

Prior to opening a file, the name of the volume on which the file resides should be known. Also, that volume should be already mounted. While opening the file, the name of the volume is to be passed along with the file name.

Parameters

Parameters	Description
path	Path to the file along with the volume name. The string of volume and file name has to be preceded by "/mnt/". Also, the volume name and file name has to be separated by a slash "/".
buf	pointer to SYS_FS_FSTAT structure.

Returns

If Success - `SYS_FS_RES_SUCCESS`

If Failure - `SYS_FS_RES_FAILURE`

The reason for failure could be retrieved with `SYS_FS_Error`

Remarks

None

Example

```
SYS_FS_fStat fileStat;

if(SYS_FS_FileStat("/mnt/myDrive/FILE.TXT", &fileStat) == SYS_FS_RES_SUCCESS)
{
    // Successfully read the status of file "FILE.TXT"
}
```

5.7.4.6.1.8 SYS_FS_FileTell Function**C**

```
int32_t SYS_FS_FileTell(
    SYS_FS_HANDLE handle
);
```

Description

Obtains the current value of the file position indicator for the file, pointed to by handle.

Preconditions

A valid file handle must be obtained before performing a file tell.

Parameters

Parameters	Description
handle	File handle obtained during file Open.

Returns

If Success - current file position

If Failure - (-1)

The reason for failure could be retrieved with `SYS_FS_Error`

Remarks

None

Example

```
SYS_FS_HANDLE fileHandle;
int32_t tell;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
                             (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

tell = SYS_FS_FileTell(fileHandle);

if(tell != -1)
{
    // Success
}
```

5.7.4.6.1.9 SYS_FS_FileWrite Function

C

```
size_t SYS_FS_FileWrite(
    const void * buf,
    size_t nbyte,
    SYS_FS_HANDLE handle
);
```

Description

The SYS_FS_FileWrite() function shall attempt to write nbyte bytes from the buffer pointed to by buf to the file associated with the file handle.

Preconditions

A valid file handle must be obtained before writing a file.

Parameters

Parameters	Description
handle	File handle obtained during file open.
buf	Pointer to buffer from which data is to be written
nbyte	Number of bytes to be written

Returns

- If Success - The number of bytes successfully written (0 or positive number)
 - If Failure - (-1)
- The reason for failure could be retrieved with SYS_FS_Error

Remarks

None

Example

```
...
```

```
const char *buf = "Hello World";
size_t nbytes;
size_t bytes_written;
SYS_FS_HANDLE fd;
...

bytes_written = SYS_FS_FileWrite((const void *)buf, nbytes, fd);
...
```

5.7.4.6.1.10 SYS_FS_FileNameGet Function

C

```
bool SYS_FS_FileNameGet(
    SYS_FS_HANDLE handle,
    uint8_t* cName,
    uint16_t wLen
);
```

Description

Reads the file name of a file that is already open.

Preconditions

The file handle referenced by handle is already open.

Parameters

Parameters	Description
handle	file handle obtained during file Open.
cName	where to store the name of the file.
wLen	the maximum length of data to store in cName.

Returns

If Success -

The file name was successfully located - true

If Failure The file handle provided is not currently open - false

The reason for failure could be retrieved with SYS_FS_Error

Remarks

None

Example

```
SYS_FS_HANDLE fileHandle;
bool stat;
uint8_t fileName[255];

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
    (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

stat = SYS_FS_FileNameGet(fileHandle, fileName, 8 );

if(stat == false)
{
```

```
    // file not located based on handle passed  
    // Check the error state using SYS_FS_FileError  
}
```

5.7.4.6.2 General Operation Functions

5.7.4.6.2.1 SYS_FS_FileError Function

C

```
SYS_FS_ERROR SYS_FS_FileError();
```

Description

When a file system operation fails, the application can know the exact reason of failure by calling the SYS_FS_FileError.

Preconditions

This function has to be called immediately after a failure is observed while doing a file operation. Any subsequent failure will overwrite the cause of previous failure.

Returns

Error code of type [SYS_FS_ERROR](#)

Remarks

None

Example

```
SYS_FS_HANDLE fileHandle;  
SYS_FS_ERROR err;  
  
fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",  
    (SYS_FS_FILE_OPEN_READ));  
  
if(fileHandle == SYS_FS_HANDLE_INVALID)  
{  
    // If failure, now know the specific reason for failure  
    err = SYS_FS_FileError();  
}
```

5.7.4.6.2.2 SYS_FS_Initialize Function

C

```
SYS_FS_RESULT SYS_FS_Initialize(  
    const void* initData  
);
```

Description

Initializes the abstraction layer (sys_fs layer) and sets up the necessary parameters.

Preconditions

This is the first function to be called during usage of sys_fs. Calling other functions from sys_fs without initializing the sys_fs will cause un-predictable behavior.

Parameters

Parameters	Description
initData	The pointer to array of the type SYS_FS_REGISTRATION_TABLE , but type casted to (const void *). The number of elements of array is decided by the definition SYS_FS_MAX_FILE_SYSTEM_TYPE. If the application uses 1 file system (say only FAT FS), then SYS_FS_MAX_FILE_SYSTEM_TYPE is defined to be 1. Otherwise, if the application uses 2 file systems (say FAT FS and MPFS2), then SYS_FS_MAX_FILE_SYSTEM_TYPE is defined to be 2. The SYS_FS_MAX_FILE_SYSTEM_TYPE has to be defined in system_config.h file.

Returns

If Success - SYS_FS_RES_SUCCESS

If Failure - SYS_FS_RES_FAILURE

The reason for failure could be retrieved with SYS_FS_Error

Example

```
// This code snippet shows an example of how the
// SYS FS is initialized

// Only 1 file system is used

#define SYS_FS_MAX_FILE_SYSTEM_TYPE          1

// Functions pointer table for FAT FS

const SYS_FS_FUNCTIONS FatFsFunctions =
{
    .mount    = f_mount,
    .unmount  = f_unmount,
    .open     = f_open,
    .read     = f_read,
    .write    = f_write,
    .close    = f_close,
    .seek     = f_lseek,
    .tell     = f_tell,
    .eof      = f_eof,
    .size     = f_size,
    .fstat    = f_stat,
};

const SYS_FS_REGISTRATION_TABLE sysFSInit [ SYS_FS_MAX_FILE_SYSTEM_TYPE ] =
{
    {
        .nativeFileSystemType = FAT,
        .nativeFileSystemFunctions = &FatFsFunctions
    }
};

SYS_FS_Initialize( (const void *) sysFSInit );
```

5.7.4.6.2.3 SYS_FS_Mount Function**C**

```
SYS_FS_RESULT SYS_FS_Mount(
    const char * devName,
    const char * mountName,
    SYS_FS_FILE_SYSTEM_TYPE filesystemtype,
    unsigned long mountflags,
    const void * data
);
```


Description

The mount command attaches the filesystem specified to a volume.

When a media (say, SD card or USB thumb drive) is attached to the system, the [SYS_FS_Tasks\(\)](#) function needs to run atleast 8 times (for each partition), from the [SYS_Tasks\(\)](#), before the volume names are assigned to each partitions. Hence it is mandatory that the [SYS_FS_Tasks\(\)](#) is run periodically. Also, it is mandatory that the [SYS_FS_Mount\(\)](#) function call from the application code, is not blocking. The application code has to allow the [SYS_FS_Tasks\(\)](#) to run periodically while calling the [SYS_FS_Mount](#) function. If the [SYS_FS_Mount\(\)](#) is called in a blocking mode, then the [SYS_Tasks\(\)](#) never gets a chance to run and hence, the media will not be analyzed and finally, the [SYS_FS_Mount](#) will never succeed. This will result in a dead-lock.

There is no mechanism available for the application to know if the specified volume (devName) is really attached or not. The only available possibility is to keep trying to mount the volume (with the devname), untill success is achieved. It is prudent that the application code implements a time out mechanism while trying to mount a volume (by calling [SYS_FS_Mount](#)). The trial for mount should continue atleast 10 times before assuming that the mount will never succeeded. This has to be done for every new volume to be mounted.

The name standard of volume (devName) used in Harmony file system is as below: -

For NVM - "nvm" "media number" "volume number"

For SD card - "mmcblk" "media number" "volume number"

For MSD - "sd" "media number" "volume number"

Where, "media number" - a, b, c... depending upon number of media of certain type connected.

Where, "volume number" - 1, 2, 3... depending upon number of partitions in that media.

The convention for assigning names to volumes is given by an example: -

If an SD card (with 4 partitions) is attached to the system, and assuming all 4 partitions are recognized, then there will be 4 devNames -- mmcblk1, mmcblk2, mmcblk3 and mmcblk4.

Subsequently, a NVM media is attached which has only 1 partition, then the devname will be -- nvma1.

Later, another SD card is attached to the system which has 1 partition, then the devname will be -- mmcblk1.

Finally, there will be 6 volume names (or devNames), which is available for the application to be mounted and used for file system.

Preconditions

The "devName" name for the volume has to be known. The file sytem type with which each of the volumes are formatted has to be known. Trying to mount a volume with a file system which is different from what the volume is actually formatted, will cause mounting failure.

Parameters

Parameters	Description
devName	The device name (name of volume) which needs to be mounted. The devName has to be preceeded by the string "/dev/".
mountName	Mount name for the device to be mounted. This is a name provided by the user. In future, while accessing the mounted volume (say, during SYS_FS_FileOpen operation), the mountName is used to refer the path for file. The mount name has to be preceeded by the string "/mnt/".
filesystemtype	native file system of SYS_FS_FILE_SYSTEM_TYPE type.
mountflags	mounting control flags. This parameter is reserved for future enhancements. Hence, always pass zero.

data	The data argument is interpreted by the different file systems. This parameter is reserved for future enhancements. Hence, always pass NULL.

Returns

If Success - SYS_FS_RES_SUCCESS

If Failure - SYS_FS_RES_FAILURE

The reason for failure could be retrieved with SYS_FS_Error

Example

```

switch(appState)
{
    case TRY_MOUNT:
        if(SYS_FS_Mount("/dev/mmcblk1", "/mnt/myDrive", FAT, 0, NULL) !=
SYS_FS_RES_SUCCESS)
        {
            // Failure, hence try mounting again
        }
        else
        {
            // Mount was successful. Do further file operations
            appState = DO_FURTHER_STUFFS;
        }
        break;

```

5.7.4.6.2.4 SYS_FS_Tasks Function**C**

```
void SYS_FS_Tasks();
```

Description

This routine is used to run the various tasks and functionalities of sys_fs layer.

Preconditions

The [SYS_FS_Initialize](#) routine must have been called before running the tasks.

Returns

None

Remarks

This routine is not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)).

Example

```

void SYS_Tasks ( void )
{
    SYS_FS_Tasks ();

    // Do other tasks
}

```

5.7.4.6.2.5 SYS_FS_Unmount Function**C**

```

SYS_FS_RESULT SYS_FS_Unmount(
    const char * mountName
);

```

Description

remove the attachment of the volume from the fil system

Preconditions

The volume name has to be know in order to pass as input to Unmount. The specified volume name to be unmounted should have been already mounted.

Parameters

Parameters	Description
mountName	Mount name for the volume to be mounted. The mount name has to be preceeded by the string "/mnt/".

Returns

If Success - SYS_FS_RES_SUCCESS

If Failure - SYS_FS_RES_FAILURE

The reason for failure could be retrieved with SYS_FS_Error

Example

```

if(SYS_FS_Unmount("/mnt/myDrive") != SYS_FS_RES_SUCCESS)
{
    // Failure, hence try unmouting again
}
else
{
    // Unmount was successful.
}

```

5.7.4.6.3 Data Types and Constants

5.7.4.6.3.1 SYS_FS_ERROR Enumeration

C

```

typedef enum {
    SYS_FS_ERROR_OK = 0,
    SYS_FS_ERROR_DISK_ERR,
    SYS_FS_ERROR_INT_ERR,
    SYS_FS_ERROR_NOT_READY,
    SYS_FS_ERROR_NO_FILE,
    SYS_FS_ERROR_NO_PATH,
    SYS_FS_ERROR_INVALID_NAME,
    SYS_FS_ERROR_DENIED,
    SYS_FS_ERROR_EXIST,
    SYS_FS_ERROR_INVALID_OBJECT,
    SYS_FS_ERROR_WRITE_PROTECTED,
    SYS_FS_ERROR_INVALID_DRIVE,
    SYS_FS_ERROR_NOT_ENABLED,
    SYS_FS_ERROR_NO_FILESYSTEM,
    SYS_FS_ERROR_FORMAT_ABORTED,
    SYS_FS_ERROR_TIMEOUT,
    SYS_FS_ERROR_LOCKED,
    SYS_FS_ERROR_NOT_ENOUGH_CORE,
    SYS_FS_ERROR_TOO_MANY_OPEN_FILES,
    SYS_FS_ERROR_INVALID_PARAMETER,
    SYS_FS_ERROR_NOT_ENOUGH_FREE_VOLUME,
    SYS_FS_ERROR_FS_NOT_SUPPORTED,
    SYS_FS_ERROR_FS_NOT_MATCH_WITH_VOLUME,
    SYS_FS_ERROR_NOT_SUPPORTED_IN_NATIVE_FS
}

```

```
} SYS_FS_ERROR;
```

Description

File Error error enumeration

This enumeration lists the various error cases. When the application calls for a file system function which has a return type of [SYS_FS_RESULT](#) and if the return value is `SYS_FS_RES_FAILURE`, the application can know the specific reason for failure by calling the [SYS_FS_FileError](#) function. The return value of [SYS_FS_FileError](#) function will be one of the enumeration of the type `SYS_FS_ERROR`.

Members

Members	Description
<code>SYS_FS_ERROR_OK = 0</code>	Success
<code>SYS_FS_ERROR_DISK_ERR</code>	(1) A hard error occurred in the low level disk I/O layer
<code>SYS_FS_ERROR_INT_ERR</code>	(2) Assertion failed
<code>SYS_FS_ERROR_NOT_READY</code>	(3) The physical drive cannot work
<code>SYS_FS_ERROR_NO_FILE</code>	(4) Could not find the file
<code>SYS_FS_ERROR_NO_PATH</code>	(5) Could not find the path
<code>SYS_FS_ERROR_INVALID_NAME</code>	(6) The path name format is invalid
<code>SYS_FS_ERROR_DENIED</code>	(7) Access denied due to prohibited access or directory full
<code>SYS_FS_ERROR_EXIST</code>	(8) Access denied due to prohibited access
<code>SYS_FS_ERROR_INVALID_OBJECT</code>	(9) The file/directory object is invalid
<code>SYS_FS_ERROR_WRITE_PROTECTED</code>	(10) The physical drive is write protected
<code>SYS_FS_ERROR_INVALID_DRIVE</code>	(11) The logical drive number is invalid
<code>SYS_FS_ERROR_NOT_ENABLED</code>	(12) The volume has no work area
<code>SYS_FS_ERROR_NO_FILESYSTEM</code>	(13) There is no valid volume
<code>SYS_FS_ERROR_FORMAT_ABORTED</code>	(14) The Format() aborted due to any parameter error
<code>SYS_FS_ERROR_TIMEOUT</code>	(15) Could not get a grant to access the volume within defined period
<code>SYS_FS_ERROR_LOCKED</code>	(16) The operation is rejected according to the file sharing policy
<code>SYS_FS_ERROR_NOT_ENOUGH_CORE</code>	(17) LFN working buffer could not be allocated
<code>SYS_FS_ERROR_TOO_MANY_OPEN_FILES</code>	(18) Number of open files
<code>SYS_FS_ERROR_INVALID_PARAMETER</code>	(19) Given parameter is invalid
<code>SYS_FS_ERROR_NOT_ENOUGH_FREE_VOLUME</code>	(20) Too many mounts requested. Not enough free volume available
<code>SYS_FS_ERROR_FS_NOT_SUPPORTED</code>	(21) Requested native file system is not supported
<code>SYS_FS_ERROR_FS_NOT_MATCH_WITH_VOLUME</code>	(22) Requested native file system does not match the format of volume
<code>SYS_FS_ERROR_NOT_SUPPORTED_IN_NATIVE_FS</code>	(23) Function not supported in native file system layer

5.7.4.6.3.2 SYS_FS_FILE_OPEN_ATTRIBUTES Enumeration

C

```
typedef enum {
    SYS_FS_FILE_OPEN_READ = 0,
    SYS_FS_FILE_OPEN_WRITE,
    SYS_FS_FILE_OPEN_APPEND,
    SYS_FS_FILE_OPEN_READ_PLUS,
    SYS_FS_FILE_OPEN_WRITE_PLUS,
    SYS_FS_FILE_OPEN_APPEND_PLUS
} SYS_FS_FILE_OPEN_ATTRIBUTES;
```

Description

File open attributes

Lists the various attributes (modes) in which a file can be opened

Members

Members	Description
SYS_FS_FILE_OPEN_READ = 0	reading the file = possible, if file exists. reading the file = file open returns error, if file does not exist. writing to the file = not possible. Write operation returns error
SYS_FS_FILE_OPEN_WRITE	reading the file = not possible. Read operation returns error writing to the file = possible. If file exists, write happens from the beginning of the file, overwriting the existing content of the file. writing to the file = If file does not exist, a new file will be created and data will be written into the newly created file.
SYS_FS_FILE_OPEN_APPEND	reading the file = not possible. Read operation returns error writing to the file = possible. If file exists, write happens from the end of the file, preserving the existing content of the file. writing to the file = If file does not exist, a new file will be created and data will be written into the newly created file.
SYS_FS_FILE_OPEN_READ_PLUS	reading the file = possible, if file exists. reading the file = file open returns error, if file does not exist. writing to the file = possible, if file exists, starting from the beginning of the file (overwriting). writing to the file = file open returns error, if file does not exist.
SYS_FS_FILE_OPEN_WRITE_PLUS	reading the file = possible, if file exists. reading the file = If file does not exist, a new file will be created. writing to the file = possible. If file exists, write happens from the beginning of the file, overwriting the existing content of the file. writing to the file = If file does not exist, a new file will be created and data will be written into the newly created file.
SYS_FS_FILE_OPEN_APPEND_PLUS	reading the file = possible, if file exists. reading the file = If file does not exist, a new file will be created. writing to the file = possible. If file exists, write happens from the end of the file, preserving the existing content of the file. writing to the file = If file does not exist, a new file will be created and data will be written into the newly created file.

5.7.4.6.3.3 SYS_FS_FILE_SEEK_CONTROL Enumeration

C

```
typedef enum {
    SYS_FS_SEEK_SET,
    SYS_FS_SEEK_CUR,
    SYS_FS_SEEK_END
} SYS_FS_FILE_SEEK_CONTROL;
```

Description

File Seek control

This enumeration lists the various modes of file seek. When the application calls the [SYS_FS_FileSeek](#) function, it specifies the kind of seek that needs to be performed. The enumeration lists the various modes of file seek.

Members

Members	Description
SYS_FS_SEEK_SET	The file offset shall be set to input number of bytes from the start
SYS_FS_SEEK_CUR	The file offset shall be set to its current location plus input number of bytes

SYS_FS_SEEK_END	The file offset shall be set to the size of the file plus input number of bytes

Remarks

None.

5.7.4.6.3.4 SYS_FS_FILE_SYSTEM_TYPE Enumeration

C

```
typedef enum {
    FAT,
    MPFS2
} SYS_FS_FILE_SYSTEM_TYPE;
```

Description

File System type

These enumerated values are the possible native file system that can be supported by the SYS FS layer.

Members

Members	Description
FAT	FAT FS native File system
MPFS2	MPFS2 native File system

Remarks

None.

5.7.4.6.3.5 SYS_FS_FSTAT Structure

C

```
typedef struct {
    uint32_t fsize;
    uint16_t fdate;
    uint16_t ftime;
    uint8_t fattrib;
    char fname[13];
    char* lfname;
    uint32_t lfsize;
} SYS_FS_FSTAT;
```

Description

SYS FS File status structure

This structure holds the various status of a file. The structure is passed when SYS_FS_Stat function is called and after a successful execution of the function, the members of this structure carries the file status.

Members

Members	Description
uint32_t fsize;	File size
uint16_t fdate;	Last modified date
uint16_t ftime;	Last modified time
uint8_t fattrib;	Attribute
char fname[13];	Short file name (8.3 format)
char* lfname;	Pointer to the LFN buffer
uint32_t lfsize;	Size of LFN buffer in TCHAR

Remarks

None.

5.7.4.6.3.6 SYS_FS_FUNCTIONS Structure**C**

```
typedef struct {
    int (* mount)(uint8_t vol);
    int (* unmount)(uint8_t vol);
    int (* open)(uintptr_t handle, const char* path, uint8_t mode);
    int (* read)(uintptr_t fp, void* buff, uint32_t btr, uint32_t *br);
    int (* write)(uintptr_t fp, const void* buff, uint32_t btw, uint32_t* bw);
    int (* close)(uintptr_t fp);
    int (* seek)(uintptr_t handle, uint32_t offset);
    uint32_t (* tell)(uintptr_t handle);
    bool (* eof)(uintptr_t handle);
    uint32_t (* size)(uintptr_t handle);
    int (* fstat)(const char* path, uintptr_t fno);
} SYS_FS_FUNCTIONS;
```

Description

SYS FS Function signature structure for native file systems

The SYS FS layer supports functions from each native file system layer. This structure specifies the signature for each function from native file system (parameter that needs to be passed to each function and return type for each function). If a new native file system is to be integrated with the SYS FS layer, the functions should follow the signature.

The structure of function pointer for 2 native file system -- FAT FS and MPFS2 is already provided in the respective source files for the native file system. Hence the following structure is not immediately useful for the user. But the explanation for the structure is still provided for advanced users who would wish to integrate a new native file system to the Harmony File system framework.

Members

Members	Description
int (* mount)(uint8_t vol);	Function pointer of native file system for mounting a volume
int (* unmount)(uint8_t vol);	Function pointer of native file system for unmounting a volume
int (* open)(uintptr_t handle, const char* path, uint8_t mode);	Function pointer of native file system for opening a file
int (* read)(uintptr_t fp, void* buff, uint32_t btr, uint32_t *br);	Function pointer of native file system for reading a file
int (* write)(uintptr_t fp, const void* buff, uint32_t btw, uint32_t* bw);	Function pointer of native file system for writing to a file
int (* close)(uintptr_t fp);	Function pointer of native file system for closing a file
int (* seek)(uintptr_t handle, uint32_t offset);	Function pointer of native file system for moving the file pointer by a desired offset
uint32_t (* tell)(uintptr_t handle);	Function pointer of native file system for finding the position of the file pointer
bool (* eof)(uintptr_t handle);	Function pointer of native file system to check if the end of file is reached
uint32_t (* size)(uintptr_t handle);	Function pointer of native file system to know the size of file
int (* fstat)(const char* path, uintptr_t fno);	Function pointer of native file system to know the status of file

Remarks

None.

5.7.4.6.3.7 SYS_FS_HANDLE Type

C

```
typedef uintptr_t SYS_FS_HANDLE;
```

Description

SYS FS File Handle

While the application code opens a file, a file handle is returned by the open function (on successful open). This type defines the type of file handle.

Remarks

None.

5.7.4.6.3.8 SYS_FS_REGISTRATION_TABLE Structure

C

```
typedef struct {  
    SYS_FS_FILE_SYSTEM_TYPE nativeFileSystemType;  
    const SYS_FS_FUNCTIONS * nativeFileSystemFunctions;  
} SYS_FS_REGISTRATION_TABLE;
```

Description

SYS_FS_REGISTRATION_TABLE structure

When the SYS FS layer is initialised, it has to know the type of native file system it has to support and the list of functions for native file system. The members of this structure can be initialized with suitable values and then passed on to [SYS_FS_Initialize](#) initialization function. Please refer to the example code provided for [SYS_FS_Initialize](#).

Members

Members	Description
SYS_FS_FILE_SYSTEM_TYPE nativeFileSystemType;	Native file system of type SYS_FS_FILE_SYSTEM_TYPE
const SYS_FS_FUNCTIONS * nativeFileSystemFunctions;	Pointer to the structure of type SYS_FS_FUNCTIONS which has the list of

Remarks

None.

5.7.4.6.3.9 SYS_FS_RESULT Enumeration

C

```
typedef enum {  
    SYS_FS_RES_SUCCESS = 0,  
    SYS_FS_RES_FAILURE = -1  
} SYS_FS_RESULT;
```

Description

File operation result enum

This enumeration lists the various results of a file operation. When a file operation function is called from the application, and if the return type of the function is SYS_FS_RESULT, then, the enumeration below specifies the value of function return.

Members

Members	Description
SYS_FS_RES_SUCCESS = 0	Operation successful
SYS_FS_RES_FAILURE = -1	operation failed

5.7.4.6.3.10 FAT_FS_MAX_LFN Macro**C**

```
#define FAT_FS_MAX_LFN 255
```

Description

FAT File System LFN (long file name) max length

Set the value to 255

5.7.4.6.3.11 FAT_FS_MAX_SS Macro**C**

```
#define FAT_FS_MAX_SS 512
```

Description

FAT File System Sector size

Maximum sector size to be handled. Always set the value of sector size to 512

5.7.4.6.3.12 FAT_FS_USE_LFN Macro**C**

```
#define FAT_FS_USE_LFN 1
```

Description

FAT File System LFN (long file name) selection

The FAT_FS_USE_LFN option switches the LFN support. Set the value to 1

5.7.4.6.3.13 SYS_FS_HANDLE_INVALID Macro**C**

```
#define SYS_FS_HANDLE_INVALID ((SYS_FS_HANDLE)(-1))
```

Description

SYS FS File Invalid Handle

While the application code opens a file, if the file open is unsuccessful, an invalid file handle is returned. This definition defines the type of invalid file handle.

Remarks

None.

5.7.4.7 Files

Files

Name	Description
sys_fs.h	This file contains function and type declarations required to interact with the Harmony File System System Framework.

Description

5.7.4.7.1 sys_fs.h

File System System-Library Interface Declarations and types.

This file contains function and type declarations required to interact with the Harmony File System System Framework.

Enumerations

	Name	Description
	SYS_FS_ERROR	Lists the various error cases
	SYS_FS_FILE_OPEN_ATTRIBUTES	Lists the various attributes (modes) in which a file can be opened
	SYS_FS_FILE_SEEK_CONTROL	Lists the various modes of file seek
	SYS_FS_FILE_SYSTEM_TYPE	Enumerated data type identifying native file systems supported.
	SYS_FS_RESULT	Lists the various results of a file operation

Functions

	Name	Description
	SYS_FS_FileClose	Close a file.
	SYS_FS_FileEOF	check handle status
	SYS_FS_FileError	check the type of error
	SYS_FS_FileNameGet	Reads the file name.
	SYS_FS_FileOpen	Open a file
	SYS_FS_FileRead	Read specified bytes from a file
	SYS_FS_FileSeek	Move the file pointer.
	SYS_FS_FileSize	Returns the size of the file
	SYS_FS_FileStat	Get file status
	SYS_FS_FileTell	Obtains the file pointer position
	SYS_FS_FileWrite	Write on the file
	SYS_FS_Initialize	Initializes the File system Abstraction Layer (sys_fs layer).
	SYS_FS_Mount	Mount filesystems
	SYS_FS_Tasks	Tasks for the sys_fs layer
	SYS_FS_Unmount	unmount filesystems

Macros

	Name	Description
	FAT_FS_MAX_LFN	Lists the maximum length of file name during LFN selection
	FAT_FS_MAX_SS	Lists the definitions for FAT file system sector size.
	FAT_FS_USE_LFN	Lists the definitions for FAT file system LFN selection

	SYS_FS_HANDLE_INVALID	This definitions defines the SYS FS file invalid handle.

Structures

	Name	Description
	SYS_FS_FSTAT	The structure to obtain the status of a file.
	SYS_FS_FUNCTIONS	SYS FS Function signature structure for native file systems
	SYS_FS_REGISTRATION_TABLE	The sys_fs layer has to be initialized by passing this structure with suitably initialized members.

Types

	Name	Description
	SYS_FS_HANDLE	This type defines the file handle.

File Name

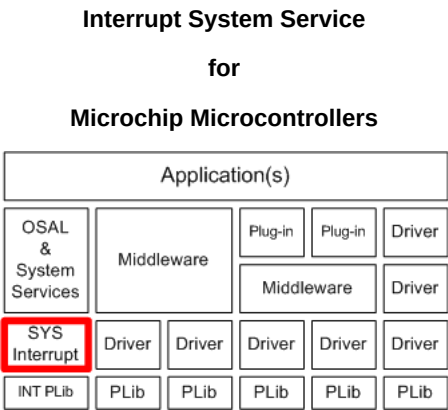
sys_fs.h

Company

Microchip Technology Incorporated

5.7.5 Interrupt System Service Library

5.7.5.1 Introduction



This library implements the Interrupt System Service. It is part of the system services that provides support for processing interrupts. The Interrupt System provides support for initializing the processor's interrupt controller, registering Interrupt Service Routines (ISRs) and managing interrupts. These features enable making efficient and dynamic applications, drivers, and middleware that respond to external events as they occur in real time.

5.7.5.2 Release Notes

MPLAB Harmony Version: v0.70b Interrupt System Service Library Version : 1.0 Release Date: 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.7.5.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.7.5.4 Using the Library

This topic describes the basic architecture of the Interrupt System Service Library and provides information and examples on how to use it.

Interface Header File: [sys_int.h](#)

The interface to the Interrupt System Service library is defined in the "[sys_int.h](#)" header file, which is included by the "sys.h" system service header file. Any C language source (.c) file that uses the Interrupt System Service library should include "sys.h".

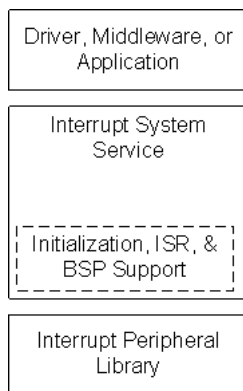
Library File:

The Interrupt System Service library (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the library interacts with the framework.

5.7.5.4.1 Abstraction Model

This library provides an abstraction of the interrupt subsystem that is used by device drivers, middleware libraries and applications to receive and control interrupts in real time.



Interrupt System Service

The interrupt system services provide support for initializing the processor's interrupt controller, managing Interrupt Service Routines (ISRs) and managing interrupts.

Initialization

Each software module (device driver, library, or application) that needs to receive an interrupt must enable that interrupt itself. This is normally done in the module's initialization routine which is called by the "[SYS_Initialize](#)" service.

A module that intends to use an interrupt must first register the "Tasks" function that is to be called when the desired source causes an interrupt. Then, it must enable that source, once it is ready to start receiving interrupts.

If the interrupt system service is configured for static usage, then the routine that dynamically registers the "Tasks" function will be nulled out by a macro (generating no run-time code) and, instead, the "Tasks" routine must be called statically from the function that implements the raw ISR vector. How this is done is different for each processor family, as explained below.

Interrupt Service Routine (ISR)

Each software module (device driver, library, or application) that needs to receive an interrupt must implement a "Tasks" routine to handle that interrupt. In order for the module to operate in an interrupt-driven mode, the "Tasks" routine must be called from within the appropriate "raw" Interrupt Service Routine (ISR).

How the raw ISR is implemented is highly dependent upon the specific processor being used. Libraries are available that implement raw ISRs for each processor family in a way that allows dynamic registration and de-registration of "Tasks" routines. These libraries maintain tables that associate the "Tasks" routine registered by the SYS INIT service with each interrupt source in the system.

Alternately, in a statically-linked system implementation, the ISR may be implemented by the system designer or integrator (in the configuration-specific "sys_int.c" file). Such "static" ISR implementations must identify the source of the interrupt then directly call the appropriate module's "Tasks" routine. This requires knowledge of the modules that have been included in the system and cannot be implemented in advance as a library.

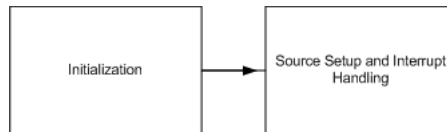
Note: It is also possible, in a highly optimized system (or to support highly resource-restricted parts), to implement the logic of the module's "Tasks" routine directly in the raw ISR. However, this method is not recommended unless absolutely necessary to meet system timing requirements.

Board Support Packages (BSPs)

If the processor is affixed directly to the board, the BSP may also implement any required "raw" ISRs, eliminating the need for the system designer or integrator to implement the ISR(s) himself. Refer to the documentation for the BSP in use for details on what initialization and ISR support it provides. This support is not implemented by the Interrupt System Services library.

5.7.5.4.2 Library Overview

The diagram below describes the major components of the usage model.



Library Usage Model

Each of the blocks in the diagram above correspond to the library interface section.

Library Interface Section	Description
Interrupt System Setup Functions	Provides processor specific initialization of the interrupt system.
Global Interrupt Management Functions	Provide interface routines to enable/disable all interrupts on the system.

5.7.5.4.3 How the Library Works

The Interrupt System Service Library can be used by a device driver, middleware layer, or application to provide access to, and control over, interrupts to the processor.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the modes supported for your device.

5.7.5.4.3.1 Interrupt System Setup

The Interrupt System Service library must be initialized by calling the `SYS_INT_Initialize` routine. If the MPLAB Harmony dynamic initialization service is used, the "`SYS_INT_Initialize`" routine will be called automatically when the "`SYS_Initialize`" function is called. In a statically initialized system, the system designer or integrator must implement the "`SYS_Initialize`" function and that function must call "`SYS_INT_Initialize`" before initializing any modules that might require use of the interrupt system service. Once the library has been initialized, call the function `SYS_INT_Enable` to enable interrupts to the processor. However, before enabling the generation of interrupts to the processor, each individual module (driver, library, or application) must have a "Tasks" routine in place (either registered with `SYS_INT_DynamicRegister` or statically linked to the raw ISR) to handle the interrupt before it enables its own interrupt.

Example: Initializing the System Interrupt Library

```

SYS_INT_Initialize();
// Initialize all interrupt-aware software modules
SYS_INT_Enable();
  
```

5.7.5.4.3.2 Critical Sections

Critical Sections

Critical sections of code are small sections of code that must execute atomically, with no possibility of being interrupted. To support this, the following technique can be used.

Global Interrupt Management provides routines to create a global critical section of code.

Global Critical Section

If no interrupts of any kind can be allowed within a critical section of code, the following routines can be used to ensure this.

- [SYS_INT_Disable](#): To start a critical section, all interrupts are disabled with the call of this function
- [SYS_INT_Enable](#) : To end a critical section, interrupts are enabled from the interrupt controller to the core.
- [SYS_INT_IsEnabled](#): Status to indicate if interrupts are currently enabled or not.

Example: Global Critical Section

```
bool flag;
flag = SYS_INT_Disable();

// Do something critical

if (flag)
{
    SYS_INT_Enable();
}
```

Source Interrupt Management provides interface routines to create local critical sections

Local Critical Sections

Normally, it is not necessary to globally disable all possible interrupts. For example, in a driver for a specific device, it is not normally important if an unrelated interrupt occurs in the middle of a critical section of code. However, if the interrupt for the source that the driver manages must not occur within a critical section of code, it can be protected using the following technique.

Example: Local Critical Section

```
bool flag;
// interrupt source enable status before disable is called
flag = SYS_INT_SourceDisable(MY_DRIVER_INTERRUPT_SOURCE);

// Do something critical

if (flag)
{
    SYS_INT_SourceEnable(MY_DRIVER_INTERRUPT_SOURCE);
}
```

Note: These methods of protecting critical sections is usually implemented as part of an Operating System Abstraction Layer (OSAL), so it is not normally necessary to use these examples explicitly. Normally, the OSAL will provide single functions or macros that implement this functionality. So, if available, an OSAL method is preferred over implementing the critical section code as shown in the above examples.

5.7.5.4.3.3 Source Interrupt Management

The driver, middleware, or application's interrupt-handling "Tasks" routine must do two things at a minimum, in the following order.

1. Remove the cause of the interrupt
2. Clear the interrupt source by calling the function [SYS_INT_SourceStatusClear](#)

Exactly what actions are necessary to remove the cause of an interrupt is completely dependent on the source of the interrupt. This is normally the main purpose of the driver itself and is beyond the scope of this section. Refer to the documentation for the peripheral being managed.

WARNING! The cause of the interrupt must be removed before clearing the interrupt source or the interrupt may re-occur immediately after the source is cleared potentially causing an infinite loop. An infinite loop may also occur if the source is not cleared before the interrupt-handler returns.

Example: Handling Interrupts

```
void DRV_MYDEV_Tasks( SYS_MODULE_OBJ object )
{
```

```
// Remove the cause of the interrupt  
//...  
  
// Clear Interrupt source  
SYS_INT_SourceStatusClear(myIntSourceID);  
}
```

Note: The value of "myIntSourceID" is usually either a static or dynamic configuration option. Refer to the documentation for the specific device driver to identify how to define the interrupt source ID.

Testing Interrupt

Some times it is necessary to cause an interrupt in software, possibly for testing purposes. To support this, the function [SYS_INT_SourceStatusSet](#) is provided.

Example: Causing an Interrupt in Software

```
SYS\_INT\_SourceStatusSet(MY_DRIVER_INTERRUPT_SOURCE);
```

Note: This feature is not available for all interrupt sources on all Microchip microcontrollers. Refer to the data sheet for the microcontroller being used to determine if it is possible for software to set a specific interrupt source.

5.7.5.5 Configuring the Library

Three things must be correctly configured in order to use the System Interrupt library.

1. Select the Appropriate Processor
2. Initialize the Interrupt System Service
3. Configure the Raw ISR Support

Select the Appropriate Processor

The following data types are dependent on the processor selection and are actually defined in the interrupt peripheral library for the specific microcontroller being used.

- [INT_SOURCE](#)
- [INT_PRIORITY](#)
- [INT_SUBPRIORITY](#)

These are configured by selecting the appropriate processor in MPLAB, which adds the "mprocessor" option to the compiler command line to identify the correct processor and processor-specific implementation of the peripheral library to use. Since the System Interrupt library is part of the Microchip Firmware Framework, it will get built with the correct definition of these data types.

Initialize the Interrupt System Service

There are two ways to initialize the interrupt system service, depending on whether you're using a static configuration or a dynamic configuration.

For a dynamic configuration the constant `SYS_INT_DYNAMIC` needs to be defined. This makes [SYS_INT_DynamicRegister](#) and [SYS_INT_DynamicDeregister](#) available. The required driver tasks routines need to be registered using [SYS_INT_DynamicRegister](#) function.

For a static configuration, the system designer or integrator must implement the [SYS_INT_Initialize](#) routine. This routine's purpose is to perform any actions necessary to initialize the interrupt subsystem and interrupt controller on the specific processor and system, usually interacting directly with the Interrupt controller modules Peripheral Library (PLIB) to accomplish these tasks.

Configure the Raw ISR Support

In some systems, there may only be a single actual ("raw") ISR to handle all interrupts. In this sort of system, most of the System Interrupt library may be implemented in software, with only the highest level interrupt being supported by hardware. In other systems, all interrupts may be supported by separate ISRs and vector selection and prioritization will be supported by hardware.

ISRs may be dynamically linked to specific interrupt sources or they may be statically linked at build time. If a dynamic interrupt library is used (by defining the constant `SYS_INT_DYNAMIC`), the calls to [SYS_INT_DynamicRegister](#) will register a pointer to the given "Tasks" routine for each registered interrupt source in an internal table. The dynamic library will then determine the source of the interrupt and call the given "Tasks" routine.

If a static configuration is desired, the "raw" ISR support must be implemented so that it directly calls (using static, build-time linkage) the appropriate module's "Tasks" routine. This requires the system implementer or integrator to implement the raw ISR, but it reduces the amount of overhead necessary to handle interrupts, reducing both interrupt latency and code size.

5.7.5.5.1 Static Configuration

When statically configuring raw ISR support, the system implementer or integrator must directly implement the raw ISRs in an appropriate manner for the selected processor. The Raw ISR, must then call the appropriate "Tasks" routine to properly handle and clear the interrupt source.

Description

A static configuration of the raw ISR support for MPLAB Harmony requires processor-family-specific knowledge. Or, more accurately, it requires compiler-specific knowledge. The examples below will show a sampling of how to implement a raw ISR for different part families. Refer to the compiler manual for the selected processor family for details of how to implement an interrupt service routine.

Raw ISR Responsibilities:

1. Identify the Interrupt Source
2. Call the Appropriate Module's "Tasks" Routine

The first thing a raw ISR must do is identify the source of the interrupt. For some processor families, each interrupt source has its own interrupt "vector". This means that the only time a specific ISR is called is when a specific source has caused an interrupt. Thus, the raw ISR can assume that every time it is called it's source has caused an interrupt. Other processor families combine two or more interrupt sources into a single interrupt "vector". In such a case, the raw ISR must verify which source caused the interrupt and it must verify if that source is enabled. In either case, once the raw ISR has identified the interrupt source, it must call the appropriate module's "Tasks" routine to service and clear the interrupt.

Example: PIC32 Timer-1 Raw ISR

```
void __ISR ( _TIMER_1_VECTOR ) _InterruptHandler_TMR_1_stub( void )
{
    /* Call the timer driver's "Tasks" routine */
    DRV_TMR_Tasks ( gTMRObject );
}
```

Example: PIC24 Timer-1 Raw ISR

```
void __attribute__ ( ( interrupt, no_auto_psv ) ) _ISR _T1Interrupt( void )
{
    /* Call the timer driver's "Tasks" routine */
    DRV_TMR_Tasks ( gTMRObject );
}
```

Example: PIC18 Timer-1 Raw ISR

```
#pragma interrupt_level 0
void interrupt low_priority InterruptHandler_low_priority_stub ( void )
{
    if ( SYS_INT_SourceIsEnabled( gTMRObject ) && PLIB_INT_SourceFlagGet( gTMRObject ) )
    {
        /* Call the timer driver's "Tasks" routine */
        DRV_TMR_Tasks ( gTMRObject );
    }
    // Check other sources...
}
```

In all of the above examples, "gTMRObject" holds the return value from the ["DRV_TMR_Initialize"](#) routine.

The [SYS_INT_DynamicRegister](#) and [SYS_INT_DynamicDeregister](#) routines are macro switched to compile away to nothing if a static configuration is chosen

5.7.5.5.2 Dynamic Configuration

When dynamically configuring raw ISR support, the system implementer or integrator must register each interrupt-driven driver or module's "Tasks" routine with the dynamic system interrupt service for the appropriate interrupt source. The dynamic SYS INT service will then ensure that the appropriate "Tasks" routine is called when an interrupt occurs.

Description

When using the dynamic system interrupt (SYS INT) service, it is not necessary to implement "raw ISRs" for interrupt-driven modules. The processor-family-specific, dynamic SYS INT implementation provided with MPLAB Harmony implements the "raw ISR(s)" so the system developer or integrator doesn't have to. Instead, the system developer must register the module's "Tasks" routine(s) using the "SYS_ModuleRegister" routine after registering the module in the module registration routine (described in the SYS INIT documentation). The example below shows how a module must register its ISR "Tasks" routine.

Example: Dynamic Registration of an Interrupt-Driven Module

```
// Register the TMR driver's "Tasks" routine with the SYS INT service
SYS_INT_DynamicRegister(object, DRV_TMR_Tasks, PLIB_INT_SOURCE_TIMER_1);
```

The module init routine must register the module's "Tasks" routine(s) with the SYS INT service instead of the SYS TASKS service. to do this, it calls the ["SYS_INT_DynamicRegister"](#) routine, passing in the object(the same object handle returned by the module's initialization routine), along with a pointer to the module's "Tasks" routine and the interrupt source with which it will be associated.

5.7.5.6 Library Interface

Data Types and Constants

	Name	Description
	SYS_INT_TASKS_POINTER	Pointer to an interrupt-handling "Tasks" routine.

Global Interrupt Management Functions

	Name	Description
	SYS_INT_Disable	Disables interrupts to the processor.
	SYS_INT_Enable	Enables global interrupts to the processor.
	SYS_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.

Interrupt Source Management Functions

	Name	Description
≡	SYS_INT_SourceDisable	Disables the specified source from generating interrupts to the processor.
≡	SYS_INT_SourceEnable	Enables the specified source to generate interrupts to the processor.
≡	SYS_INT_SourceIsEnabled	Identifies if the specified source is enabled or disabled.
≡	SYS_INT_SourceStatusClear	Clears the interrupt request for the specified interrupt source.
≡	SYS_INT_SourceStatusGet	Determines the status of the specified interrupt source.
≡	SYS_INT_SourceStatusSet	Sets the specified interrupt source.
≡	SYS_INT_VectorPrioritySet	Sets the given interrupt vector to the specified priority.
≡	SYS_INT_VectorSubprioritySet	Sets the specified interrupt vector to the given sub priority.

Interrupt System Setup Functions

	Name	Description
≡	SYS_INT_Initialize	Configures and initializes the interrupt subsystem.
≡	SYS_INT_DynamicDeregister	Deregisters the current ISR from the given interrupt source.
≡	SYS_INT_DynamicRegister	Registers an Interrupt "Tsks" Routine for the specified interrupt source or trap).

Description**5.7.5.6.1 Interrupt System Setup Functions****5.7.5.6.1.1 SYS_INT_Initialize Function****C**

```
void SYS_INT_Initialize();
```

Description

This function configures and initializes the interrupt subsystem appropriately for the current system design.

Preconditions

None.

Returns

None.

Remarks

None.

The System Interrupt library must be initialized by calling the SYS_INT_Initialize routine. This is normally done in the "[SYS_Initialize](#)" routine before any interrupt support is used. If the dynamic interrupt system service is not used, the "[SYS_Initialize](#)" routine must be implemented by the system designer or integrator as required by the system design and is not implemented by the System Interrupt library. The global interrupts are enabled as a part the call to "SYS_INT_Initialize". However, before enabling the generation of interrupts to the processor, each individual module (driver, library, or application) must have a "Tasks" routine to in place (statically linked to the raw ISR) to handle the interrupt before it enables it's own interrupt.

Example: Initializing the System Interrupt Library

```
SYS_INT_Initialize();
```

Example

```
// Initialize the interrupt system. This needs to be done in the initialization  
// code.  
SYS_INT_Initialize();
```

5.7.5.6.1.2 SYS_INT_DynamicDeregister Function**C**

```
void SYS_INT_DynamicDeregister(  
    INT_SOURCE source  
);
```

Description

This function deregisters the current Interrupt Service Routine (ISR), if any, from the specified interrupt source.

Preconditions

[SYS_INT_Initialize](#) must have been called.

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Returns

None.

Remarks

It is safe to call this routine, even if no ISR has been registered for the given interrupt source.

Calling this routine is optional. If the system is designed such that the given ISR is expected to always be available once the system has been initialized, this routine does not need to be called.

Example

```
SYS_INT_DynamicDeregister(SYS_INT_TIMER_1);
```

5.7.5.6.1.3 SYS_INT_DynamicRegister Function**C**

```
void SYS_INT_DynamicRegister(  
    INT_SOURCE source,  
    SYS_INT_TASKS_POINTER tasks,  
    SYS_MODULE_OBJ object  
);
```

Description

This function registers an Interrupt "Tasks" Routine for the specified interrupt source or trap).

Preconditions

[SYS_INT_Initialize](#) must have been called.

Parameters

Parameters	Description
source	Identifier for the desired interrupt source
tasks	Pointer to the tasks routine
object	Handle to the module instance

Returns

None.

Remarks

This routine only generates executable code when a driver is configured to register dynamically its "Tasks" routine with the system interrupt service. However, it should be called even if the ISR-to-source association is defined statically at compile time to maintain source code compatibility. A device driver normally registers its own ISR from its initialization routine.

In the example code, the macros MY_DRIVER_INTERRUPT_SOURCE would be defined in the appropriate configuration header, which would be included by the driver source file where the "DRV_MYDEV_Tasks" routine and the MyParam data would be defined.

It is safe to call this routine without first calling [SYS_INT_DynamicDeregister](#), even if a previous ISR has been registered. The effect will be that the new ISR supplants the old one.

The System Interrupt library must be initialized by calling the [SYS_INT_Initialize](#) routine. This is normally done in the "[SYS_Initialize](#)" routine before any interrupt support is used.

If the dynamic interrupt system service is not used, the "[SYS_Initialize](#)" routine must be implemented by the system designer or integrator as required by the system design and is not implemented by the System Interrupt library.

Once the library has been initialized, call the function [SYS_INT_Enable](#) to enable interrupts to the processor. However, before enabling the generation of interrupts to the processor, each individual module (driver, library, or application) must have a "Tasks" routine in place (either registered with SYS_INT_DynamicRegister or statically linked to the raw ISR) to handle the interrupt before it enables its own interrupt.

Example: Initializing the System Interrupt Library

```
// Initialize the interrupt system.  
SYS_INT_Initialize();  
// Initialize all interrupt-aware software modules  
SYS_INT_Enable();
```

Example

```
SYS_INT_Initialize();  
SYS_INT_DynamicRegister(MY_DRIVER_INTERRUPT_SOURCE, DRV_MYDEV_Tasks, MyObject);
```

5.7.5.6.2 Global Interrupt Management Functions

5.7.5.6.2.1 SYS_INT_Disable Function

C

```
bool SYS_INT_Disable();
```

Description

This function disables interrupts to the processor at the top level. This function can be called to prevent any source from being able to generate an interrupt. It returns the global interrupt status before disabling the interrupts.

Preconditions

[SYS_INT_Initialize](#) must have been called.

Returns

true - Global Interrupts are enabled (before the call to disable) false - Global Interrupts are disabled (before the call to disable)

Remarks

None.

Example

```
// Interrupt enable status
bool flag;
// Disable the global interrupts
flag = SYS_INT_Disable();
// Do something critical
// Check if interrupts were disabled/enabled
if (flag)
{
    // enable the global interrupts if they were enabled before the
    // call to SYS_INT_Disable()
    SYS_INT_Enable();
}
```

5.7.5.6.2.2 SYS_INT_Enable Function**C**

```
void SYS_INT_Enable();
```

Description

This function enables interrupts to the processor at the top level, allowing any currently enabled source to generate an interrupt. This function must be called before any source will be able to generate an interrupt.

Preconditions

None.

Returns

None.

Remarks

SYS_INT_Enable is called from the [SYS_INT_Initialize\(\)](#) function.

Example

```
// Check if global interrupts are enabled
if(!SYS_INT_IsEnabled())
{
    // Enable the global interrupts.
    SYS_INT_Enable();
}
```

5.7.5.6.2.3 SYS_INT_IsEnabled Function**C**

```
bool SYS_INT_IsEnabled();
```

Description

This function identifies if interrupts are enabled or disabled at the top level.

Preconditions

[SYS_INT_Initialize](#) must have been called.

Returns

- true - If the interrupts are currently enabled
- false - If the interrupts are currently disabled

Remarks

None.

Example

```
// Check if global interrupts are enabled
if ( SYS_INT_IsEnabled() )
{
    // Interrupt enable status
    bool flag;
    // Disable the global interrupts.
    flag = SYS_INT_Disable();
}
```

5.7.5.6.3 Interrupt Source Management Functions

5.7.5.6.3.1 SYS_INT_SourceDisable Function

C

```
bool SYS_INT_SourceDisable(
    INT_SOURCE source
);
```

Description

This routine disables the given source from generating interrupts the processor when events occur. It returns the interrupt source enable/disable status before disabling the interrupt source.

Preconditions

[SYS_INT_Initialize](#) must have been called.

Parameters

Parameters	Description
source	Identifier for the desired interrupt source.

Returns

- true - The Interrupt source is enabled (before the call to SYS_INT_SourceDisable)
- false - The Interrupt source is disabled (before the call to SYS_INT_SourceDisable)

Remarks

None.

Example

```
// interrupt source enable/disable status.
bool flag
// Initialize the interrupt system. This needs to be done in the initialization
// code.
SYS_INT_Initialize();
// Disable the interrupt source
flag = SYS_INT_SourceDisable(INT_SOURCE_PARALLEL_PORT);
// before enabling the source check the enable/disable status
if(flag)
{
    SYS_INT_SourceEnable(INT_SOURCE_PARALLEL_PORT);
}
```

5.7.5.6.3.2 SYS_INT_SourceEnable Function

C

```
void SYS_INT_SourceEnable(  
    INT_SOURCE source  
);
```

Description

This routine enables the specified source to generate interrupts to the processor when events occur.

Preconditions

[SYS_INT_Initialize](#) must have been called and an ISR must have been registered for the source.

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Returns

None.

Remarks

An Interrupt Service Routine (ISR) for the given interrupt source must be ready to receive the call before the source is enabled.

Example

```
// Initialize the interrupt system. This needs to be done in the initialization  
// code.  
SYS_INT_Initialize();  
// Enable the interrupt source  
SYS_INT_SourceEnable(INT_SOURCE_PARALLEL_PORT);
```

5.7.5.6.3.3 SYS_INT_SourceIsEnabled Function

C

```
bool SYS_INT_SourceIsEnabled(  
    INT_SOURCE source  
);
```

Description

This function identifies if the specified source is currently enabled or is currently disabled.

Preconditions

[SYS_INT_Initialize](#) must have been called.

Parameters

Parameters	Description
source	Identifier for the desired interrupt source.

Returns

- true - If the given source is currently enabled.
- false - If the given source is currently disabled.

Remarks

None.

Example

```
// Initialize the interrupt system. This needs to be done in the initialization
// code.
SYS_INT_Initialize();
// Check if the required interrupt source is enabled
if ( SYS_INT_SourceIsEnabled(INT_SOURCE_PARALLEL_PORT))
{
    // App code
}
```

5.7.5.6.3.4 SYS_INT_SourceStatusClear Function**C**

```
void SYS_INT_SourceStatusClear(
    INT_SOURCE source
);
```

Description

This function clears the interrupt request for the specified interrupt source.

Preconditions

[SYS_INT_Initialize](#) must have been called.

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Returns

None.

Remarks

None.

Example

```
// Initialize the interrupt system. This needs to be done in the initialization
// code.
SYS_INT_Initialize();
// Check if the interrupt source flag is set
if ( SYS_INT_SourceStatusGet(INT_SOURCE_PARALLEL_PORT) )
{
    // Clear the interrupt flag
    SYS_INT_SourceStatusClear(INT_SOURCE_PARALLEL_PORT);
}
```

5.7.5.6.3.5 SYS_INT_SourceStatusGet Function**C**

```
bool SYS_INT_SourceStatusGet(
    INT_SOURCE source
);
```

Description

This function determines the current status of the interrupt source.

Preconditions

[SYS_INT_Initialize](#) must have been called.

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Returns

- true - If the given interrupt source is currently set
- false - If the given interrupt source is not currently set

Remarks

Works even if the interrupt source or interrupts in general have not been enabled, so it can be used for polling implementations.

Example

```
// Initialize the interrupt system.This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Check if the required interrupt source is set
if ( SYS_INT_SourceStatusGet(INT_SOURCE_PARALLEL_PORT) )
{
    // Handle interrupt
}
```

5.7.5.6.3.6 SYS_INT_SourceStatusSet Function**C**

```
void SYS_INT_SourceStatusSet(
    INT_SOURCE source
);
```

Description

This function sets the specified interrupt source, causing the processor to be interrupted if interrupts are enabled, the source has been enabled, and the priority is higher than the current priority.

Preconditions

[SYS_INT_Initialize](#) must have been called and an ISR must have been registered for the source (if interrupts and the source are enabled).

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Returns

None.

Remarks

Not supported for all interrupt sources. Check the specific data sheet for software clear only interrupt sources.

The driver, middleware, or application's interrupt-handling "Tasks" routine must do two things at a minimum, in the following order.

1. Remove the cause of the interrupt
2. Clear the interrupt source by calling the function [SYS_INT_SourceStatusClear](#)

Exactly what actions are necessary to remove the cause of an interrupt is completely dependent on the source of the interrupt. This is normally the main purpose of the driver itself and is beyond the scope of this section. Refer to the documentation for the peripheral being managed.

WARNING! The cause of the interrupt must be removed before clearing the interrupt source or the interrupt may re-occur immediately after the source is cleared potentially causing an infinite loop. An infinite loop may also occur if the source is not cleared before the interrupt-handler returns.

Example: Handling Interrupts

```
void DRV_MYDEV_Tasks( SYS_MODULE_OBJ object )
{
    // Remove the cause of the interrupt
    //...

    // Clear Interrupt source
    SYS_INT_SourceStatusClear(myIntSourceID);
}
```

Note: the value of "myIntSourceID" is usually either a static or dynamic configuration option. Refer to the documentation for the specific device driver to identify how to define the interrupt source ID.

Testing Interrupt

Some times it is necessary to cause an interrupt in software, possibly for testing purposes. To support this, the function SYS_INT_SourceStatusSet is provided.

Example: Causing an Interrupt in Software

```
SYS_INT_SourceStatusSet(MY_DRIVER_INTERRUPT_SOURCE);
```

Note: This feature is not available for all interrupt sources on all Microchip microcontrollers. Refer to the data sheet for the microcontroller being used to determine if it is possible for software to set a specific interrupt source.

Example

```
// Initialize the interrupt system. This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Check if interrupt source flag is set
if ( !SYS_INT_SourceStatusGet(INT_SOURCE_PARALLEL_PORT) )
{
    // Set the interrupt source flag
    SYS_INT_SourceStatusSet(INT_SOURCE_PARALLEL_PORT);
}
```

5.7.5.6.3.7 SYS_INT_VectorPrioritySet Function

C

```
void SYS_INT_VectorPrioritySet(
    INT_VECTOR vector,
    INT_PRIORITY_LEVEL priority
);
```

Description

This routine sets the given interrupt vector to the specified priority.

Preconditions

[SYS_INT_Initialize](#) must have been called.

Parameters

Parameters	Description
vector	Identifier for the desired interrupt vector
priority	Priority (if supported)

Returns

None.

Remarks

This feature is not supported on all devices. Refer to the specific device data sheet or family reference manual to determine whether this feature is supported.

In the example code, the macros MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_PRIORITY would be defined appropriately during configuration.

Example

```
#define MY_DRIVER_INTERRUPT_VECTOR INT_VECTOR_T1
#define MY_DRIVER_ISR_PRIORITY     INT_PRIORITY_LEVEL2
// Initialize the interrupt system.This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Assign priority to the interrupt vector
SYS_INT_VectorPrioritySet(MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_PRIORITY);
```

5.7.5.6.3.8 SYS_INT_VectorSubprioritySet Function**C**

```
void SYS_INT_VectorSubprioritySet(
    INT_VECTOR vector,
    INT_SUBPRIORITY_LEVEL subpriority
);
```

Description

This function sets the specified interrupt vector to the specified sub-priority.

Preconditions

[SYS_INT_Initialize](#) must have been called.

Parameters

Parameters	Description
vector	Identifier for the desired interrupt vector
subpriority	Subpriority (if supported)

Returns

None.

Remarks

This feature is not supported on all devices. Refer to the specific device data sheet or family reference manual to determine whether this feature is supported.

In the example code, the macros MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_SUB_PRIORITY would be defined appropriately during configuration.

Example

```
#define MY_DRIVER_INTERRUPT_VECTOR INT_VECTOR_T1
#define MY_DRIVER_ISR_PRIORITY     INT_PRIORITY_LEVEL2
#define MY_DRIVER_ISR_SUB_PRIORITY INT_SUBPRIORITY_LEVEL1
// Initialize the interrupt system.This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Assign priority to the interrupt vector
SYS_INT_VectorPrioritySet(MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_PRIORITY);
```

```
// Assign sub-priority to the interrupt vector
SYS_INT_VectorSubprioritySet(MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_SUB_PRIORITY);
```

5.7.5.6.4 Data Types and Constants

5.7.5.6.4.1 SYS_INT_TASKS_POINTER Type

C

```
typedef void (* SYS_INT_TASKS_POINTER)(SYS_MODULE_OBJ object);
```

Description

Interrupt Tasks Routine Pointer

This data type defines a pointer to an interrupt-handling "Tasks" routine. The form of a tasks routine is as follows:

```
void My_Tasks ( SYS_MODULE_OBJ object );
```

Where "MyTasks" is the name of the tasks routine and object is a Handle to the module instance.

Remarks

"Tasks" is normally defined by a device driver, middleware, or system layer.

The term Interrupt Service Routine (ISR) is used for the "raw" ISR code that is either located directly at the interrupt vector address or whose address is loaded from the interrupt vector. The term "Tasks" routine is used to identify the driver-specific routine that is called by the actual ISR to perform the tasks necessary to handle and clear the interrupt.

5.7.5.7 Files

Files

Name	Description
sys_int.h	Interrupt System Service.

Description

5.7.5.7.1 sys_int.h

Interrupt System Service Library Interface Definition

This file contains the interface definition for the Interrupt System Service. It provides a way to interact with the interrupt subsystem to manage the occurrence of interrupts for sources supported by the system.

Functions

	Name	Description
≡	SYS_INT_Disable	Disables interrupts to the processor.
≡	SYS_INT_DynamicDeregister	Deregisters the current ISR from the given interrupt source.
≡	SYS_INT_DynamicRegister	Registers an Interrupt "Tsks" Routine for the specified interrupt source or trap).
≡	SYS_INT_Enable	Enables global interrupts to the processor.
≡	SYS_INT_Initialize	Configures and initializes the interrupt subsystem.
≡	SYS_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.
≡	SYS_INT_SourceDisable	Disables the specified source from generating interrupts to the processor.

	SYS_INT_SourceEnable	Enables the specified source to generate interrupts to the processor.
	SYS_INT_SourcesEnabled	Identifies if the specified source is enabled or disabled.
	SYS_INT_SourceStatusClear	Clears the interrupt request for the specified interrupt source.
	SYS_INT_SourceStatusGet	Determines the status of the specified interrupt source.
	SYS_INT_SourceStatusSet	Sets the specified interrupt source.
	SYS_INT_VectorPrioritySet	Sets the given interrupt vector to the specified priority.
	SYS_INT_VectorSubprioritySet	Sets the specified interrupt vector to the given sub priority.

Types

	Name	Description
	SYS_INT_TASKS_POINTER	Pointer to an interrupt-handling "Tasks" routine.

File Name

sys_int.h

Company

Microchip Technology Inc.

5.7.6 Messaging System Service Library

5.7.6.1 Introduction

Messaging System Service Library

for

Microchip Microcontrollers

This library provides intra-process and inter-process communication by the sending and receiving of simple messages. The format of these messages is under developer control, providing flexibility to tune message format for each application. The number and size of message queues is under developer control, with a message priority scheme implemented by multiple queues. Each priority queue has an configurable size. The number of message types and the number of receiving mailboxes is also configurable.

Description

This library provides intra-process and inter-process communication by the sending and receiving of simple messages. The format of these messages is under developer control, providing flexibility to tune message format for each application. The number and size of message queues is under developer control, with a message priority scheme implemented by multiple queues. Each priority queue has an configurable size. The number of message types and the number of receiving mailboxes is also configurable.

Consider an application that combines graphics on a display screen, with a touch overlay on top of the screen, and several buttons. System messages can be used by touch software to alert the graphics part of the application to update the screen as well as alert other parts of the application to perhaps change the audio volume whenever a slider value has changed. This is easily supported by the Messaging System Services Library.

5.7.6.2 Release Notes

MPLAB Harmony Version: v0.70b **Messaging System Service Library Version :** 0.01a **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.7.6.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.7.6.4 Using the Library

Interface Header File: [sys_msg.h](#)

The interface to the Messaging System Service library is defined in the "[sys_msg.h](#)" header file".

This library uses calloc to allocate memory for:

- Message queues for each priority as part of [SYS_MSG_Initialize](#)
- Mailbox definition objects
- Message type objects

If calls to calloc fails to allocate the needed memory then object handles are returned with a value of [SYS_OBJ_HANDLE_INVALID](#).

Here is a simple example that can be run on any PIC32 starter kit:

```
#include <sys/appio.h>
#include <GenericTypeDefs.h>

#include "system/common/sys_module.h"
#include "system/msg/sys_msg.h"
#include "system/msg/src/sys_msg_local.h"
```

```

// Strawman callback functions for three mailboxes
void myCallBack0( SYS_MSG_OBJECT *pMessage )
{
    DBPRINTF("CallBack0:: Message Type: %d, nSource: %d\r\n"
            "        param 0: %d, param1: %d, param2: %d \r\n",
            pMessage->nMessageTypeID,
            pMessage->nSource, pMessage->param0,
            pMessage->param1, pMessage->param2
            );
}

void myCallBack1( SYS_MSG_OBJECT *pMessage )
{
    DBPRINTF("CallBack1:: Message Type: %d, nSource: %d\r\n"
            "        param 0: %d, param1: %d, param2: %d \r\n",
            pMessage->nMessageTypeID,
            pMessage->nSource, pMessage->param0,
            pMessage->param1, pMessage->param2
            );
}

void myCallBack2( SYS_MSG_OBJECT *pMessage )
{
    DBPRINTF("CallBack2:: Message Type: %d, nSource: %d\r\n"
            "        param 0: %d, param1: %d, param2: %d \r\n",
            pMessage->nMessageTypeID,
            pMessage->nSource, pMessage->param0,
            pMessage->param1, pMessage->param2
            );
}

int main(void)
{
    SYS_MSG_MESSAGING_OBJECT oSysMsg;
    SYS_OBJ_HANDLE hSysMsg, hMsgType[5], hMailbox[3];
    SYS_MSG_INSTANCE iSysMsg = SYS_MSG_0;
    SYS_MSG_OBJECT myMessage[5];

    { // SYS_MSG_Initialize
        uint8_t nQSizes[] = SYS_MSG_BUFFER_SIZES;

        // Initialize the messaging system. This needs to be done in the initialization code.
        hSysMsg = SYS_MSG_Initialize(iSysMsg, SYS_MSG_MAX_PRIORITY+1, nQSizes);

        SYS_ASSERT( SYS_OBJ_HANDLE_INVALID != hSysMsg, "Bad hSysMsg!" );
    }

    // Create the message types to be used
    // ID:      :Priority
    hMsgType[0] = SYS_MSG_TypeCreate(iSysMsg, 1<<0, 0);
    hMsgType[1] = SYS_MSG_TypeCreate(iSysMsg, 1<<1, 1);
    hMsgType[2] = SYS_MSG_TypeCreate(iSysMsg, 1<<2, 2);
    hMsgType[3] = SYS_MSG_TypeCreate(iSysMsg, 1<<3, 3);
    hMsgType[4] = SYS_MSG_TypeCreate(iSysMsg, 1<<4, 4);

    // Create the mailboxes to be used
    hMailbox[0] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack0 );
    hMailbox[1] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack1 );
    hMailbox[2] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack2 );

    // Identify which messages are of interest for each mailbox.
    SYS_MSG_MailboxMsgAdd(hMailbox[0], hMsgType[0]);
    SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<1));
}

```



```

SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<2));
SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<3));
SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<4));
SYS_MSG_MailboxMsgAdd(hMailbox[1], hMsgType[0]);
SYS_MSG_MailboxMsgAdd(hMailbox[2], hMsgType[0]);

//Send and receive messages
SYS_MSGQ_ELEMENT *pQElement;
SYS_MSG_OBJECT *pMessage;
SYS_MSG_RESULTS myResult;
SYS_MSG_QUEUE_STATUS qStatus[5];
uint16_t iPriority;

for (iPriority=0; iPriority<=SYS_MSG_MAX_PRIORITY; iPriority++)
{
    myMessage[iPriority].nSource = SYS_MSG_MAX_PRIORITY-iPriority;
    myMessage[iPriority].nMessageTypeID = 1<<iPriority;
    myMessage[iPriority].param0 = iPriority;
    myMessage[iPriority].param1 = 0;
    myMessage[iPriority].param2 = 0;
}

myMessage[4].param1 = 0;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[4].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[4].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[4].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] ); // Should fail, queue full!
SYS_ASSERT(myResult == SYS_MSG_NOT_SENT, "Result should NOT Be Sent!");

myMessage[3].param1 = myMessage[4].param1;
myMessage[3].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[3] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[3].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[3] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[2].param1 = myMessage[3].param1;
myMessage[2].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[2] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[2].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[2] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[1].param1 = myMessage[2].param1;
myMessage[1].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[1] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[1].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[1] );

```

```

SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[0].param1 = myMessage[1].param1;
myMessage[0].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[0] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[0].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[0] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

myMessage[0].param1++;
myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[0] );
SYS_ASSERT(myResult == SYS_MSG_SENT, "Message not sent!");

// Preemptively read all Mailbox Zero messages
while ( NULL != (pMessage = SYS_MSG_MailboxMessagesGet(hMailbox[0])) )
{
    // "Process" message.
    DBPRINTF("Mailbox Zero: Message Type: %d, nSource: %d\r\n"
            "                param 0: %d, param1: %d, param2: %d \r\n",
            pMessage->nMessageTypeID,
            pMessage->nSource, pMessage->param0,
            pMessage->param1, pMessage->param2
            );
}

// Deliver messages for all mailboxes.
DBPRINTF("\r\nGot Messages: %d\r\n", SYS_MSG_GotMessages(iSysMsg));
for (iPriority=0; iPriority<=SYS_MSG_MAX_PRIORITY; iPriority++)
{
    qStatus[iPriority] =
        SYS_MSG_QueueStatus((SYS_OBJ_HANDLE)iSysMsg, iPriority);
}
DBPRINTF("Queue Status (4:-1:0): %d, %d, %d, %d, %d\r\n\r\n",
        qStatus[4], qStatus[3], qStatus[2], qStatus[1], qStatus[0]);
while ( (pQElement = SYS_MSG_MessageReceive(iSysMsg)) != NULL )
{
    SYS_MSG_MessageDeliver(iSysMsg, pQElement);
    for (iPriority=0; iPriority<=SYS_MSG_MAX_PRIORITY; iPriority++)
    {
        qStatus[iPriority] =
            SYS_MSG_QueueStatus((SYS_OBJ_HANDLE)iSysMsg, iPriority);
    }
    DBPRINTF("Queue Status (4:-1:0): %d, %d, %d, %d, %d\r\n\r\n",
            qStatus[4], qStatus[3], qStatus[2], qStatus[1], qStatus[0]);
}
DBPRINTF("Got Messages: %d\r\n", SYS_MSG_GotMessages(iSysMsg));
}

return 0;
}

```

5.7.6.4.1 Abstraction Model

This library provides an abstraction of the messaging subsystem that is used by device drivers, middleware libraries and applications to receive and control interrupts in real time.

[<add figure> and supporting text](#)

5.7.6.4.2 Library Overview

The following diagram describes the major components of the usage model.

<add figure>

Library Usage Model

Each of the blocks in the diagram correspond to the library interface section.

Library Interface Section	Description
Initialization, Tasks, and Versioning Functions	Provides
Mailbox Functions	Provides
Message Type Functions	Provides
Message Send/Receive Functions	Provides
Utility Functions	Provides
Configuration Functions	Provides

5.7.6.4.3 How the Library Works

The Messaging System Service Library can be used by a device driver, middleware layer, or application to provide access to, and control over, interrupts to the processor.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the modes supported for your device.

5.7.6.5 Configuring the Library

The file `framework/system/msg/config/sys_msg_config.h` provides configuration parameters that the implementer can use to adapt the System Messaging Service library to a particular application.

By default, the system message format provides 64 bits of information in a message:

```
typedef union
{
    struct
    {
        uint16_t    param0; // Message parameter zero
        uint16_t    param1; // Message parameter one
        uint16_t    param2; // Message parameter two
        uint16_t    nMessageTypeID; // Message type identifier
    };
    struct
    {
        uint16_t    nSource; // Message source identifier
        uintptr_t   * pData; // Pointer to additional message data
    };
} SYS_MSG_OBJECT;
```

The only required field in the message format definition is `nMessageTypeID`. But it doesn't need to be 16 bits long.

The maximum number of mailboxes is defined by:

```
#define SYS_MSG_MAX_MAILBOXES (32)
```

The minimum number of mailboxes is one.

The maximum number of message types is defined by:

```
#define SYS_MSG_MAX_TYPES (32)
```

The minimum number of message types is one.

Message priorities run from zero to [SYS_MSG_MAX_PRIORITY](#), which is defined by:

```
#define SYS_MSG_MAX_PRIORITY    (4)
```

For each message priority from 0,1, to [SYS_MSG_MAX_PRIORITY](#) a queue is created. The size of each queue is defined by:

```
//           Message Priority:  0   1   2   3   4
#define SYS_MSG_BUFFER_SIZES { 64, 32, 16,  8,  4 }
```

In this example there are five priorities, 0,1,..4, and the sizes of each message queue is provided in the definition of [SYS_MSG_BUFFER_SIZES](#).

Then this information is used when initializing the System Messaging Service in the application startup code:

```
SYS_OBJ_HANDLE hSysMsg;
{ //SYS_MSG_Initialize
    uint16_t nQSizes[] = SYS_MSG_BUFFER_SIZES;
    hSysMsg = SYS_MSG_Initialize(SYS_MSG_MAX_PRIORITY+1,nQSizes);
    SYS_ASSERT( SYS_OBJ_HANDLE_INVALID != hSysMsg, "Bad hSysMsg!" );
}
```

5.7.6.6 Building the Library

This section list the files that are available in the \src of the System Message Service Library driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.7.6.7 Library Interface

Data Types and Constants

	Name	Description
	SYS_MSG_INIT	Contains all the data necessary to initialize an instance of the System Messaging Service.
	SYS_MSG_INSTANCE	System Messaging instances numbering is from 0,1, to SYS_MSG_MAX_INSTANCE .
	SYS_MSG_OBJECT	This is type SYS_MSG_OBJECT .
	SYS_MSG_QUEUE_STATUS	Messaging queue status enumeration.
	SYS_MSG_RECEIVE_CALLBACK	Pointer to the System Message Received Callback Function
	SYS_MSG_RESULTS	Enumeration of message send results.
	SYS_MSGQ_ELEMENT	Defines queue element for message queue belonging to each priority.
	_SYS_MSG_H	This is macro _SYS_MSG_H .
	_SYS_MSG_OBJECT	Defines the contents of a system message.
	SYS_MSG_MAILBOXES_ADDONE	don't need to round up the number of bitmaps
	SYS_MSG_NUM_MAILBOX_BITMAPS	This is macro SYS_MSG_NUM_MAILBOX_BITMAPS .

Configuration Functions

	Name	Description
	SYS_MSG_BUFFER_SIZES	define SYS_MSG_BUFFER_SIZES { 4, 4, 4, 4, 1
	SYS_MSG_MAX_MAILBOXES	Specifies the maximum number of mailboxes possible.
	SYS_MSG_MAX_MSGS_DELIVERED	Specifies the maximum number of messages delivered per each call to SYS_MSG_Tasks .
	SYS_MSG_MAX_PRIORITY	Specifies the maximum message priority.
	SYS_MSG_MAX_TYPES	Specifies the maximum number of message types possible.

Initialization, Tasks, and Versioning Functions

	Name	Description
≡	<u>SYS_MSG_Deinitialize</u>	Deinitializes System Messaging Instance.
≡	<u>SYS_MSG_Initialize</u>	Configures and initializes the messaging subsystem.
≡	<u>SYS_MSG_Tasks</u>	System Messaging Service Tasks routine.
≡	<u>SYS_MSG_VersionGet</u>	Gets the System Messaging Service version in numerical format.
≡	<u>SYS_MSG_VersionStrGet</u>	Gets System Messaging Service version in string format.

Mailbox Functions

	Name	Description
≡	<u>SYS_MSB_MailboxClose</u>	Closes (destroys) a mailbox previously opened with <u>SYS_MSB_MailboxOpen</u> .
≡	<u>SYS_MSB_MailboxOpen</u>	Opens a system messaging mailbox.
≡	<u>SYS_MSB_MailboxReinit</u>	Reinitialize previously opened mailbox.
≡	<u>SYS_MSG_MailboxMessagesGet</u>	Get queued messages for a mailbox.
≡	<u>SYS_MSG_MailboxMsgAdd</u>	Adds a message type to the list of messages received by a mailbox.
≡	<u>SYS_MSG_MailboxMsgRemove</u>	Removes a message type from the list of messages received by a mailbox.

Message Send/Receive Functions

	Name	Description
≡	<u>SYS_MSG_GotMessages</u>	Returns true if system messaging has undelivered messages, false otherwise.
≡	<u>SYS_MSG_MessageDeliver</u>	Deliver message to mailboxes.
≡	<u>SYS_MSG_MessageReceive</u>	Receive next message in message queues.
≡	<u>SYS_MSG_MessageSend</u>	Sends a message, as defined by a message structure.

Message Type Functions

	Name	Description
≡	<u>SYS_MSG_TypeCreate</u>	Creates a new message type.
≡	<u>SYS_MSG_TypeRemove</u>	Removes an existing message type.

Utility Functions

	Name	Description
≡	<u>SYS_MSG_ID2hMsgType</u>	Translates message type identifier into handle of corresponding message type object.
≡	<u>SYS_MSG_QueueStatus</u>	Returns message queue status for a given message priority.

Description**5.7.6.7.1 Initialization, Tasks, and Versioning Functions****5.7.6.7.1.1 SYS_MSG_Deinitialize Function****C**

```
void SYS_MSG_Deinitialize(
    SYS_OBJ_HANDLE handleSysMsg
);
```

Description

Deinitializes System Messaging Instance and frees up allocated memory for it.

Preconditions

None.

Parameters

Parameters	Description
hSysMsg	handle to System Messaging Object for instance to be removed.

Returns

None.

Remarks

None.

Example

```
SYS_OBJ_HANDLE hSysMsg;
uint16_t nQSizes[] = SYS_MSG_BUFFER_SIZES;

// Initialize the messaging system. This needs to be done in the initialization code.
// Choose System Messaging instance that supports 8 byte messages
hSysMsg = SYS_MSG_Initialize(SYS_MSG_8Bytes, SYS_MSG_MAX_PRIORITY+1, nQSizes);

if (SYS_OBJ_HANDLE_INVALID == hSysMsg)
{
    // Handle error
}
.
.
.
// Remove this instance.
SYS_MSG_Deinitialize( hSysMsg );
```

5.7.6.7.1.2 SYS_MSG_Initialize Function**C**

```
SYS_OBJ_HANDLE SYS_MSG_Initialize(
    const SYS_MSG_INSTANCE iSysMsg,
    SYS_OBJ_HANDLE pInitializeSysMsg
);
```

Description

This routine configures and initializes the messaging subsystem appropriately for the current system design.

Preconditions

None.

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service to be initialized.
pInitSysMsg	Pointer to System Messaging initialization data structure. If NULL default config values are used.

Returns

Handle to the System Messaging object created. Returns `SYS_OBJ_HANDLE_INVALID` if allocation of data structure fails.

Returns [SYS_OBJ_HANDLE_INVALID](#) if pointer to initialization data structure is NULL.

Remarks

None.

Example

```
SYS_OBJ_HANDLE hSysMsg;  
SYS_MSG_INIT sInitSysMsg = { 0, (SYS_MSG_MAX_PRIORITY+1), { SYS_MSG_BUFFER_SIZES } };  
//uint16_t nQSizes[] = SYS_MSG_BUFFER_SIZES;  
  
// Initialize the messaging system. This needs to be done in the initialization code.  
// Choose System Messaging instance that supports 8 byte messages  
hSysMsg = SYS_MSG_Initialize(SYS_MSG_8Bytes,&sInitSysMsg);  
  
if (SYS_OBJ_HANDLE_INVALID == hSysMsg)  
{  
    // Handle error  
}
```

5.7.6.7.1.3 SYS_MSG_Tasks Function**C**

```
void SYS_MSG_Tasks(  
    SYS_OBJ_HANDLE handleSysMsg  
);
```

Description

System Messaging Service Tasks routine.

Preconditions

hSysMsg must have been returned from a call to [SYS_MSG_Initialize](#).

Parameters

Parameters	Description
hSysMsg	handle to System Messaging Object.

Returns

None.

Remarks

None.

Example

```
while ( SYS_MSG_GotMessages(iSysMsg) )  
{  
    SYS_MSG_Tasks(hSysMsg);  
}
```

5.7.6.7.1.4 SYS_MSG_VersionGet Function**C**

```
unsigned int SYS_MSG_VersionGet();
```

Description

This routine gets the System Messaging Service version. The version is encoded as major * 10000 + minor * 100 + patch. The stringized version can be obtained using [SYS_MSG_VersionStrGet](#)()

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for.

Returns

Current System Messaging Service version in numerical format.

Remarks

None.

Example

```
unsigned int version;

version = SYS_MSG_VersionGet();

if(version < 110200)
{
    // Do Something
}
```

5.7.6.7.1.5 SYS_MSG_VersionStrGet Function**C**

```
char * SYS_MSG_VersionStrGet();
```

Description

This routine gets the System Messaging Service version. The version is returned as major.minor.path[type], where type is optional. The numerical version can be obtained using [SYS_MSG_VersionGet\(\)](#)

Preconditions

None.

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to get the version for.

Returns

Current System Messaging Service version in the string format.

Remarks

None.

Example

```
char *version;

version = SYS_MSG_VersionStrGet();

printf("%s", version);
```

5.7.6.7.2 Mailbox Functions

5.7.6.7.2.1 SYS_MSB_MailboxClose Function

C

```
void SYS_MSB_MailboxClose(  
    SYS_OBJ_HANDLE hMailbox  
);
```

Description

Closes (destroys) a mailbox previously opened with [SYS_MSB_MailboxOpen](#).

Preconditions

hMailbox provided by call to [SYS_MSB_MailboxOpen](#).

Parameters

Parameters	Description
hMailbox	Handle to mailbox that is to be closed (destroyed).

Returns

None.

None.

Example

```
SYS_OBJ_HANDLE hMyMailbox;  
  
hMyMailbox = SYS_MSB_MailboxOpen( iSysMsg, &myCallBackFunction );  
SYS_MSB_MailboxClose(hMyMailbox);
```

5.7.6.7.2.2 SYS_MSB_MailboxOpen Function

C

```
SYS_OBJ_HANDLE SYS_MSB_MailboxOpen(  
    const SYS_MSG_INSTANCE iSysMsg,  
    SYS_MSG_RECEIVE_CALLBACK msgCallBackFunction  
);
```

Description

Opens a system messaging mailbox, providing a message callback function that is called whenever a message is received.

Preconditions

iSysMsg must have been used in a call to [SYS_MSG_Initialize](#). Message callback function will not be called until SYS_MSG_MailboxSignUp has been used to sign up the mailbox for messages of interest.

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
msgCallBackFunction	pointer to message callback function

Returns

Handle to new system messaging mailbox.

Remarks

A null callback function disables messaging callbacks.

Example

```

SYS_OBJ_HANDLE hMyMailbox;

hMyMailbox = SYS_MSB_MailboxOpen( iSysMsg, &myCallBackFunction );
if (SYS_OBJ_HANDLE_INVALID == SYS_MSG_TypeRemove)
{
    // Handle error
}

```

5.7.6.7.2.3 SYS_MSB_MailboxReinit Function**C**

```

void SYS_MSB_MailboxReinit(
    SYS_OBJ_HANDLE hMailbox,
    SYS_MSG_RECEIVE_CALLBACK msgCallBackFunction
);

```

Description

Reinitialize previously opened mailbox by providing new call back function and clearing all message type assignments.

Preconditions

hMailbox provided by call to [SYS_MSB_MailboxOpen](#).

Parameters

Parameters	Description
hMailbox	Object handle to mailbox
msgCallBackFunction	pointer to new message callback function

Remarks

A null callback function disables messaging callbacks.

Example

```

SYS_OBJ_HANDLE hMyMailbox;

hMyMailbox = SYS_MSB_MailboxOpen( iSysMsg, &myCallBackFunction );
SYS_MSB_MailboxReinit(hMyMailbox,&anotherCallBackFunction );

```

5.7.6.7.2.4 SYS_MSG_MailboxMessagesGet Function**C**

```

SYS_MSG_OBJECT * SYS_MSG_MailboxMessagesGet(
    SYS_OBJ_HANDLE hMailbox
);

```

Description

Get queued messages for a mailbox. Messages returned by this function will not be received via the mailbox's callback function.

Preconditions

hMailbox provided by call to [SYS_MSB_MailboxOpen](#).

Parameters

Parameters	Description
hMailbox	Object handle to mailbox

Returns

Pointer to next message in the queue(s) that is of interest to the mailbox. Function returns NULL if no messages are found in the

queue.

Remarks

None.

Example

```
SYS_MSG_OBJECT *pNextMessage;
while ( NULL != (pNextMessage = SYS_MSG_MailboxMessagesGet(hMyMailbox)) )
{
    // Process message at *pNextMessage.
}
```

5.7.6.7.2.5 SYS_MSG_MailboxMsgAdd Function

C

```
void SYS_MSG_MailboxMsgAdd(
    SYS_OBJ_HANDLE hMailbox,
    SYS_OBJ_HANDLE hMsgType
);
```

Description

Adds a message type to the list of messages received by a mailbox.

Preconditions

hMailbox provided by call to [SYS_MSB_MailboxOpen](#).

Parameters

Parameters	Description
hMailbox	Object handle to mailbox
hMsgType	Handle to message type of interest for this mailbox.

Returns

None.

Remarks

When the message type handle is unknown but the message ID is known use [SYS_MSG_ID2hMsgType](#) to provide the message type handle. See code example.

Example

```
const SYS_MSG_INSTANCE iSysMsg;
SYS_OBJ_HANDLE hMsgType[5], hMailbox[3];

// Create three mailboxes
hMailbox[0] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack0 );
hMailbox[1] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack1 );
hMailbox[2] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack2 );

// Create five message types
//                                     Message ID: :Priority
hMsgType[0] = SYS_MSG_TypeCreate(iSysMsg,1<<0,0);
hMsgType[1] = SYS_MSG_TypeCreate(iSysMsg,1<<1,1);
hMsgType[2] = SYS_MSG_TypeCreate(iSysMsg,1<<2,2);
hMsgType[3] = SYS_MSG_TypeCreate(iSysMsg,1<<3,3);
hMsgType[4] = SYS_MSG_TypeCreate(iSysMsg,1<<4,4);

// Add messages to each mailbox
SYS_MSG_MailboxMsgAdd(hMailbox[0],hMsgType[0]);
SYS_MSG_MailboxMsgAdd(hMailbox[1],hMsgType[1]);
SYS_MSG_MailboxMsgAdd(hMailbox[2],hMsgType[2]);
```

```

SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<1));
SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<2));
SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<3));
SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<4));

```

5.7.6.7.2.6 SYS_MSG_MailboxMsgRemove Function

C

```

void SYS_MSG_MailboxMsgRemove(
    SYS_OBJ_HANDLE hMailbox,
    SYS_OBJ_HANDLE hMsgType
);

```

Description

Removes a message type from the list of messages received by a mailbox.

Preconditions

hMailbox provided by call to [SYS_MSB_MailboxOpen](#).

Parameters

Parameters	Description
hMailbox	Object handle to mailbox
hMsgType	Handle to message type to be ignored by this mailbox.

Returns

None.

Remarks

When the message type handle is unknown but the message ID is known use [SYS_MSG_ID2hMsgType](#) to provide the message type handle. See code example.

Example

```

const SYS_MSG_INSTANCE iSysMsg;
SYS_OBJ_HANDLE hMsgType[5], hMailbox[3];

// Create three mailboxes
hMailbox[0] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack0 );
hMailbox[1] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack1 );
hMailbox[2] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack2 );

// Create five message types
//                                     Message ID:  :Priority
hMsgType[0] = SYS_MSG_TypeCreate(iSysMsg, 1<<0, 0);
hMsgType[1] = SYS_MSG_TypeCreate(iSysMsg, 1<<1, 1);
hMsgType[2] = SYS_MSG_TypeCreate(iSysMsg, 1<<2, 2);
hMsgType[3] = SYS_MSG_TypeCreate(iSysMsg, 1<<3, 3);
hMsgType[4] = SYS_MSG_TypeCreate(iSysMsg, 1<<4, 4);

// Add messages to each mailbox
SYS_MSG_MailboxMsgAdd(hMailbox[0], hMsgType[0]);
SYS_MSG_MailboxMsgAdd(hMailbox[1], hMsgType[1]);
SYS_MSG_MailboxMsgAdd(hMailbox[2], hMsgType[2]);

SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<1));
SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<2));
SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<3));
SYS_MSG_MailboxMsgAdd(hMailbox[0], SYS_MSG_ID2hMsgType(iSysMsg, 1<<4));

```

5.7.6.7.3 Message Type Functions

5.7.6.7.3.1 SYS_MSG_TypeCreate Function

C

```
SYS_OBJ_HANDLE SYS_MSG_TypeCreate(  
    const SYS_MSG_INSTANCE iSysMsg,  
    uint8_t nMessageTypeID,  
    uint8_t nMessagePriority  
);
```

Description

Creates a new message type, defining an integer message type and priority.

Preconditions

iSysMsg must have been used in a call to [SYS_MSG_Initialize](#).

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
nMessageTypeID	Integer message type identifier
nMessagePriority	Message priority, between 0 and SYS_MSG_MAX_PRIORITIES

Returns

Handle to new message type definition.

Remarks

None.

Example

```
SYS_OBJ_HANDLE hMsgType;  
hMsgType = SYS_MSG_TypeCreate( iSysMsg, 1, 3 );  
if (SYS_OBJ_HANDLE_INVALID == hMsgType)  
{  
    // Handle error  
}
```

5.7.6.7.3.2 SYS_MSG_TypeRemove Function

C

```
void SYS_MSG_TypeRemove(  
    SYS_OBJ_HANDLE hMsgType  
);
```

Description

Removes an existing message type.

Preconditions

None.

Parameters

Parameters	Description
hMsgType	Handle to message type that is to be removed

Returns

None.

Remarks

When the message type handle is unknown but the message ID is known use [SYS_MSG_ID2hMsgType](#) to provide the message type handle. See code example.

Example

```
SYS_OBJ_HANDLE hMsgType;
hMsgType = SYS_MSG_TypeCreate( iSysMsg, 1, 3 );
SYS_MSG_TypeRemove( hMsgType );
```

Alternately:

```
SYS_OBJ_HANDLE hMsgType;
hMsgType = SYS_MSG_TypeCreate( iSysMsg, 1, 3 );
SYS_MSG_TypeRemove( SYS_MSG_ID2hMsgType(iSysMsg,1) );
```

5.7.6.7.4 Message Send/Receive Functions**5.7.6.7.4.1 SYS_MSG_GotMessages Function****C**

```
bool SYS_MSG_GotMessages(
    const SYS_MSG_INSTANCE iSysMsg
);
```

Description

Returns true if system messaging has undelivered messages, false otherwise.

Preconditions

iSysMsg must have been used in a call to [SYS_MSG_Initialize](#).

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.

Returns

True if there are undelivered system messages, false otherwise.

Remarks

None.

Example

```
DBPRINTF("rnGot Messages: %drn",SYS_MSG_GotMessages(iSysMsg));
while ( (pNextMessage = SYS_MSG_MessageReceive(iSysMsg)) != NULL )
{
    SYS_MSG_MessageDeliver(iSysMsg,pNextMessage);
    DBPRINTF("rn");
}
DBPRINTF("Got Messages: %drn",SYS_MSG_GotMessages(ihSysMsg));
```

5.7.6.7.4.2 SYS_MSG_MessageDeliver Function**C**

```
void SYS_MSG_MessageDeliver(
    const SYS_MSG_INSTANCE iSysMsg,
    SYS_MSGQ_ELEMENT * pQElement
);
```

Description

Deliver message to mailboxes, remove message from queue when done.

Preconditions

iSysMsg must have been used in a call to [SYS_MSG_Initialize](#).

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
pQElement	pointer to queue element to be delivered.

Returns

None.

Remarks

None.

Example

```
SYS_MSGQ_ELEMENT *pQElement;  
SYS_MSG_OBJECT nextMessage;  
while ( (pQElement = SYS_MSG_MessageReceive(iSysMsg)) != NULL )  
{  
    // In case you desire to examine message before delivering it.  
    nextMessage = pQElement->sMessage;  
  
    // Deliver message to all interested mailboxes  
    SYS_MSG_MessageDeliver(iSysMsg,pQElement);  
}
```

5.7.6.7.4.3 SYS_MSG_MessageReceive Function**C**

```
SYS_MSGQ_ELEMENT * SYS_MSG_MessageReceive(  
    const SYS_MSG_INSTANCE iSysMsg  
);
```

Description

Receive next message in message queues, returning NULL if queues are empty.

Preconditions

iSysMsg must have been used in a call to [SYS_MSG_Initialize](#).

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.

Returns

Pointer to next message, as found in a message queue element, NULL if there are no messages.

Remarks

None.

Example

```
SYS_MSGQ_ELEMENT *pQElement;  
SYS_MSG_OBJECT nextMessage;  
while ( (pQElement = SYS_MSG_MessageReceive(iSysMsg)) != NULL )
```

```
{
    // In case you desire to examine message before delivering it.
    nextMessage = pQElement->sMessage;

    // Deliver message to all interested mailboxes
    SYS_MSG_MessageDeliver(iSysMsg, pQElement);
}
```

5.7.6.7.4.4 **SYS_MSG_MessageSend Function**

C

```
SYS_MSG_RESULTS SYS_MSG_MessageSend(
    const SYS_MSG_INSTANCE iSysMsg,
    SYS_MSG_OBJECT * pMessage
);
```

Description

Sends a message, as defined by a message structure.

Preconditions

iSysMsg must have been used in a call to [SYS_MSG_Initialize](#).

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
pMessage	Pointer to message definition.

Returns

Message result from [SYS_MSG_RESULTS](#) enumeration.

Remarks

None.

Example

```
SYS_MSG_OBJECT myMessage;
SYS_MSG_RESULTS myMessageStatus;
SYS_OBJ_HANDLE hMyMsgType;

hMyMsgType = SYS_MSG_TypeCreate( iSysMsg, 1, 3 );
if (SYS_OBJ_HANDLE_INVALID == hMyMsgType)
{
    // Handle error
}

myMessage.hMsgType = hMyMsgType;
myMessage.nSource = myMsgSource;
myMessage.param1 = parameterOneValue;
myMessage.param1 = parameterTwoValue;

myMessageStatus = SYS_MSG_MessageSend(iSysMsg, &myMessage);
SYS_ASSERT( myMessageStatus > 0, "Bad message status!" );
```

5.7.6.7.5 **Utility Functions**

5.7.6.7.5.1 **SYS_MSG_ID2hMsgType Function**

C

```
SYS_OBJ_HANDLE SYS_MSG_ID2hMsgType(
```



```
    const SYS_MSG_INSTANCE iSysMsg,  
    uint8_t nMessageTypeID  
);
```

Description

Translates message type identifier into handle of corresponding message type object.

Preconditions

iSysMsg must have been used in a call to [SYS_MSG_Initialize](#).

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
nMessageTypeID	Integer message type identifier.

Returns

Handle to message type definition object corresponding to the message type identifier. Returns NULL if the message type is not defined.

Remarks

This function is useful in situations where the message type identifier is known but the message type handle is not. This allows applications to statically define message type IDs and use them in code instead of having to wait until message handles are known and then dynamically sharing message handles via global variables.

Example

```
#define MY_MESSAGE_TYPE_ID 1;  
  
// Create message type  
SYS_OBJ_HANDLE hMsgType;  
uint8_t nPriority = 3;  
hMsgType = SYS_MSG_TypeCreate( iSysMsg, MY_MESSAGE_TYPE_ID, nPriority );  
if (SYS_OBJ_HANDLE_INVALID == hMsgType)  
{  
    // Handle error  
}  
.  
.  
.  
// Remove message type without knowing message type handle  
SYS_MSG_TypeRemove( iSysMsg, SYS_MSG_ID2hMsgType( iSysMsg, MY_MESSAGE_TYPE_ID ) );
```

5.7.6.7.5.2 SYS_MSG_QueueStatus Function

C

```
SYS_MSG_QUEUE_STATUS SYS_MSG_QueueStatus(  
    const SYS_MSG_INSTANCE iSysMsg,  
    uint8_t nMessagePriority  
);
```

Description

Returns message queue status for a given message priority.

Preconditions

iSysMsg must have been used in a call to [SYS_MSG_Initialize](#).

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
nMessagePriority	message priority of interest, from zero to SYS_MSG_MAX_PRIORITIES.

Returns

Number of messages in queue if not full or SYS_MSG_QUEUE_FULL if full. If message priority is not legal, returns SYS_MSG_QUEUE_BAD.

Remarks

None.

Example

5.7.6.7.6 Configuration Functions

5.7.6.7.6.1 SYS_MSG_BUFFER_SIZES Macro

```
C
#define SYS_MSG_BUFFER_SIZES { 16 }
```

Description

define SYS_MSG_BUFFER_SIZES { 4, 4, 4, 4, 1

5.7.6.7.6.2 SYS_MSG_MAX_MAILBOXES Macro

```
C
#define SYS_MSG_MAX_MAILBOXES (2)
```

Description

System Messaging Max Number of Mailboxes
Specifies the maximum number of mailboxes possible.

Remarks

Minimum number is 1 mailbox.

5.7.6.7.6.3 SYS_MSG_MAX_MSGS_DELIVERED Macro

```
C
#define SYS_MSG_MAX_MSGS_DELIVERED (1)
```

Description

System Messaging Maximum Number of Messages Delivered per call to [SYS_MSG_Tasks](#)
Specifies the maximum number of messages delivered per each call to [SYS_MSG_Tasks](#). If zero then all message queues are emptied before the tasks routine finishes execution.

Remarks

0 implies all queues are empty after [SYS_MSG_Tasks](#) is done.

5.7.6.7.6.4 SYS_MSG_MAX_PRIORITY Macro

C

```
#define SYS_MSG_MAX_PRIORITY (0)
```

Description

System Messaging Maximum Priority

Specifies the maximum message priority.

Remarks

Message priorities run from 0 to SYS_MSG_MAX_PRIORITIES. The number of message queues is SYS_MSG_MAX_PRIORITIES+1.

5.7.6.7.6.5 SYS_MSG_MAX_TYPES Macro

C

```
#define SYS_MSG_MAX_TYPES (2)
```

Description

System Messaging Max Number of Message Types

Specifies the maximum number of message types possible.

Remarks

Minimum number is 1.

5.7.6.7.7 Data Types and Constants

5.7.6.7.7.1 SYS_MSG_INIT Structure

C

```
typedef struct {
    uint8_t nMaxMsgsDelivered;
    uint8_t nMessagePriorities;
    uint16_t * nQSizes;
} SYS_MSG_INIT;
```

Description

System Messaging Service Initialization Data

Contains all the data necessary to initialize an instance of the System Messaging Service.

Members

Members	Description
uint8_t nMaxMsgsDelivered;	Maximum number of messages delivered per call to SYS_MSG_Tasks
uint8_t nMessagePriorities;	Number of message priorities desired
uint16_t * nQSizes;	Array of qukeue sizes for priorities 0,1,... SYS_MSG_MAX_PRIORITY

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the [SYS_MSG_Initialize](#) routine. If nMaxMsgsDelivered == 0 then ALL messages in priority queues are delivered each time [SYS_MSG_Tasks](#) is called.

5.7.6.7.7.2 SYS_MSG_INSTANCE Enumeration

C

```
typedef enum {
    SYS_MSG_0,
    SYS_MSG_1,
    SYS_MSG_2,
    SYS_MSG_3,
    SYS_MSG_4,
    SYS_MSG_NUM_INSTANCES
} SYS_MSG_INSTANCE;
```

Description

Enumeration of the Allowable of System Messaging Instances

System Messaging instances numbering is from 0,1, to SYS_MSG_MAX_INSTANCE.

Remarks

None.

5.7.6.7.7.3 SYS_MSG_OBJECT Structure

C

```
typedef struct {
    union {
        struct {
            uint8_t nMessageTypeID;
            uint8_t nSource;
            uint16_t param0;
            uint16_t param1;
            uint16_t param2;
        }
        struct {
            uint16_t dummy;
            uint16_t nSizeData;
            uintptr_t * pData;
        }
    }
} SYS_MSG_OBJECT;
```

Description

This is type SYS_MSG_OBJECT.

Members

Members	Description
uint8_t nMessageTypeID;	Message type identifier
uint8_t nSource;	Message source identifier
uint16_t param0;	Message parameter zero
uint16_t param1;	Message parameter one
uint16_t param2;	Message parameter two
uint16_t nSizeData;	Size of data that pData identifies
uintptr_t * pData;	Pointer to additional message data

5.7.6.7.7.4 SYS_MSG_QUEUE_STATUS Enumeration

C

```
typedef enum {
    SYS_MSG_QUEUE_BAD,
    SYS_MSG_QUEUE_FULL,
    SYS_MSG_QUEUE_EMPTY
} SYS_MSG_QUEUE_STATUS;
```

Description

System Messaging Queue Status Enumeration

Messaging queue status enumeration. Positive values indicate number of messages in the queue.

Members

Members	Description
SYS_MSG_QUEUE_BAD	QUEUE Status: full
SYS_MSG_QUEUE_FULL	QUEUE Status: full
SYS_MSG_QUEUE_EMPTY	QUEUE Status: empty

Remarks

None.

5.7.6.7.7.5 SYS_MSG_RECEIVE_CALLBACK Type

C

```
typedef void (* SYS_MSG_RECEIVE_CALLBACK)(SYS_MSG_OBJECT *pMessage);
```

Description

Pointer to the System Message Received Callback Function

Pointer to the function provided for each system messaging mailbox that is called when a system message is received for each mailbox.

5.7.6.7.7.6 SYS_MSG_RESULTS Enumeration

C

```
typedef enum {
    SYS_MSG_NOT_SENT_QFULL,
    SYS_MSG_BAD_PRIORITY,
    SYS_MSG_BAD_MSGTYPE,
    SYS_MSG_NOT_SENT,
    SYS_MSG_SENT
} SYS_MSG_RESULTS;
```

Description

System Messaging Results Enumeration

Enumeration of message [send](#) results.

Members

Members	Description
SYS_MSG_NOT_SENT_QFULL	Message could not be sent, no room available in priority queues
SYS_MSG_BAD_PRIORITY	Message could not be sent, Message Message priority bad
SYS_MSG_BAD_MSGTYPE	Message could not be sent, Message type bad

SYS_MSG_NOT_SENT	Message could not be sent, no other information available
SYS_MSG_SENT	Message sent

Remarks

SYS_MSG_SENT aligns with SYS_MSGQ_Success. SYS_MSG_NOT_SENT aligns with SYS_MSGQ_Failure

5.7.6.7.7.7 SYS_MSGQ_ELEMENT Structure

C

```
typedef struct {
    SYS_MSG_OBJECT sMessage;
    uint16_t mailboxInterestBitMap[SYS_MSG_NUM_MAILBOX_BITMAPS];
} SYS_MSGQ_ELEMENT;
```

Description

System Messaging Queues Element

Defines queue element for message queue belonging to each priority.

Members

Members	Description
SYS_MSG_OBJECT sMessage;	System Message Bit map for mailboxes interested in this message type, modified as each mailbox is notified.

Remarks

None

5.7.6.7.7.8 _SYS_MSG_H Macro

C

```
#define _SYS_MSG_H
```

Description

This is macro _SYS_MSG_H.

5.7.6.7.7.9 _SYS_MSG_OBJECT Macro

C

```
#define _SYS_MSG_OBJECT
```

Description

System Message Format

Defines the contents of a system message. This information is copied into an element of the system message queue identified by the message priority.

Remarks

None

5.7.6.7.7.10 SYS_MSG_MAILBOXES_ADDONE Macro

C

```
#define SYS_MSG_MAILBOXES_ADDONE 0
```

Description

don't need to round up the number of bitmaps

5.7.6.7.7.11 SYS_MSG_NUM_MAILBOX_BITMAPS Macro

C

```
#define SYS_MSG_NUM_MAILBOX_BITMAPS (SYS_MSG_MAX_MAILBOXES/16 + SYS_MSG_MAILBOXES_ADDONE)
```

Description

This is macro SYS_MSG_NUM_MAILBOX_BITMAPS.

5.7.6.8 Files**Files**

Name	Description
sys_msg.h	System Service for the messaging module.
sys_msg_config.h	System Messaging Configuration definitions file

Description**5.7.6.8.1 sys_msg.h**

Messaging System-Library Interface Definition

This file contains the interface definition for the messaging system service. It provides a way to interact with the messaging subsystem.

Enumerations

	Name	Description
	SYS_MSG_INSTANCE	System Messaging instances numbering is from 0,1, to SYS_MSG_MAX_INSTANCE.
	SYS_MSG_QUEUE_STATUS	Messaging queue status enumeration.
	SYS_MSG_RESULTS	Enumeration of message send results.

Functions

	Name	Description
	SYS_MSB_MailboxClose	Closes (destroys) a mailbox previously opened with SYS_MSB_MailboxOpen .
	SYS_MSB_MailboxOpen	Opens a system messaging mailbox.
	SYS_MSB_MailboxReinit	Reinitialize previously opened mailbox.
	SYS_MSG_Deinitialize	Deinitializes System Messaging Instance.
	SYS_MSG_GotMessages	Returns true if system messaging has undelivered messages, false otherwise.
	SYS_MSG_ID2hMsgType	Translates message type identifier into handle of corresponding message type object.
	SYS_MSG_Initialize	Configures and initializes the messaging subsystem.
	SYS_MSG_MailboxMessagesGet	Get queued messages for a mailbox.
	SYS_MSG_MailboxMsgAdd	Adds a message type to the list of messages received by a mailbox.
	SYS_MSG_MailboxMsgRemove	Removes a message type from the list of messages received by a mailbox.

	SYS_MSG_MessageDeliver	Deliver message to mailboxes.
	SYS_MSG_MessageReceive	Receive next message in message queues.
	SYS_MSG_MessageSend	Sends a message, as defined by a message structure.
	SYS_MSG_QueueStatus	Returns message queue status for a given message priority.
	SYS_MSG_Tasks	System Messaging Service Tasks routine.
	SYS_MSG_TypeCreate	Creates a new message type.
	SYS_MSG_TypeRemove	Removes an existing message type.
	SYS_MSG_VersionGet	Gets the System Messaging Service version in numerical format.
	SYS_MSG_VersionStrGet	Gets System Messaging Service version in string format.

Macros

	Name	Description
	_SYS_MSG_H	This is macro _SYS_MSG_H.
	_SYS_MSG_OBJECT	Defines the contents of a system message.
	SYS_MSG_MAILBOXES_ADDONE	don't need to round up the number of bitmaps
	SYS_MSG_NUM_MAILBOX_BITMAPS	This is macro SYS_MSG_NUM_MAILBOX_BITMAPS.
	SYS_OBJ_HANDLE_INVALID	This is macro SYS_OBJ_HANDLE_INVALID.
	SYS_OBJ_HANDLE_STATIC	This is macro SYS_OBJ_HANDLE_STATIC.

Structures

	Name	Description
	SYS_MSG_INIT	Contains all the data necessary to initialize an instance of the System Messaging Service.
	SYS_MSG_OBJECT	This is type SYS_MSG_OBJECT.
	SYS_MSGQ_ELEMENT	Defines queue element for message queue belonging to each priority.

Types

	Name	Description
	SYS_MSG_RECEIVE_CALLBACK	Pointer to the System Message Received Callback Function
	SYS_OBJ_HANDLE	SYS_MODULE_OBJ is badly named. It should be SYS_MODULE_OBJ_HANDLE or something shorter. For brevity, it is renamed to SYS_OBJ_HANDLE.

File Name

sys_msg.h

Company

Microchip Technology Incorporated

5.7.6.8.2 sys_msg_config.h

System Messaging Configuration Definitions

These definitions statically define the operation of the System Messaging service.

Macros

	Name	Description
	SYS_MSG_BUFFER_SIZES	define SYS_MSG_BUFFER_SIZES { 4, 4, 4, 4, 1
	SYS_MSG_MAX_MAILBOXES	Specifies the maximum number of mailboxes possible.

	SYS_MSG_MAX_MSGS_DELIVERED	Specifies the maximum number of messages delivered per each call to SYS_MSG_Tasks .
	SYS_MSG_MAX_PRIORITY	Specifies the maximum message priority.
	SYS_MSG_MAX_TYPES	Specifies the maximum number of message types possible.

File Name

sys_msg_config.h

Company

Microchip Technology Incorporated

5.7.7 Ports System Service Library

5.7.7.1 Introduction

Ports System Service Library

for

Microchip Microcontrollers

This library provides an interface to manage and control general purpose input or output ports (GPIO) controlled by the PORTx modules on the Microchip families of microcontrollers.

Overview

One challenge designers of general purpose microcontroller devices face is to provide a large set of available peripheral features on parts with a limited number of I/O pins. To help meet this challenge and to provide flexibility for board designers, many parts provide an ability to route I/O signals for selected peripherals to different I/O pins. Still, in some cases, general purpose I/O pins must be used to implement the desired functionality under direct software control. The purpose of the ports system service is to provide direct control of general purpose I/O pins and to support the selection of the desired peripheral functionality for supported I/O pins and ports.

General purpose I/O pins can be controlled individually, but they are also grouped together in sets and can be controlled together as a unit. These groups of GPIO pins are called "ports" on Microchip microcontrollers and this library provides the ability to read and write data patterns to or from them in sets or as individual pins. In addition peripheral IO routing and pin/port control, this library provides the ability to select and configure several other features of the I/O pins and ports available on Microchip microcontrollers as described below.

Other Features

- Individual output pin/port open-drain enable/disable
- Individual input pin/port pull-up enable/disable
- Monitor select inputs and generate interrupt on mismatch condition [Change Notification]
- Operate during CPU Sleep and Idle modes
- Port line Analog/Digital Selection

- Port slew rate control

Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software.

5.7.7.2 Release Notes

MPLAB Harmony Version: v0.70b **Ports System Service Library Library Version :** 0.01 **Release Date:** %ReleaseNotes%

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.7.7.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED AS IS MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.7.7.4 Using the Library

This topic describes the basic architecture of the Ports System Service Library and provides information and examples on how to use it.

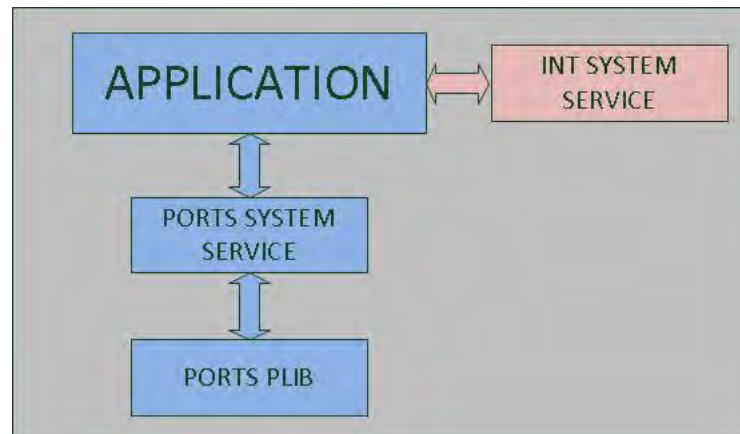
Interface Header File: [sys_ports.h](#)

The interface to the Ports System Service Library is defined in the "[sys_ports.h](#)" header file, which is included by the "sys.h" header file.

Any C language source (.c) file that uses the Ports System Service must include "sys.h".

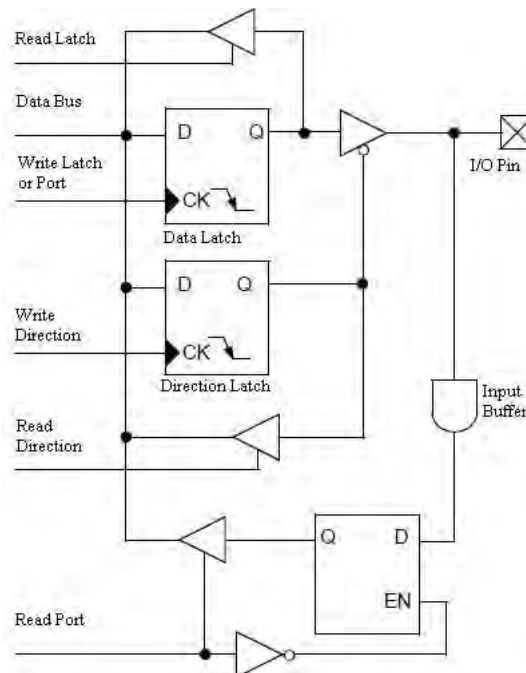
5.7.7.4.1 Abstraction Model

This model explains how the system interfaces with the Ports System Service and the application as depicted in the below diagram.



General Purpose I/O:

All port pins have three registers directly associated with their operation as digital I/O. The Data Direction register determines whether the pin is an input or an output. If the data direction bit is a '1', then the pin is an input. All port pins are defined as inputs after a Reset. Reads from the Output Latch register, read the latch. Writes to the latch, write the latch. Reads from the port, read the port pins, while writes to the port pins, write the latch.



The *pull-ups* act as a current source or sink source connected to the pin, and eliminates the need for external resistors when push-button or keypad devices are connected. These *pull-ups* prevent floating state of the pins by providing voltage to it. These features are available on some pins and some parts. Please check the data sheet of the specific device for further information.

The *open-drain* feature allows the generation of outputs higher than VDD (e.g., 5V) on any desired digital only pins by using external pull-up resistors. The maximum open-drain voltage allowed is the same as the maximum VIH specification.

The *input change notification* function of the I/O ports allows the microcontrollers to generate interrupt requests to the processor in response to a change of state on selected input pins. This feature is capable of detecting input change of states even in Sleep mode, when the clocks are disabled.

The *Alternate Pin Function selections* are used to steer specific peripheral input and output functions between different pins.

The *output slew rate* of each port is programmable to select either the standard transition rate or a reduced transition rate of x times the standard to minimize EMI. The reduced transition time is the default slew rate for all ports.

Peripheral Pin Select:

AVAILABLE PINS:

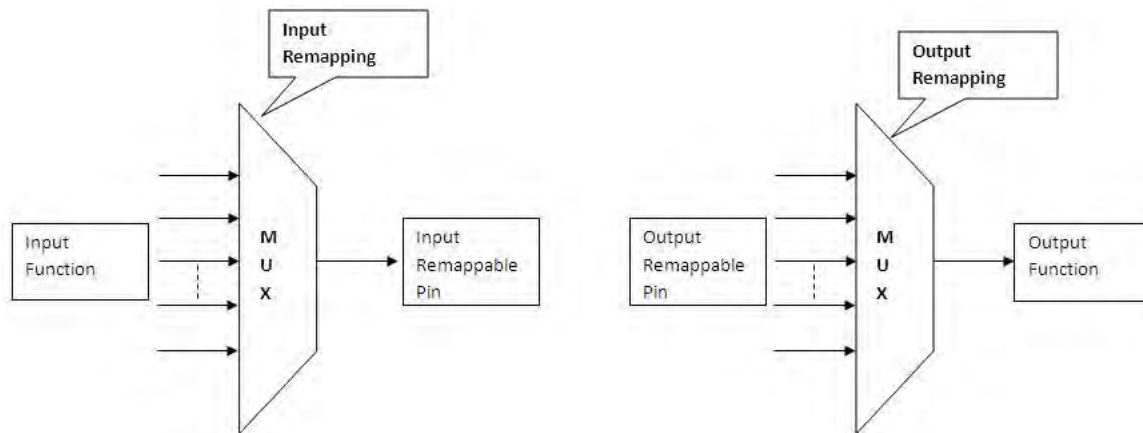
The number of available pins is dependent on the particular device and its pin count. Pins that support the peripheral pin select feature include the designation “RPn” in their full pin designation, where “RP” designates a remappable peripheral and “n” is the remappable port number.

AVAILABLE PERIPHERALS:

The peripherals managed by the peripheral pin select are all digital-only peripherals. These include general serial communications (UART and SPI), general purpose timer clock inputs, timer-related peripherals (input capture and output compare) and interrupt-on-change inputs.

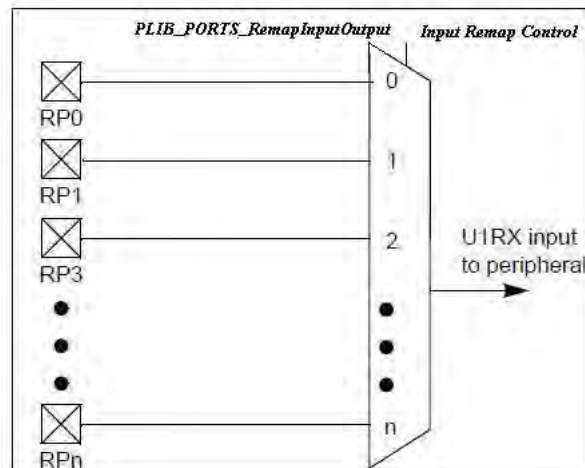
In comparison, some digital-only peripheral modules are never included in the peripheral pin select feature. This is because the peripheral's function requires special I/O circuitry on a specific port and cannot be easily connected to multiple pins. These modules include I2C among others. A similar requirement excludes all modules with analog inputs, such as the A/D converter. A key difference between remappable and non-remappable peripherals is that remappable peripherals are not associated with a default I/O pin. The peripheral must always be assigned to a specific I/O pin before it can be used. In contrast, non-remappable peripherals are always available on a default pin, assuming that the peripheral is active and not conflicting with another peripheral.

When a remappable peripheral is active on a given I/O pin, it takes priority over all other digital I/O and digital communication peripherals associated with the pin. Priority is given regardless of the type of peripheral that is mapped. Remappable peripherals never take priority over any analog functions associated with the pin.



INPUT MAPPING:

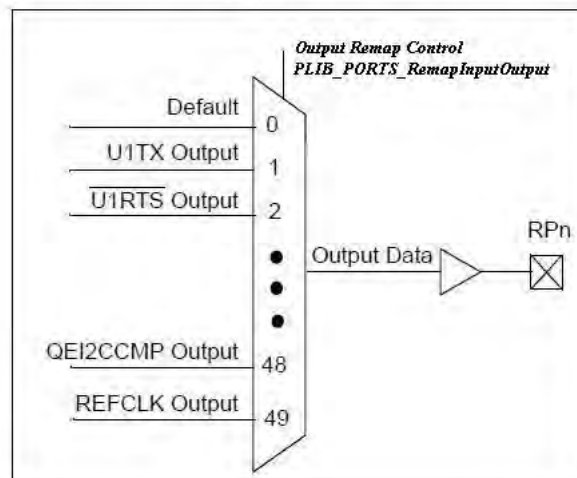
The inputs of the peripheral pin select options are mapped on the basis of the peripheral. That is, a control register associated with a peripheral dictates the pin it will be mapped to.

Remappable Input Example for USART 1 Receive

Note: For input only, peripheral pin select functionality does not have priority over I/O port settings. Therefore, when configuring Remappable Pin for input, the corresponding bit in the I/O port register must also be configured for input (set to '1').

OUTPUT MAPPING:

In contrast to inputs, the outputs of the peripheral pin select options are mapped on the basis of the pin. In this case, a control register associated with a particular pin dictates the peripheral output to be mapped.

Multiplexing of Remappable Output for RPn**Mapping Limitations:**

The control schema of the peripheral select pins is not limited to a small range of fixed peripheral configurations. There are no mutual or hardware-enforced lockouts between any of the peripheral mapping SFRs. Literally any combination of peripheral mappings across any or all of the RPn pins is possible. This includes both many-to-one and one-to-many mappings of peripheral inputs and outputs to pins. While such mappings may be technically possible from a configuration point of view, they may not be supportable from an electrical point of view.

5.7.7.4.2 Library Overview

The Ports System Service Library is a MPLAB Harmony system service. Please refer to the system services documentation for a detailed description of MPLAB system services.

The library interface routines are divided into various subsections, each subsection addresses one of the blocks or the overall operation of the PORTx module.

Library Interface Section	Description
Pin Control Functions	Port bit/pin read/write/toggle/clear/set interfaces.
Ports Control Functions	Port access read/write/toggle/clear/set interfaces.
Change Notification Functions	Interface routines for Port line change notification.
Peripheral Pin Select Functions	Interface routines for mapping the digital input/output to a specific PPS Remappable input/output pin.

5.7.7.4.3 How the Library Works

5.7.7.4.3.1 Pin Control

Pins Functions Usage

Pin Read:

Port pin can be read at bit/pin level using "[SYS_PORTS_PinRead](#)" with appropriate parameters.

Pin Write:

Port pin can be written at bit/pin level using "[SYS_PORTS_PinWrite](#)" with appropriate parameters.

Pin Clear:

Port pin can be cleared at bit/pin level using "[SYS_PORTS_PinClear](#)" with appropriate parameters.

Pin Set:

Port pin can be set at bit/pin level using "[SYS_PORTS_PinSet](#)" with appropriate parameters.

Pin Direction Control:

Port pin direction can be set at bit/pin level using "[SYS_PORTS_PinDirectionSelect](#)" with appropriate parameters.

The Direction information can be obtained through the interface "[SYS_PORTS_DirectionGet](#)".

Pin Toggle:

Port pin can be toggled at bit/pin level using "[SYS_PORTS_PinToggle](#)" with appropriate parameters.

Pin Open Drain:

Port pin can be enabled open drain functionality at bit/pin level using "[SYS_PORTS_PinOpenDrainEnable](#)" with appropriate parameters. Similarly, the Port pin can be disabled open drain functionality at bit/pin level using "[SYS_PORTS_PinOpenDrainDisable](#)" with appropriate parameters.

Pin Pull up:

Port pin can be enabled pull up functionality at bit/pin level using "[SYS_PORTS_PinPullUpEnable](#)" with appropriate parameters. Similarly, the Port pin can be disabled pull up functionality at bit/pin level using "[SYS_PORTS_PinPullUpDisable](#)" with appropriate parameters.

Example:

```
// PORT Direction setting for output
SYS_PORTS_PinDirectionSelect(MY_PORTS_INSTANCE, SYS_PORTS_DIRECTION_OUTPUT, MY_CHANNEL,
MY_PINNUM);
// PORT Direction setting for input
SYS_PORTS_PinDirectionSelect(MY_PORTS_INSTANCE, SYS_PORTS_DIRECTION_INPUT, MY_CHANNEL,
MY_PINNUM);
// Writing a value into a PORT
SYS_PORTS_PinWrite(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM, MY_VALUE);
// Reading back the previously written value
bool readData = SYS_PORTS_PinRead(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
// Clearing the PORT
SYS_PORTS_PinClear(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
// Setting the port
SYS_PORTS_PinSet(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
// Toggling a PORT
SYS_PORTS_PinToggle(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

Note: Not all the features are available on all devices. Please refer to the specific device data sheet to determine availability.

5.7.7.4.3.2 Ports Control

Port Functions Usage**Port Read:**

Ports can be read at byte/word level using the interface "[SYS_PORTS_Read](#)" with appropriate parameters.

Port Write:

Ports can be written to at byte/word level using the interface "[SYS_PORTS_Write](#)" appropriate parameters.

Port Clear:

Ports can be cleared at byte/word level using the interface "[SYS_PORTS_Clear](#)" with appropriate parameters.

Port Set:

Ports can be set at byte/word level using the interface "[SYS_PORTS_Set](#)" with appropriate parameters.

Port Direction Control:

Ports direction can be set at byte/word level using the interface "[SYS_PORTS_DirectionSelect](#)" with appropriate parameters.

The Direction information can be obtained through the interface "[SYS_PORTS_DirectionGet](#)".

Port Toggle:

Ports can be toggled at byte/word level using the interface "[SYS_PORTS_Toggle](#)" with appropriate parameters.

Port Open Drain:

Ports can be enabled open drain functionality at byte/word level using the interface "[SYS_PORTS_OpenDrainEnable](#)" with appropriate parameters. Similarly, the Ports can be disabled open drain functionality at byte/word level using the interface "[SYS_PORTS_OpenDrainDisable](#)" with appropriate parameters.

Port Pull up:

Ports can be enabled pull up functionality at byte/word level using the interface "[SYS_PORTS_PullUpEnable](#)" with appropriate parameters. Similarly, the Ports can be disabled pull up functionality at byte/word level using the interface "[SYS_PORTS_PullUpDisable](#)" with appropriate parameters.

Example:

```
// PORT Direction setting for output
SYS_PORTS_DirectionSelect(MY_PORTS_INSTANCE, SYS_PORTS_DIRECTION_OUTPUT, MY_CHANNEL,
(PORTS_DATA_MASK)0xFFFF);
// PORT Direction setting for input
SYS_PORTS_DirectionSelect(MY_PORTS_INSTANCE, SYS_PORTS_DIRECTION_INPUT, MY_CHANNEL,
(PORTS_DATA_MASK)0xFFFF);
// Writing a value into a PORT
SYS_PORTS_Write(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_TYPE)0x1234)
// Reading back the previously written value
PORTS_DATA_TYPE readData = SYS_PORTS_Read(MY_PORTS_INSTANCE, MY_CHANNEL);
// Clearing the PORT
SYS_PORTS_Clear(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
// Setting the port
SYS_PORTS_Set(MY_PORTS_INSTANCE, MY_CHANNEL, 0x1234, (PORTS_DATA_MASK)0x00FF);
// Toggling a PORT
SYS_PORTS_Toggle(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
```

Note: Not all the features are available on all devices. Please refer to the specific device data sheet to determine availability.

5.7.7.4.3.3 Change Notification

Change Notification Feature Usage

The change notification feature can be enabled using "[SYS_PORTS_ChangeNotificationEnable](#)". This routine does the following operations.

Change notification can be disabled after the successful usage using the interface "[SYS_PORTS_ChangeNotificationDisable](#)".

Certain microcontrollers support the global control over the change notification feature using the following interfaces "[SYS_PORTS_ChangeNotificationGlobalEnable](#)" and "[SYS_PORTS_ChangeNotificationGlobalDisable](#)".

The status of the change notification pin can be obtained through interface "[SYS_PORTS_ChangeNotificationStatus](#)".

If there are any requirements to control the pull downs the following interfaces could be used,

"[SYS_PORTS_ChangeNotificationPullDownEnable](#)" and "[SYS_PORTS_ChangeNotificationPullDownDisable](#)".

If there are any requirements to control the pull ups the following interfaces could be used,

"[SYS_PORTS_ChangeNotificationPullUpEnable](#)" and "[SYS_PORTS_ChangeNotificationPullUpDisable](#)".

Change Notification Operation in Sleep and Idle Modes

The change notification module continues to operate during Sleep or Idle mode. Its operation can be enabled /disabled using the interfaces "[SYS_PORTS_ChangeNotificationInIdleModeEnable](#)" / "[SYS_PORTS_ChangeNotificationInIdleModeDisable](#)".

If one of the enabled change notify pins changes states, the respective status bit will be set which can be monitored using the interface "[SYS_PORTS_ChangeNotificationStatus](#)".

Example:

```
// Enabling the global change notification
SYS_PORTS_ChangeNotificationGlobalEnable(MY_PORTS_INSTANCE);
// Enabling weak pull-ups for the change notification PIN 10
SYS_PORTS_ChangeNotificationPullUpEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10);
// Enabling change notification on PIN 10
SYS_PORTS_ChangeNotificationEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10,
SYS_PORTS_PULLUP_ENABLE);
```



```
// Enabling the change notification in idle mode.
SYS_PORTS_ChangeNotificationInIdleModeEnable(MY_PORTS_INSTANCE);
//Obtaining the status of the change notification pin.
bool status;
status = SYS_PORTS_ChangeNotificationStatus(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10);
```

Note: Not all the features are available on all devices. Please refer to the specific device data sheet to determine availability.

5.7.7.4.3.4 Peripheral Pin Select

Ports Remapping or Peripheral Pin Select Usage

Input/Output Function Remapping:

"SYS_PORTS_RemapInputOutput" with appropriate parameters can be used to remap a particular port pin as input/output for a peripheral.

Input/Output Lock:

"SYS_PORTS_IOLockEnable" and "SYS_PORTS_IOLockDisable" with appropriate parameters can be used to lock/unlock a pin.

Example:

```
// Remapping input function 'Input Capture 1' to the Remappable input pin 'RPI1'
SYS_PORTS_RemapInputOutput(MY_PORTS_INSTANCE, PORTS_REMAP_FUNCTION_IC1,
PORTS_REMAP_PIN_RPI1);
// Remapping output function 'UART1 Transmit' to the Remappable output pin 'RP3'
SYS_PORTS_RemapInputOutput(MY_PORTS_INSTANCE, PORTS_REMAP_FUNCTION_U1TX,
PORTS_REMAP_PIN_RP3);
```

Note: Not all the features are available on all devices. Please refer to the specific device data sheet to determine availability.

5.7.7.4.3.5 Miscellaneous

Other Usage

Slew Rate:

Slew rate of a particular port can be controlled through the interface "SYS_PORTS_SlewRateSetReduced" and "SYS_PORTS_SlewRateSetStandard".

Open Drain:

Peripheral based open drain can be controlled through the interfaces "SYS_PORTS_PeripheralOpenDrainEnable" and "SYS_PORTS_PeripheralOpenDrainDisable".

Note: Not all the features are available on all devices. Please refer to the specific device data sheet to determine availability.

5.7.7.4.3.6 Special Considerations

Note on Ports Usage:

1. Setting a port pin as an analog input also requires that the corresponding direction be set. If the direction is set to output, the digital output level (VOH or VOL) will be outputted.
2. When reading the port register, all pins configured as analog input channels will read as cleared.
3. Pins configured as digital inputs will not output an analog input. Analog levels on any pin that is defined as a digital input may cause the input buffer to consume current that exceeds the device specifications.
4. Pull-ups and pull-downs on change notification pins should always be disabled when the port pin is configured as a digital output.

Considerations for the Peripheral Pin Select:

The ability to control Peripheral Pin Select options introduces several considerations into application design that could be overlooked. This is particularly true for several common peripherals that are available only as remappable peripherals.

The main consideration is that the Peripheral Pin Selects are not available on default pins in the device's default (Reset) state, all Peripheral Pin Select inputs are tied to Vss and all Peripheral Pin Select outputs are disconnected. This situation requires the user to initialize the device with the proper peripheral configuration before any other application code is executed. For application safety, however, it is best to lock the configuration after writing to the control registers.

A final consideration is that Peripheral Pin Select functions neither override analog inputs, nor reconfigure pins with analog functions for digital I/O. If a pin is configured as an analog input on device Reset, it must be explicitly reconfigured as digital I/O when used with a Peripheral Pin Select.

Note: Not all the features are available on all devices. Please refer to the specific device data sheet to determine availability.

5.7.7.5 Configuring the Library

The configuration of the Ports System Service is based on the file `sys_config.h`

This header file contains the configuration selection for the PORTS System Service. Based on the selections made, the Ports System Service will support or not support selected features. These configuration settings will apply to all instances of the Ports System Service.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.7.7.6 Building the Library

5.7.7.7 Library Interface

Data Types and Constants

	Name	Description
	SYS_PORTS_PIN_DIRECTION	Defines the direction of the port pins.
	SYS_PORTS_PULLUP_PULLDOWN_STATUS	Provides the pull-up and pull-down status.

Change Notification Functions

	Name	Description
	SYS_PORTS_ChangeNotificationDisable	Disables the change notification for the selected port.
	SYS_PORTS_ChangeNotificationEnable	Enables the change notification for the selected port.
	SYS_PORTS_ChangeNotificationGlobalDisable	Globally disables the change notification for the selected port.
	SYS_PORTS_ChangeNotificationGlobalEnable	Globally enables the change notification for the selected port.
	SYS_PORTS_ChangeNotificationInIdleModeDisable	Disables the change notification for the selected port in Sleep or Idle mode.
	SYS_PORTS_ChangeNotificationInIdleModeEnable	Enables the change notification for the selected port in Sleep or Idle mode.
	SYS_PORTS_ChangeNotificationPullDownDisable	Disables pull-down on input change.
	SYS_PORTS_ChangeNotificationPullDownEnable	Enables pull-down on input change.
	SYS_PORTS_ChangeNotificationPullUpDisable	Disables pull-up on input change.

	SYS_PORTS_ChangeNotificationPullUpEnable	Enables a weak pull-up on the change notification pin.
	SYS_PORTS_ChangeNotificationStatus	Gets the status of the corresponding notification pin.

Peripheral Pin Select Functions

	Name	Description
	SYS_PORTS_IOLockDisable	Disables the I/O lock.
	SYS_PORTS_IOLockEnable	Enables the I/O lock.
	SYS_PORTS_IOLockIsActive	Returns the status of I/O lock.
	SYS_PORTS_RemapInput	Input/Output (I/O) function remapping.
	SYS_PORTS_RemapOutput	Input/Output (I/O) function remapping.

Pin Control Functions

	Name	Description
	SYS_PORTS_PinModeSelect	Enables the selected pin as analog or digital.
	SYS_PORTS_PinOpenDrainDisable	Disables the open-drain functionality for the selected pin.
	SYS_PORTS_PinOpenDrainEnable	Enables the open-drain functionality for the selected pin.
	SYS_PORTS_PinPullUpDisable	Disables the pull-up functionality for the selected pin
	SYS_PORTS_PinPullUpEnable	Enables the pull-up functionality for the selected pin.
	SYS_PORTS_PinRead	Reads the selected digital pin.
	SYS_PORTS_PinSet	Sets the selected digital pin/latch.
	SYS_PORTS_PinToggle	Toggles the selected digital pin.
	SYS_PORTS_PinWrite	Writes the selected digital pin.
	SYS_PORTS_PinClear	Clears the selected digital pin.
	SYS_PORTS_PinDirectionSelect	Enables the direction for the selected pin.

Ports Control Functions

	Name	Description
	SYS_PORTS_Clear	Clears the selected digital port.
	SYS_PORTS_DirectionGet	Reads the direction for the selected port.
	SYS_PORTS_DirectionSelect	Enables the direction for the selected port.
	SYS_PORTS_OpenDrainDisable	Disables the open-drain functionality for the selected port.
	SYS_PORTS_OpenDrainEnable	Enables the open-drain functionality for the selected port.
	SYS_PORTS_PullUpDisable	Disables the pull-up for the selected port.
	SYS_PORTS_PullUpEnable	Enables the pull-up for the selected port.
	SYS_PORTS_Read	Reads the data from the I/O port.
	SYS_PORTS_Set	Writes the selected digital port/latch based on the mask.
	SYS_PORTS_Toggle	Toggles the selected digital port pins.
	SYS_PORTS_Write	Writes the data to the I/O port.

Description

5.7.7.7.1 Pin Control Functions

5.7.7.7.1.1 SYS_PORTS_PinModeSelect Function

C

```
void SYS_PORTS_PinModeSelect(  
    PORTS_MODULE_ID index,  
    PORTS_ANALOG_PIN pin,  
    PORTS_PIN_MODE mode  
);
```

Description

This function enables the selected pin as analog or digital.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	Possible values of PORTS_ANALOG_PIN
mode	Possible values of PORTS_PIN_MODE

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PIN -  
PORTS_ANALOG_PIN_AN0 // MY_PIN_MODE - PORTS_PIN_MODE_ANALOG  
SYS_PORTS_PinModSYS_PORTS_PinModeSelecteSelect( MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE );
```

5.7.7.7.1.2 SYS_PORTS_PinOpenDrainDisable Function

C

```
void SYS_PORTS_PinOpenDrainDisable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function disables the open-drain functionality for the selected pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_PIN_10 SYS_PORTS_PinOpenDrainDisable(MY_PORTS_INSTANCE, MY_PINNUM);
```

5.7.7.7.1.3 SYS_PORTS_PinOpenDrainEnable Function

C

```
void SYS_PORTS_PinOpenDrainEnable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function enables the open-drain functionality for the selected pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_PIN_10 SYS_PORTS_PinOpenDrainEnable( MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM );
```

5.7.7.7.1.4 SYS_PORTS_PinPullUpDisable Function

C

```
void SYS_PORTS_PinPullUpDisable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function disables the pull-up functionality for the selected pin.

Preconditions

None

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability. On some devices, pull-up disable leads to pull-down enable.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_PIN_10 SYS_PORTS_PinPullUpDisable( MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM );
```

5.7.7.7.1.5 SYS_PORTS_PinPullUpEnable Function**C**

```
void SYS_PORTS_PinPullUpEnable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function enables the pull-up functionality for the selected pin.

Preconditions

None

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_PIN_10 SYS_PORTS_PinPullUpEnable( MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM );
```

5.7.7.7.1.6 SYS_PORTS_PinRead Function

C

```
bool SYS_PORTS_PinRead(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function reads the selected digital pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

The status of the port pin.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_PIN_10 bool bitStatus = SYS_PORTS_PinRead(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.7.7.7.1.7 SYS_PORTS_PinSet Function

C

```
void SYS_PORTS_PinSet(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_BIT_POS bitPos  
);
```

Description

This function sets the selected digital pin/latch.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
SYS_PORTS_PinSet(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

5.7.7.7.1.8 SYS_PORTS_PinToggle Function**C**

```
void SYS_PORTS_PinToggle(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function toggles the selected digital pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -
PORTS_PIN_10 SYS_PORTS_PinToggle( MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM );
```

5.7.7.7.1.9 SYS_PORTS_PinWrite Function**C**

```
void SYS_PORTS_PinWrite(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos,
    bool value
);
```

Description

This function writes the selected digital pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

5.7.7.7.1.10 SYS_PORTS_PinClear Function**C**

```
void SYS_PORTS_PinClear(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This function clears the selected digital pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -
PORTS_IO_PIN_10 SYS_PORTS_PinClear( MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM );
```

5.7.7.7.1.11 SYS_PORTS_PinDirectionSelect Function**C**

```
void SYS_PORTS_PinDirectionSelect(
    PORTS_MODULE_ID index,
    SYS_PORTS_PIN_DIRECTION pinDir,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

Description

This functino enables the direction for the selected pin.

Preconditions

None

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinDir	Pin direction
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_PIN_10 SYS\_PORTS\_PIN\_DIRECTION pinDir; pinDir = SYS_PORTS_DIRECTION_INPUT;  
SYS_PORTS_PinDirectionSelect(MY_PORTS_INSTANCE, pinDir, MY_CHANNEL, MY_PINNUM);
```

5.7.7.7.2 Ports Control Functions

5.7.7.7.2.1 SYS_PORTS_Clear Function

C

```
void SYS_PORTS_Clear(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_DATA_MASK clearMask  
);
```

Description

This function clears the selected digital port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
clearMask	Identifies the bits to be cleared

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. PORTS_DATA_MASK
clearMask = (PORTS_DATA_MASK)0x00FF; SYS_PORTS_Clear( MY_PORTS_INSTANCE, MY_CHANNEL, clearMask );
```

5.7.7.7.2.2 SYS_PORTS_DirectionGet Function**C**

```
PORTS_DATA_MASK SYS_PORTS_DirectionGet(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel
);
```

Description

This function reads the direction for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.

Returns

Direction of the port.

Remarks

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. PORTS_DATA_MASK
value; value = SYS_PORTS_DirectionGet( MY_PORTS_INSTANCE, MY_CHANNEL );
```

5.7.7.7.2.3 SYS_PORTS_DirectionSelect Function**C**

```
void SYS_PORTS_DirectionSelect(
    PORTS_MODULE_ID index,
    SYS_PORTS_PIN_DIRECTION pinDir,
    PORTS_CHANNEL channel,
    PORTS_DATA_MASK mask
);
```

Description

This function enables the direction for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinDir	Pin direction

channel	Identifier for the PORT channel: A, B, C, etc.
mask	Mask for the direction of width PORTS_DATA_MASK

Returns

None

Remarks

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.
SYS_PORTS_PIN_DIRECTION pinDir; pinDir = SYS_PORTS_DIRECTION_INPUT; PORTS\_DATA\_MASK myMask =
(PORTS\_DATA\_MASK)0x00FF; SYS_PORTS_DirectionSelect(MY_PORTS_INSTANCE, pinDir, MY_CHANNEL, myMask );
```

5.7.7.7.2.4 SYS_PORTS_OpenDrainDisable Function**C**

```
void SYS_PORTS_OpenDrainDisable(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_DATA_MASK mask
);
```

Description

This function disables the open-drain functionality for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
mask	Mask of type PORTS_DATA_MASK

Returns

None

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.
SYS_PORTS_PeripheralOpenDrainDisable( MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS\_DATA\_MASK)0x00FF );
```

5.7.7.7.2.5 SYS_PORTS_OpenDrainEnable Function**C**

```
void SYS_PORTS_OpenDrainEnable(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_DATA_MASK mask
);
```

Description

This function enables the open-drain functionality for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
mask	Mask of type PORTS_DATA_MASK

Returns

None

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.  
SYS_PORTS_OpenDrainEnable( MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS\_DATA\_MASK)0x00FF );
```

5.7.7.7.2.6 SYS_PORTS_PullUpDisable Function**C**

```
void SYS_PORTS_PullUpDisable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_DATA_MASK mask  
);
```

Description

This function disables the pull-up for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
mask	Mask of type PORTS_DATA_MASK

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability. On some devices, pull-up disable leads to pull-down enable.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.
```

```
SYS_PORTS_PullUpDisable( MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF );
```

5.7.7.7.2.7 SYS_PORTS_PullUpEnable Function

C

```
void SYS_PORTS_PullUpEnable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_DATA_MASK mask  
);
```

Description

This function enables the pull-up for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
mask	Mask of type PORTS_DATA_MASK

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.  
SYS_PORTS_PullUpEnable( MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF );
```

5.7.7.7.2.8 SYS_PORTS_Read Function

C

```
PORTS_DATA_TYPE SYS_PORTS_Read(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel  
);
```

Description

This function reads the data from the I/O port.

Preconditions

The direction of the port to be set as input.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.

Returns

Return the data read from the port.

Remarks

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. PORTS\_DATA\_TYPE  
readData; readData = SYS_PORTS_Read( MY_PORTS_INSTANCE, MY_CHANNEL );
```

5.7.7.7.2.9 SYS_PORTS_Set Function**C**

```
void SYS_PORTS_Set(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_DATA_TYPE value,  
    PORTS_DATA_MASK mask  
);
```

Description

This function writes to the selected digital port/latch relative to the mask.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
value	Value to be written into a port of width PORTS_DATA_TYPE
mask	Identifies the bits to be written

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// MY_VALUE - 0x1234  
PORTS_DATA_MASK myMask = (PORTS_DATA_MASK)0x00FF;  
SYS_PORTS_Set(MY_PORTS_INSTANCE, MY_CHANNEL, MY_VALUE, myMask);
```

5.7.7.7.2.10 SYS_PORTS_Toggle Function**C**

```
void SYS_PORTS_Toggle(  
    PORTS_MODULE_ID index,  
    PORTS_CHANNEL channel,  
    PORTS_DATA_MASK toggleMask  
);
```

Description

This function toggles the selected digital port pins.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
toggleMask	Identifies the bits to be toggled

Returns

None

Remarks

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. PORTS_DATA_MASK
toggleMask = (PORTS_DATA_MASK)0x00FF; SYS_PORTS_Toggle( MY_PORTS_INSTANCE, MY_CHANNEL, toggleMask );
```

5.7.7.7.2.11 SYS_PORTS_Write Function**C**

```
void SYS_PORTS_Write(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_DATA_TYPE value
);
```

Description

This function writes the data to the I/O port.

Preconditions

The direction of the port to be set as output.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
value	Value to be written into a port of width PORTS_DATA_TYPE

Returns

None.

Remarks

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. PORTS_DATA_TYPE
writeData; SYS_PORTS_Write( MY_PORTS_INSTANCE, MY_CHANNEL, MY_VALUE );
```

5.7.7.7.3 Change Notification Functions

5.7.7.7.3.1 SYS_PORTS_ChangeNotificationDisable Function

C

```
void SYS_PORTS_ChangeNotificationDisable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANGE_NOTICE_PIN pinNum  
);
```

Description

This function disables the change notification for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.  
PORTS\_CHANGE\_NOTICE\_PIN pinNum; SYS_PORTS_ChangeNotificationDisable( index, pinNum );
```

5.7.7.7.3.2 SYS_PORTS_ChangeNotificationEnable Function

C

```
void SYS_PORTS_ChangeNotificationEnable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANGE_NOTICE_PIN pinNum,  
    SYS_PORTS_PULLUP_PULLDOWN_STATUS value  
);
```

Description

This function enables the change notification for the selected port.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Pull-up enable or disable value
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.  
SYS_PORTS_PULLUP_PULLDOWN_STATUS value; PORTS_CHANGE_NOTICE_PIN pinNum;  
SYS_PORTS_ChangeNotificationEnable( index, pinNum, value );
```

5.7.7.7.3.3 SYS_PORTS_ChangeNotificationGlobalDisable Function**C**

```
void SYS_PORTS_ChangeNotificationGlobalDisable(  
    PORTS_MODULE_ID index  
);
```

Description

This function globally disables the change notification for the selected port.

Preconditions

None.

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.  
SYS_PORTS_ChangeNotificationGlobalDisable( MY_PORTS_INSTANCE);
```

5.7.7.7.3.4 SYS_PORTS_ChangeNotificationGlobalEnable Function**C**

```
void SYS_PORTS_ChangeNotificationGlobalEnable(  
    PORTS_MODULE_ID index  
);
```

Description

This function globally enables the change notification for the selected port.

Preconditions

None.

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.  
SYS_PORTS_ChangeNotificationGlobalEnable( MY_PORTS_INSTANCE );
```

5.7.7.7.3.5 SYS_PORTS_ChangeNotificationInIdleModeDisable Function

C

```
void SYS_PORTS_ChangeNotificationInIdleModeDisable(  
    PORTS_MODULE_ID index  
);
```

Description

This function disables the change notification for the selected port in Sleep or Idle mode.

Preconditions

None.

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.  
SYS_PORTS_ChangeNotificationInIdleModeDisable( MY_PORTS_INSTANCE );
```

5.7.7.7.3.6 SYS_PORTS_ChangeNotificationInIdleModeEnable Function

C

```
void SYS_PORTS_ChangeNotificationInIdleModeEnable(  
    PORTS_MODULE_ID index  
);
```

Description

This function enables the change notification for the selected port in Sleep or Idle mode.

Preconditions

None.

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer.  
SYS_PORTS_ChangeNotificationInIdleModeEnable( MY_PORTS_INSTANCE );
```

5.7.7.7.3.7 SYS_PORTS_ChangeNotificationPullDownDisable Function

C

```
void SYS_PORTS_ChangeNotificationPullDownDisable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANGE_NOTICE_PIN pinNum  
);
```

Description

This function disables pull-down on input change.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_CHANGE_NOTICE_PIN_10      SYS_PORTS_ChangeNotificationPullDownDisable(      MY_PORTS_INSTANCE,  
MY_PINNUM );
```

5.7.7.7.3.8 SYS_PORTS_ChangeNotificationPullDownEnable Function**C**

```
void SYS_PORTS_ChangeNotificationPullDownEnable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANGE_NOTICE_PIN pinNum  
);
```

Description

This function enables pull-down on input change.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_CHANGE_NOTICE_PIN_10      SYS_PORTS_ChangeNotificationPullDownEnable(      MY_PORTS_INSTANCE,  
MY_PINNUM );
```

5.7.7.7.3.9 SYS_PORTS_ChangeNotificationPullUpDisable Function

C

```
void SYS_PORTS_ChangeNotificationPullUpDisable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANGE_NOTICE_PIN pinNum  
);
```

Description

This function disables pull-up on input change.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_CHANGE_NOTICE_PIN_10 SYS_PORTS_ChangeNotificationPullUpDisable( MY_PORTS_INSTANCE, MY_PINNUM );
```

5.7.7.7.3.10 SYS_PORTS_ChangeNotificationPullUpEnable Function

C

```
void SYS_PORTS_ChangeNotificationPullUpEnable(  
    PORTS_MODULE_ID index,  
    PORTS_CHANGE_NOTICE_PIN pinNum  
);
```

Description

This function enables a weak pull-up on the change notification pin.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

None.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. // MY_PINNUM -  
PORTS_CHANGE_NOTICE_PIN_10 SYS_PORTS_ChangeNotificationPullUpEnable( MY_PORTS_INSTANCE, MY_PINNUM );
```

5.7.7.7.3.11 SYS_PORTS_ChangeNotificationStatus Function**C**

```
bool SYS_PORTS_ChangeNotificationStatus(  
    PORTS_MODULE_ID index,  
    PORTS_CHANGE_NOTICE_PIN pinNum  
);
```

Description

This function gets the status of the corresponding notification pin..

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Returns

- true - Change notice has occurred
- false - Change notice has not occurred

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the // application developer. //pinNUM is the change  
notification pin bool status; status = SYS_PORTS_ChangeNotificationStatus( MY_PORTS_INSTANCE, MY_PINNUM );
```

5.7.7.7.4 Peripheral Pin Select Functions**5.7.7.7.4.1 SYS_PORTS_IOLockDisable Function****C**

```
void SYS_PORTS_IOLockDisable(  
    PORTS_MODULE_ID index  
);
```

Description

This function disables the I/O lock.

Preconditions

index - Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is available for devices with PPS functionality. If the associated Configuration bit is set and if the I/O lock is enabled, this feature can never be used to disable the I/O lock. The only way to clear the bit and re-enable peripheral remapping is to perform a device Reset.

This function is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
SYS_PORTS_IOLockDisable(MY_PORTS_INSTANCE);
```

5.7.7.7.4.2 SYS_PORTS_IOLockEnable Function**C**

```
void SYS_PORTS_IOLockEnable(  
    PORTS_MODULE_ID index  
);
```

Description

This function enables the I/O lock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature is available for devices with Peripheral Pin Select (PPS) functionality. If the associated Configuration bit is set, this function can only be set once.

This function is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
SYS_PORTS_IOLockEnable(MY_PORTS_INSTANCE);
```

5.7.7.7.4.3 SYS_PORTS_IOLockIsActive Function**C**

```
bool SYS_PORTS_IOLockIsActive(  
    PORTS_MODULE_ID index  
);
```

Description

This function returns the status of I/O lock.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

- true - I/O lock is active
- false - I/O lock is not active

Remarks

This feature is available for devices with PPS functionality.

This function is not available on all devices. Please refer to the specific device data sheet for availability.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
bool lockStatus;  
lockStatus = SYS_PORTS_IOLockIsActive(MY_PORTS_INSTANCE);
```

5.7.7.7.4.4 SYS_PORTS_RemapInput Function**C**

```
void SYS_PORTS_RemapInput(  
    PORTS_MODULE_ID index,  
    PORTS_REMAP_INPUT_FUNCTION function,  
    PORTS_REMAP_INPUT_PIN remapPin  
);
```

Description

This function controls the I/O function remapping.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use SYS_PORTS_ExistsRemapInputOutput in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// Remapping input function 'Input Capture 1' to the Remappable pin 'RPD2'  
SYS_PORTS_RemapInput(MY_PORTS_INSTANCE, INPUT_FUNC_IC1, INPUT_PIN_RPD2 );
```

5.7.7.7.4.5 SYS_PORTS_RemapOutput Function**C**

```
void SYS_PORTS_RemapOutput(  
    PORTS_MODULE_ID index,
```



```
    PORTS_REMAP_OUTPUT_FUNCTION function,  
    PORTS_REMAP_OUTPUT_PIN remapPin  
);
```

Description

This function controls the I/O function remapping.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Returns

None.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use SYS_PORTS_ExistsRemapInputOutput in your application to determine whether this feature is available.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the  
// application developer.  
// Remapping output function 'UART3 Transmit' to the Remappable pin 'RPA14'  
SYS_PORTS_RemapInputOutput(MY_PORTS_INSTANCE, OUTPUT_FUNC_U3TX, OUTPUT_PIN_RPA14);
```

5.7.7.7.5 Data Types and Constants

5.7.7.7.5.1 SYS_PORTS_PIN_DIRECTION Enumeration

C

```
typedef enum {  
    SYS_PORTS_DIRECTION_OUTPUT,  
    SYS_PORTS_DIRECTION_INPUT  
} SYS_PORTS_PIN_DIRECTION;
```

Description

SYS_PORTS_PIN_DIRECTION

These constants provide the port pin direction definitions.

Members

Members	Description
SYS_PORTS_DIRECTION_OUTPUT	Direction as output
SYS_PORTS_DIRECTION_INPUT	Direction as input

Remarks

None.

5.7.7.7.5.2 SYS_PORTS_PULLUP_PULLDOWN_STATUS Enumeration

C

```
typedef enum {  
    SYS_PORTS_PULLUP_DISABLE,
```

```
SYS_PORTS_PULLUP_ENABLE
} SYS_PORTS_PULLUP_PULLDOWN_STATUS;
```

Description

SYS PORTS PULLUP status

These constants provide the pull-up or pull-down status definitions.

Members

Members	Description
SYS_PORTS_PULLUP_DISABLE	PULLUP Disable
SYS_PORTS_PULLUP_ENABLE	PULLUP Enable

Remarks

None.

5.7.7.8 Files

Files

Name	Description
help_sys_ports.h	This is file help_sys_ports.h.
sys_ports.h	Ports System Service interface definitions

Description

5.7.7.8.1 help_sys_ports.h

This is file help_sys_ports.h.

Enumerations

	Name	Description
	PORTS_ANALOG_PIN	Data type defining the different Analog input pins
	PORTS_CHANGE_NOTICE_PIN	Data type defining the different Change Notification Pins enumeration
	PORTS_CHANNEL	Identifies the PORT Channels Supported.
	PORTS_MODULE_ID	Identifies the PORT Modules Supported.
	PORTS_PERIPHERAL_OD	Data type defining the different Peripherals available for Open drain Configuration
	PORTS_PIN	Data type defining the different PORTS IO Pins enumeration
	PORTS_REMAP_FUNCTION	Data type defining the different remap function enumeration
	PORTS_REMAP_PIN	Data type defining the different remappable input/output enumeration

5.7.7.8.2 sys_ports.h

Ports System Service Interface Definition

This file contains the interface definition for the Ports system service. It provides a way to interact with the Ports subsystem to manage the timing requests supported by the system

Enumerations

	Name	Description
	SYS_PORTS_PIN_DIRECTION	Defines the direction of the port pins.
	SYS_PORTS_PULLUP_PULLDOWN_STATUS	Provides the pull-up and pull-down status.

Functions

	Name	Description
	SYS_PORTS_ChangeNotificationDisable	Disables the change notification for the selected port.
	SYS_PORTS_ChangeNotificationEnable	Enables the change notification for the selected port.
	SYS_PORTS_ChangeNotificationGlobalDisable	Globally disables the change notification for the selected port.
	SYS_PORTS_ChangeNotificationGlobalEnable	Globally enables the change notification for the selected port.
	SYS_PORTS_ChangeNotificationInIdleModeDisable	Disables the change notification for the selected port in Sleep or Idle mode.
	SYS_PORTS_ChangeNotificationInIdleModeEnable	Enables the change notification for the selected port in Sleep or Idle mode.
	SYS_PORTS_ChangeNotificationPullDownDisable	Disables pull-down on input change.
	SYS_PORTS_ChangeNotificationPullDownEnable	Enables pull-down on input change.
	SYS_PORTS_ChangeNotificationPullUpDisable	Disables pull-up on input change.
	SYS_PORTS_ChangeNotificationPullUpEnable	Enables a weak pull-up on the change notification pin.
	SYS_PORTS_ChangeNotificationStatus	Gets the status of the corresponding notification pin.
	SYS_PORTS_Clear	Clears the selected digital port.
	SYS_PORTS_DirectionGet	Reads the direction for the selected port.
	SYS_PORTS_DirectionSelect	Enables the direction for the selected port.
	SYS_PORTS_IOLockDisable	Disables the I/O lock.
	SYS_PORTS_IOLockEnable	Enables the I/O lock.
	SYS_PORTS_IOLockIsActive	Returns the status of I/O lock.
	SYS_PORTS_OpenDrainDisable	Disables the open-drain functionality for the selected port.
	SYS_PORTS_OpenDrainEnable	Enables the open-drain functionality for the selected port.
	SYS_PORTS_PinClear	Clears the selected digital pin.
	SYS_PORTS_PinDirectionSelect	Enables the direction for the selected pin.
	SYS_PORTS_PinModeSelect	Enables the selected pin as analog or digital.
	SYS_PORTS_PinOpenDrainDisable	Disables the open-drain functionality for the selected pin.
	SYS_PORTS_PinOpenDrainEnable	Enables the open-drain functionality for the selected pin.
	SYS_PORTS_PinPullUpDisable	Disables the pull-up functionality for the selected pin
	SYS_PORTS_PinPullUpEnable	Enables the pull-up functionality for the selected pin.
	SYS_PORTS_PinRead	Reads the selected digital pin.
	SYS_PORTS_PinSet	Sets the selected digital pin/latch.
	SYS_PORTS_PinToggle	Toggles the selected digital pin.
	SYS_PORTS_PinWrite	Writes the selected digital pin.
	SYS_PORTS_PullUpDisable	Disables the pull-up for the selected port.
	SYS_PORTS_PullUpEnable	Enables the pull-up for the selected port.
	SYS_PORTS_Read	Reads the data from the I/O port.
	SYS_PORTS_RemapInput	Input/Output (I/O) function remapping.
	SYS_PORTS_RemapOutput	Input/Output (I/O) function remapping.
	SYS_PORTS_Set	Writes the selected digital port/latch based on the mask.
	SYS_PORTS_Toggle	Toggles the selected digital port pins.

	SYS_PORTS_Write	Writes the data to the I/O port.

File Name

sys_ports.h

Company

Microchip Technology Inc.

5.7.8 Timer System Service Library

5.7.8.1 Introduction

Timer System Service Library
for
Microchip Microcontrollers



This library provides interfaces to manage alarms and/or delays.

Overview

The Timer System Service Library is capable of providing periodic or one shot alarms, delays to the user. It works as a client for the Timer driver and opens one of the instances of the driver to perform the activities. The periodicity or the delay is an integer multiple of the Timer driver alarm period.

5.7.8.2 Release Notes

MPLAB Harmony Version: v0.70b **Timer System Service Version :** 0.01 Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.7.8.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED AS IS MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.7.8.4 Using the Library

This topic describes the basic architecture of the Timer System Service Library and provides information and examples on how to use it.

Interface Header File: [sys_tmr.h](#)

The interface to the Timer System Service library is defined in the "[sys_tmr.h](#)" header file, which is included by the "sys.h" system service header file. Any C language source (.c) file that uses the Timer System Service library should include "sys.h".

Library File:

The Timer System Service Library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the System Service interacts with the framework.

Useful Terminologies:

Term	Description
Tick	Heart beats provided by the system timer. Defines the rate at which the callback gets called in a periodicity. One tick corresponds to one alarm period.
Alarm	Minimum resolution to which the Timer driver is configured, a tick or a callback is a multiple of this time.
One Shot callback	Callback gets called only once after the time is elapsed.
Periodic callback	Callback gets called at a periodic rate as defined by the user.
Delay	Non-blocking delays

Pre-requisites: Following modules are necessary for the Timer System Service layer to function,

- Timer Peripheral library
- Timer Device driver
- Interrupt Peripheral library [If Interrupt mode is opted]

- Interrupt System Service [If Interrupt mode is opted]

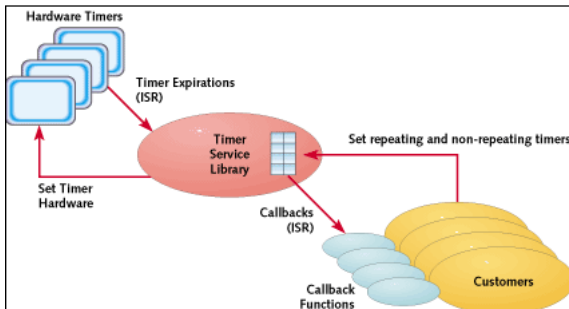
5.7.8.4.1 Abstraction Model

The Timer System Service module uses the Timer driver abstraction layer to provide the following functionalities,

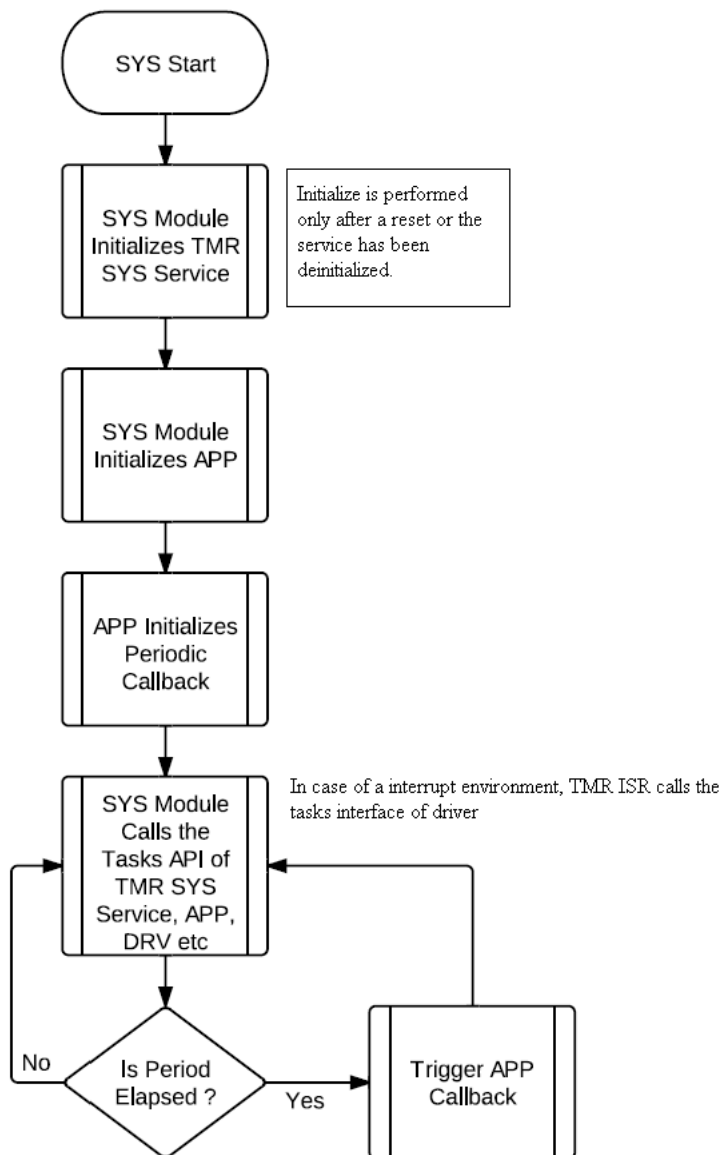
- Periodic Callback
- One Shot/Single Callback
- Delays [interrupt mode alone]

Note: Periodic/One shot modes of the Timer System Service will work in both polling and interrupt driven environments.

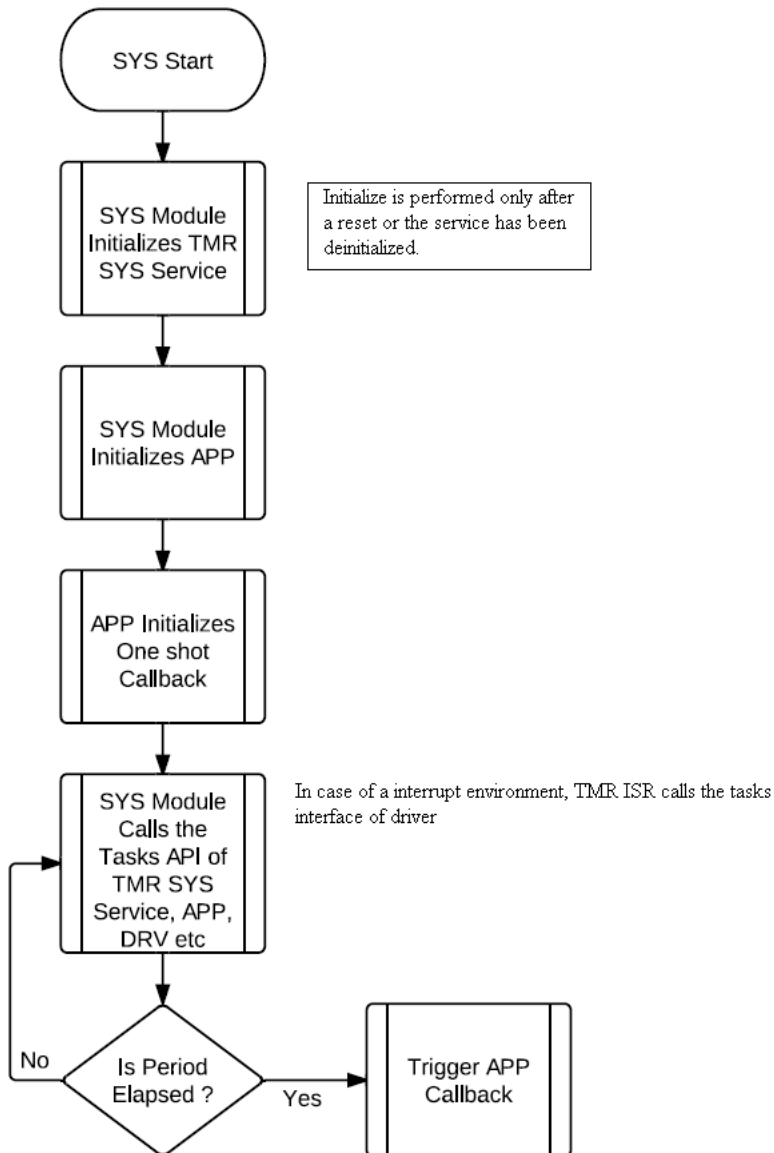
Simple Timer System Service abstraction model.



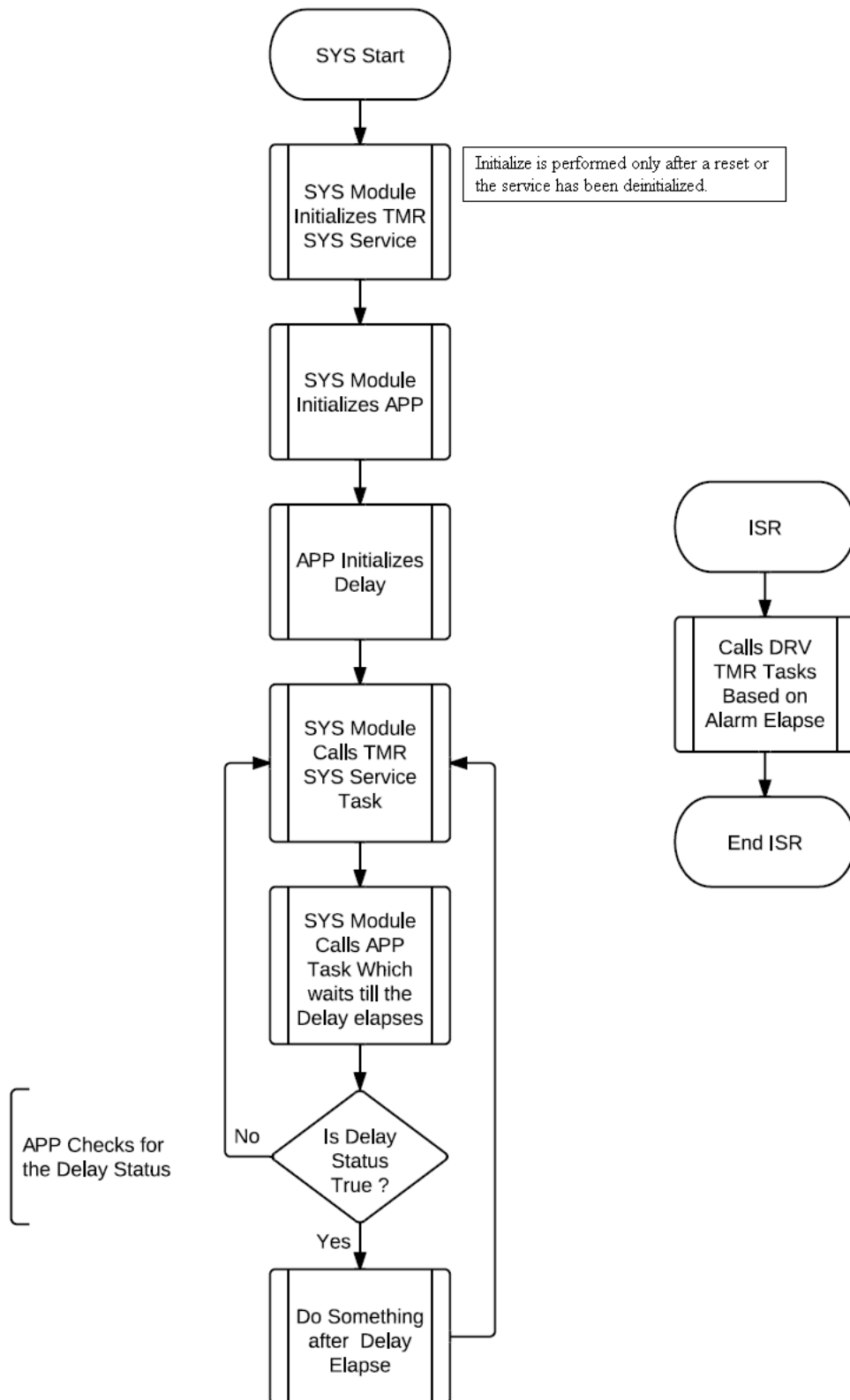
The following flow diagram depicts Periodic Callback usage model.



The following flow diagram depicts One Shot Callback usage model.



The following flow diagram depicts Delay usage model.



5.7.8.4.2 Library Overview

The library interface routines are divided into various subsections, each of the subsections addresses one of the blocks or the overall operation of the Timer System Service Library.

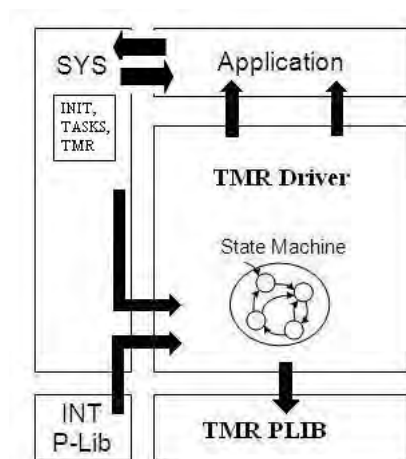
Library Interface Section	Description
System Level Interaction Functions	Provides system module APIs. Device initialization, deinitialization, reinitialization and status functions.
Timed Callback Functions	Provides interfaces to handle timed periodic or one shot callbacks
Timed Delay Functions	Provides interfaces to handle timed delays
Miscellaneous Functions	Provides interfaces for timer tick counts, etc.

5.7.8.4.3 How the Library Works

The library provides interfaces to support:

- System Interaction
- Periodic Callback
- One shot/Single Callback
- Delays [In Interrupt mode Alone]
- Tick count Information

The following model gives information on interaction between various modules.



Note: The arrows in the above diagram indicates the dependencies or support.

Module interaction occurs in the following order:

1. The initialization, reinitialization, deinitialization are handled by the SYS INIT module.
2. The Timer System Service Initialization function initializes the library's internal data structures.
3. After the initialization, Timer System Service Tasks function would be called by the SYS Tasks module to open the Timer Driver and do other configuration activities. Once the Timer driver is ready to be used the Timer System Service Tasks API makes the system ready to be used by setting the status of the Timer System Service module to `SYS_MODULE_RUNNING`.
4. The application can now enable either periodic callback or one-shot callback or enable delays.

Notes:

1. It is possible to have multiple clients access the system timer service.
2. Multiple client system calls for the configuration of multiple periodic callbacks or one shot and single periodic callback event etc.
3. It is not possible to use one client in polling and the other in the interrupt driven mode.
4. User or the application has to make sure to initialize the interrupt system module and set priorities accordingly.

5.7.8.4.3.1 System Interaction**Initialization and Reinitialization:**

The SYS module performs the initialization and the reinitialization of the Timer System Service. During initialization the following informations are populated into the internal data structures from the init structure '[SYS_TMR_INIT](#)' passed as a parameter:

Initialization Member	Description
moduleInit	System module initialization of the power state
drvIndex	Timer driver module/instance index, the same index used to initialize the timer driver
alarmPeriod	Alarm period in ms to configure the timer driver. This period will be used as the base time for further calculations of periodic or single callbacks etc.

The [SYS_TMR_Initialize](#) API returns a dummy handle of the type [SYS_MODULE_OBJ](#). There on, the object handle returned by the Initialize interface would be used by the other system interfaces like [SYS_TMR_Reinitialize](#), [SYS_TMR_Deinitialize](#), [SYS_TMR_Status](#) and [SYS_TMR_Tasks](#).

Note: Already initialized SYS TMR module if tried to initialize again, will safely return without modifying the behavior of the system.

Example for Timer System Service initialization:

```

SYS_MODULE_OBJ  objectHandle;
SYS_TMR_INIT    initConfig;

// Populate the SYS TMR initialization structure
initConfig.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
initConfig.drvIndex              = DRV_TMR_INDEX_0;
initConfig.alarmPeriod           = 100;

objectHandle = SYS_TMR_Initialize( SYS_TMR_INDEX_0, ( SYS_MODULE_INIT * )&initConfig );
if( SYS_MODULE_OBJ_INVALID == objectHandle )
{
    // Handle error
}

```

Sample initialization sequence in polling environment:

```

void SYS_Initialize( SYS_INIT_DATA *data )
{
    /* Application Initialization */
    App_Init();

    /* Initialize the Timer driver */
    drvTmrObject = DRV_TMR_Initialize( DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&tmrInitData );

    /* Initialize the SYS TMR Module */
    sysTmrObject = SYS_TMR_Initialize( DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&sysTmrInitData );
}

```

Sample initialization sequence in an interrupt driven environment:

```

void SYS_Initialize( SYS_INIT_DATA *data )
{
    /* Application Initialization */
    App_Init();

    /* Initialize the Timer driver */
    drvTmrObject = DRV_TMR_Initialize( DRV_TMR_INDEX_0, ( SYS_MODULE_INIT * )&tmrInitData );

    /* Initialize the SYS TMR Module */
    sysTmrObject = SYS_TMR_Initialize( DRV_TMR_INDEX_0, ( SYS_MODULE_INIT * )&sysTmrInitData );

    /* Set the Timer Interrupt priority, sub-priority if supported by the device */
    SYS_INT_PrioritySet( INT_SOURCE_TIMER_1, INT_PRIORITY_LEVEL1 );
    SYS_INT_SubprioritySet( INT_SOURCE_TIMER_1, INT_SUBPRIORITY_LEVEL1 );

    /* Initialize the interrupt sub system module */
    SYS_INT_Initialize();
}

```

Deinitialization:

The deinitialize operation ([SYS_TMR_Deinitialize](#)) places the module in inactive state. Once the initialize operation has been called, the deinitialize operation must be called before the initialize is called again. If the operation requires time to allow the hardware to complete, this will be reported by the [SYS_TMR_Status](#) operation.

Status:

Status of the system timer module can be checked using [SYS_TMR_Status](#) interface. After the initialization, reinitialization or the deinitialization activities the System timer Service status can be retrieved and checked for further actions.

Tasks Routine:

The system will call [SYS_TMR_Tasks](#) from System Task Service no matter whether we are in a polled or an interrupt driven environment. This interface gets the system timer module into running mode. This interface needs to be called after the timer driver is successfully initialized.

Sample call order in case of a polling environment:

```

void SYS_Tasks( void )
{
    /* Call Driver TMR Tasks API */
    DRV_TMR_Tasks( drvTmrObject );

    /* Call SYS TMR Tasks API */
    SYS_TMR_Tasks( sysTmrObject );

    /* Call the Application Tasks */
    App_Tasks();
}

```

Sample call order in case of an interrupt driven environment:

```

void SYS_Tasks( void )
{
    /* Call SYS TMR Tasks */
    SYS_TMR_Tasks( sysTmrObject );

    /* Call the Application Tasks */
    App_Tasks();
}

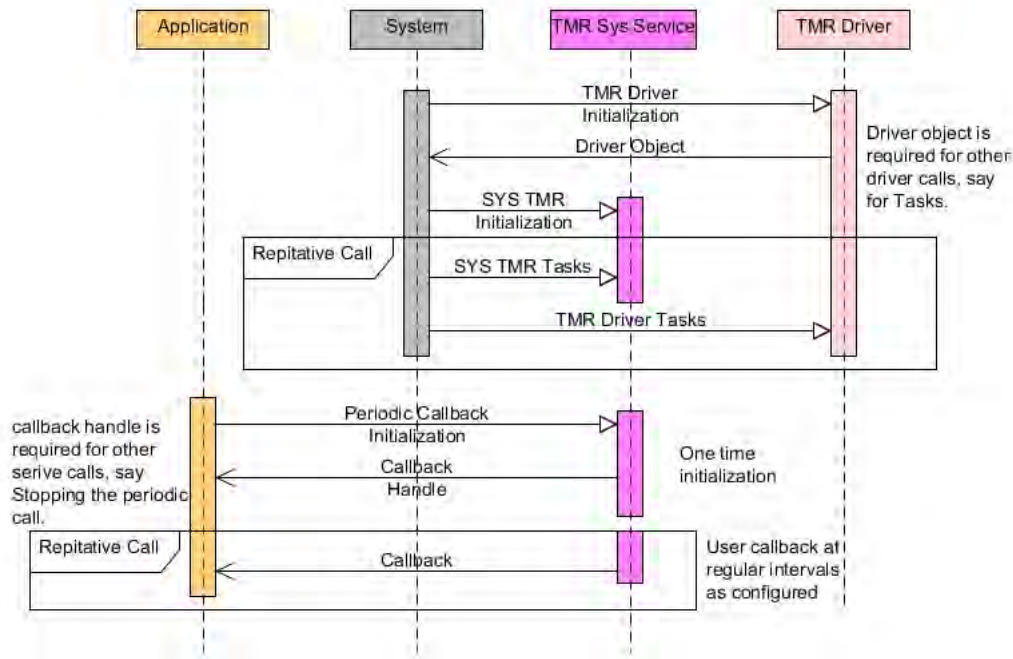
void InterruptServiceRoutine( void )
{
    /* Call the timer driver's "Tasks" routine */
    DRV_TMR_Tasks( drvTmrObject );
}

```

5.7.8.4.3.2 Periodic Callback

This section deals with the periodic callback registration and its usage.

The following diagram depicts the interaction across various modules for the periodic callback feature,



Steps involved in registering & using the feature:

1. Set the Timer System Service in the running mode (SYS_MODULE_RUNNING). See description to set to running mode. Use the SYS_TMR_STATUS function to verify that the service is in this mode.
2. Register the periodic callback feature using the [SYS_TMR_CallbackPeriodic](#) with the period and the pointer to the callback routine. The period parameter should be integral multiple of the Timer driver alarm period. The interface 'SYS_TMR_CallbackPeriodic' returns a valid handle after successful registration.

Note: The handle can be later used by the user to stop the periodic callback temporarily by the user using the interface 'SYS_TMR_CallbackStop'. Based on the periodicity set by the user the respective callback would be triggered repetitively after it elapses.

Example use case of this feature is described below,

```

void SYS_Initialize ( SYS_INIT_DATA *data )
{
    /* Initialize the Timer driver */
    drvTmrObject = DRV_TMR_Initialize ( SYS_INDEX, (SYS_MODULE_INIT *)&tmrInitData );

    /* Initialize the SYS TMR Module */
    sysTmrObject = SYS_TMR_Initialize ( SYS_INDEX, (SYS_MODULE_INIT *)&sysTmrInitData );

    /* Application Initialization */
    App_Initialize ();
}

void SYS_Tasks(void)

```

```

{
    /* Call Driver TMR Tasks API */
    DRV_TMR_Tasks ( drvTmrObject );

    /* Call SYS TMR Tasks API */
    SYS_TMR_Tasks ( sysTmrObject );

    /* Call the App Tasks */
    App_Tasks ();
}

void App_Tasks ( void )
{
    switch (testState)
    {
        /* Wait in Init state till the SYS TMR Module is in running mode */
        case TEST_STATE_Init:
            if (SYS_MODULE_RUNNING == SYS_TMR_Status(sysTmrObject))
            {
                /* SYS TMR is in running mode */
                testState = TEST_STATE_Config_Periodic;
            }
            else
            {
                testState = TEST_STATE_Init;
            }
            break;

        case TEST_STATE_Config_Periodic:
            /* Activate periodic callback */
            handle = SYS_TMR_CallbackPeriodic (120, &Test_Callback);
            testState = TEST_STATE_Count_Read;
            break;

        case TEST_STATE_Count_Read:
            testCount1 = SYS_TMR_TickCountGet ();
            break;
    }
}

// Callback which will be called repetitively after the configured
// time elapses
void Test_Callback ( void )
{
    PORTA = ~PORTA;
}

```

Note: The configuration '[SYS_TMR_INTERRUPT_MODE](#)' has no impact on this feature.

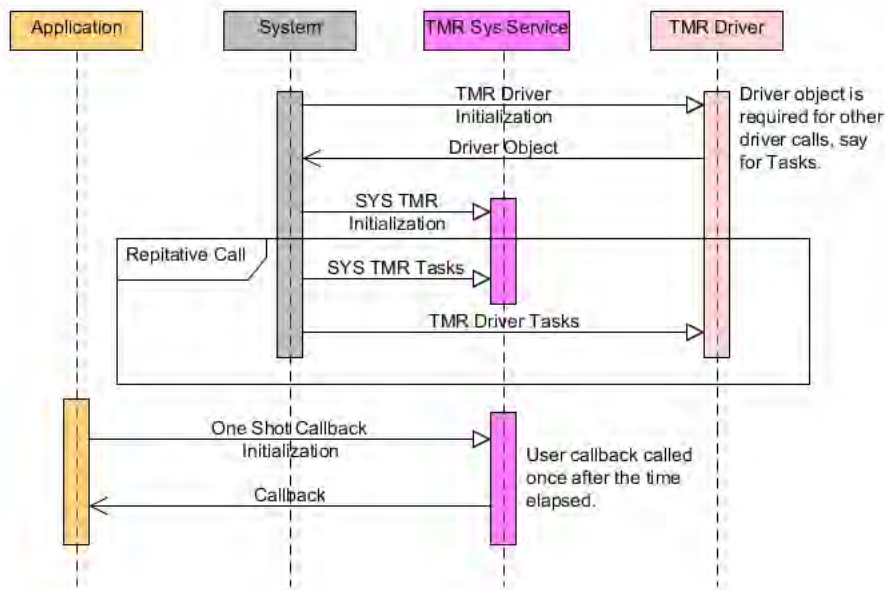
Single Periodic Event Feature: The configuration macro/switch '[SYS_TMR_SINGLE_PERIODIC_EVENT](#)', if set true will enables single periodic event feature. After the user creates one periodic callback event, no other periodic callback events are allowed. In the sense, multiple periodic events cannot be registered.

Multiple Periodic Event Feature: The configuration macro/switch '[SYS_TMR_SINGLE_PERIODIC_EVENT](#)', if set false will enable multiple periodic event feature.

5.7.8.4.3.3 One Shot Callback

This section deals with the one-shot/single callback registration and its usage.

The following diagram depicts the interaction across various modules for the one-shot callback feature,



Steps involved in registering and using the feature:

1. Initial requirement is to have the Timer System Service module in the mode 'SYS_MODULE_RUNNING'. The user can call the function [SYS_TMR_Status](#) to get the current status information of the module.
2. One-shot/Single callback feature can be registered by calling the function '[SYS_TMR_CallbackSingle](#)' with the parameters periodicity of the callback & a pointer to the user's callback routine. The period parameter has to be integral multiple of the TMR driver alarm period.

Based on the periodicity set by the user the respective callback would be triggered once after it elapses.

Note: The handle can be later used by the user to stop the one shot callback temporarily by the user using the interface '[SYS_TMR_CallbackStop](#)'.

Example use case of this feature is described below,

```

void SYS_Initialize ( SYS_INIT_DATA *data )
{
    /* Initialize the Timer driver */
    drvTmrObject = DRV_TMR_Initialize ( SYS_INDEX, (SYS_MODULE_INIT *)&tmrInitData );

    /* Initialize the SYS TMR Module */
    sysTmrObject = SYS_TMR_Initialize ( SYS_INDEX, (SYS_MODULE_INIT *)&sysTmrInitData );

    /* Application Initialization */
    App_Initialize ();
}

void SYS_Tasks(void)
{
    /* Call Driver TMR Tasks API */
    DRV_TMR_Tasks ( drvTmrObject );

    /* Call SYS TMR Tasks API */
    SYS_TMR_Tasks ( sysTmrObject );

    /* Call the App Tasks */
    App_Tasks ();
}
  
```

```

void App_Tasks ( void )
{
    switch (testState)
    {
        /* Wait in Init state till the SYS TMR Module is in running mode */
        case TEST_STATE_Init:
            if (SYS_MODULE_RUNNING == SYS_TMR_Status(sysTmrObject))
            {
                /* SYS TMR is in running mode */
                testState = TEST_STATE_Config_Single;
            }
            else
            {
                testState = TEST_STATE_Init;
            }
            break;

        case TEST_STATE_Config_Single:
            /* Activate one shot/single callback */
            handle = SYS_TMR_CallbackSingle (120, &Test_Callback);
            testState = TEST_STATE_Count_Read;
            break;

        case TEST_STATE_Count_Read:
            testCount1 = SYS_TMR_TickCountGet ();
            break;
    }
}

// Callback which will be called repetitively after the configured
// time elapses
void Test_Callback ( void )
{
    PORTA = ~PORTA;
}

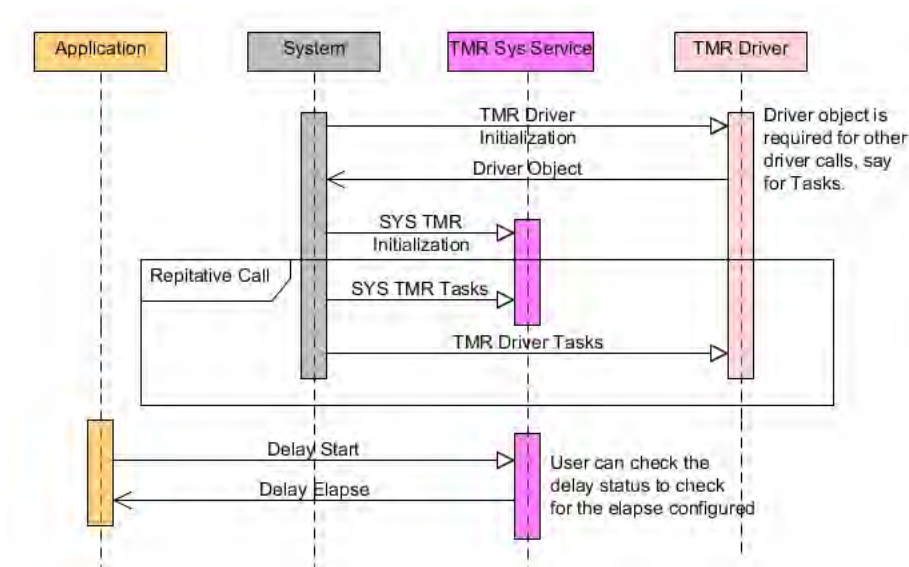
```

Note: The configuration '**SYS_TMR_INTERRUPT_MODE**' has no impact on this feature.

5.7.8.4.3.4 Delays

This section deals with the delay registration and its usage. The mechanism depicted supports only non-blocking delays.

The following diagram depicts the interaction across various modules for the delay feature,



Steps involved in registering and using the feature:

1. Initial requirement is to have the Timer System Service module in the mode 'SYS_MODULE_RUNNING'. The user can call the function [SYS_TMR_Status](#) to get the current status information of the module.
2. Delay feature can be registered by calling the function '[SYS_TMR_DelayMS](#)' with the delay parameter. The delay parameter has to be integral multiple of the Timer driver alarm period.

Note: The API '[SYS_TMR_DelayMS](#)' returns a valid handle after successful registration. This handle can be later used by the user to check the status of the delay using the API '[SYS_TMR_DelayStatusGet](#)'.

The following code fragment gives a typical use case for this feature,

```

case DELAY_START_STATE:
    handle = SYS_TMR_DelayMS (500);
    state = DELAY_CHECKSTATE;
break;

case DELAY_CHECKSTATE:
    if (SYS_TMR_DELAY_EXPIRED == SYS_TMR_DelayStatusGet(handle))
    {
        state = DELAY_START_STATE;
    }
    else
    {
        state = DELAY_CHECKSTATE;
    }
break;
  
```

Note: For the usage of the Delay feature, the configuration switch '[SYS_TMR_INTERRUPT_MODE](#)' needs to be set to 'true'.

5.7.8.4.3.5 Miscellaneous

Tick Counts Information: The API '[SYS_TMR_TickCountGet](#)' provides the current tick information.

Alarm Period: The previously configured alarm period can be obtained using the interface '[SYS_TMR_AlarmPeriodGet](#)'. Period chosen for the periodic/one shot callback should be a multiple of the driver's alarm period. Say if the alarm period is 10ms, then the period should be a multiple of 10.

5.7.8.4.3.6 Examples

Sample application tasks:

Periodic callback with only one period with 120 ms period:

```
switch (state)
{
    case STATE_Init:
        if (SYS_MODULE_RUNNING == SYS_TMR_Status(sysTmrObject))
        {
            state = STATE_Config_Periodic;
        }
        else
        {
            state = STATE_Init;
        }
        break;

    case STATE_Config_Periodic:
        handle = SYS_TMR_CallbackPeriodic (120, &Test_Callback);
        state = STATE_Count_Read;
        break;

    case STATE_Count_Read:
        testCount = SYS_TMR_TickCountGet ();
        state = STATE_Count_Read;
        break;
}
```

Periodic callback with multiple periods 120 ms and 250 ms:

```
switch (state)
{
    case STATE_Init:
        if (SYS_MODULE_RUNNING == SYS_TMR_Status(sysTmrObject))
        {
            state = STATE_Config_Periodic;
        }
        else
        {
            state = STATE_Init;
        }
        break;

    case STATE_Config_Periodic:
        handle1 = SYS_TMR_CallbackPeriodic (120, &Test_Callback1);
        handle2 = SYS_TMR_CallbackPeriodic (250, &Test_Callback2);
        state = STATE_Count_Read;
        break;

    case STATE_Count_Read:
        testCount = SYS_TMR_TickCountGet ();
        state = STATE_Count_Read;
        break;
}

void Test_Callback1 ( void )
{
    /* Assume RA0 is connected to one LED */
    PORTAbits.RA0 = ~PORTAbits.RA0;
}

void Test_Callback2 ( void )
{
    /* Assume RA1 is connected to another LED */
```

```
    PORTAbits.RA1 = ~PORTAbits.RA1;
}
```

One-Shot Callback:

```
switch (state)
{
    case STATE_Init:
        if (SYS_MODULE_RUNNING == SYS_TMR_Status(sysTmrObject))
        {
            state = STATE_Config_Single;
        }
        else
        {
            state = STATE_Init;
        }
        break;

    case STATE_Config_Single:
        SYS_TMR_CallbackSingle (500, &Test_Callback);
        state = STATE_Count_Read;
        break;

    case STATE_Count_Read:
        testCount = SYS_TMR_TickCountGet ();
        state = STATE_Count_Read;
        break;
}
```

Delay:

```
switch (state)
{
    case STATE_Init:
        if (SYS_MODULE_RUNNING == SYS_TMR_Status(sysTmrObject))
        {
            state = STATE_DelayStart;
        }
        else
        {
            state = STATE_Init;
        }
        break;

    case STATE_DelayStart:
        dlHandle = SYS_TMR_DelayMS (500);
        state = STATE_DelayCheck;
        break;

    case STATE_DelayCheck:
        if (SYS_TMR_DELAY_EXPIRED == SYS_TMR_DelayStatusGet(dlHandle))
        {
            state = STATE_DelayStart;
        }
        else
        {
            state = STATE_DelayCheck;
        }
        break;
}
```

5.7.8.5 Configuring the Library

Macros

	Name	Description
	SYS_TMR_ERROR_TOLERANCE	Sets the error tolerance in milliseconds.
	SYS_TMR_INTERRUPT_MODE	Activates the Interrupt mode.
	SYS_TMR_SINGLE_PERIODIC_EVENT	Sets the type of the periodic event.
	SYS_TMR_MAX_PERIODIC_EVENTS	Sets the maximum periodic multi-event callbacks supported.

Description

The configuration of the Timer System Service is based on the file `sys_config.h`.

This header file contains the configuration selection for the Timer System Service build. Based on the selections made, the Timer System Service will support or not support selected features. These configuration settings will apply to all instances of the Timer System Service.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.7.8.5.1 SYS_TMR_ERROR_TOLERANCE Macro

C

```
#define SYS_TMR_ERROR_TOLERANCE 0
```

Description

Error Tolerance configuration

This macro sets the error tolerance in milliseconds with reference to the configured alarm period of the timer driver.

Remarks

None.

5.7.8.5.2 SYS_TMR_INTERRUPT_MODE Macro

C

```
#define SYS_TMR_INTERRUPT_MODE false
```

Description

Interrupt Mode configuration

This macro sets up the interrupt mode, which can [accept](#) the following values:

- true - Interrupt mode
- false - No Interrupt mode

Remarks

The delay feature works only if the Interrupt mode is set to 'true' and the respective Timer has been configured for Interrupt mode. This feature has been designed to provide a non-blocking delay.

5.7.8.5.3 SYS_TMR_SINGLE_PERIODIC_EVENT Macro

C

```
#define SYS_TMR_SINGLE_PERIODIC_EVENT true
```

Description

Periodic event type configuration

This macro sets the type of the periodic event, which can [accept](#) the following values:

- true - Single periodic event
- false - Multiple periodic events

Remarks

If multiple periodic events are enabled, the switch [SYS_TMR_MAX_PERIODIC_EVENTS](#) has to be configured to support the maximum permitted events.

5.7.8.5.4 SYS_TMR_MAX_PERIODIC_EVENTS Macro

C

```
#define SYS_TMR_MAX_PERIODIC_EVENTS 5
```

Description

Maximum periodic events configuration

This macro sets the maximum periodic multi-event callbacks.

Remarks

This requires multiple periodic events enabled through the switch [SYS_TMR_SINGLE_PERIODIC_EVENT](#).

5.7.8.6 Building the Library

5.7.8.7 Library Interface

Data Types and Constants

	Name	Description
	SYS_TMR_CALLBACK	Defines a pointer to a callback function.
	SYS_TMR_DELAY_STATUS	Defines the callback service type.
	SYS_TMR_HANDLE	Identifies a particular registered event instance.
	SYS_TMR_INIT	Identifies the system timer initialize structure.
	SYS_TMR_HANDLE_INVALID	Identifies the invalid handle of the system timer.
	SYS_TMR_INDEX_0	Timer System Service index definitions.
	SYS_TMR_RUNNING	Identifies the current status/state of the system timer.

Miscellaneous Functions

	Name	Description
≡	SYS_TMR_AlarmPeriodGet	Returns the previously configured alarm period.
≡	SYS_TMR_TickCountGet	Provides the current counter value.

System Level Interaction Functions

	Name	Description
≡	SYS_TMR_Deinitialize	Deinitializes the specific module instance of the TMR module
≡	SYS_TMR_Initialize	Initializes hardware and data for the instance of the Timer module and opens the specific module instance.
≡	SYS_TMR_Reinitialize	Reinitializes and refreshes the hardware for the instance of the Timer module.
≡	SYS_TMR_Status	Returns status of the specific instance of the Timer module.
≡	SYS_TMR_Tasks	Maintains the system timer's state machine and implements its ISR.

Timed Callback Functions

	Name	Description
≡	SYS_TMR_CallbackPeriodic	The Periodic callback is registered and started using this function.
≡	SYS_TMR_CallbackSingle	The one-shot/single callback is registered and started using this function.
≡	SYS_TMR_CallbackStop	Stops the periodic callback service.

Timed Delay Functions

	Name	Description
≡	SYS_TMR_DelayMS	Registers an event for the specified delay.
≡	SYS_TMR_DelayStatusGet	Intimates if the previously configured delay has elapsed.

Description

This section describes the APIs of the Timer System Service Library.

Refer to each section for a detailed description.

5.7.8.7.1 System Level Interaction Functions**5.7.8.7.1.1 SYS_TMR_Deinitialize Function****C**

```
void SYS_TMR_Deinitialize(
    SYS_MODULE_OBJ object
);
```

Description

This function deinitializes the specific module instance disabling its operation (and any hardware for driver modules). Resets all of the internal data structures and fields for the specified instance to the default settings.

Preconditions

The [SYS_TMR_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
object	SYS TMR object handle, returned from SYS_TMR_Initialize

Returns

None.

Example

```
SYS_MODULE_OBJ    object;    // Returned from SYS_TMR_Initialize
SYS_STATUS        status;
```

```
SYS_TMR_Deinitialize (object);
```

```
status = SYS_TMR_Status (object);
if (SYS_MODULE_DEINITIALIZED == status)
{
    // Check again later if you need to know
    // when the SYS TMR is deinitialized.
}
```

Remarks: Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

5.7.8.7.1.2 SYS_TMR_Initialize Function**C**

```
SYS_MODULE_OBJ SYS_TMR_Initialize(
    const SYS_MODULE_INDEX index,
    const SYS_MODULE_INIT * const init
);
```

Description

This function initializes hardware for the instance of the Timer module, using the specified hardware initialization data. It also initializes any internal data structures.

Preconditions

None.

Parameters

Parameters	Description
index	Index for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the sys timer. This pointer may be null if no data is required because static overrides have been provided.

Returns

If successful, returns a valid handle to an object. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#). The returned object must be passed as argument to [SYS_TMR_Reinitialize](#), [SYS_TMR_Deinitialize](#), [SYS_TMR_Tasks](#) and [SYS_TMR_Status](#) routines.

Example

```
SYS_MODULE_OBJ    objectHandle;
SYS_TMR_INIT      initConfig;

// Populate the timer initialization structure
initConfig.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
initConfig.drvIndex             = DRV_TMR_INDEX_0;
initConfig.alarmPeriod           = 100;

objectHandle = SYS_TMR_Initialize (SYS_TMR_INDEX_0, (SYS_MODULE_INIT*)&initConfig);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Remarks: This routine should only be called once during system initialization unless SYS_TMR_Deinitialize is first called to deinitialize the device instance before reinitializing it. If the system was already initialized it safely returns without causing any disturbance.

5.7.8.7.1.3 SYS_TMR_Reinitialize Function

C

```
void SYS_TMR_Reinitialize(
    SYS_MODULE_OBJ object,
    const SYS_MODULE_INIT * const init
);
```

Description

This function reinitializes and refreshes the hardware for the instance of the Timer module using the supplied data. It modifies the internal data structure.

Preconditions

The [SYS_TMR_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
object	Identifies the SYS TMR Object returned by the Initialize interface
init	Pointer to the data structure containing any data necessary to initialize the hardware

Returns

None

Example

```
SYS_MODULE_OBJ  objectHandle;
SYS_TMR_INIT    initConfig;
SYS_STATUS      tmrStatus;

// Populate the timer initialization structure
initConfig.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
initConfig.drvIndex              = DRV_TMR_INDEX_0;
initConfig.alarmPeriod           = 100;

SYS_TMR_Reinitialize (objectHandle, (SYS_MODULE_INIT*)&initConfig);

tmrStatus = SYS_TMR_Status (object);
if (SYS_STATUS_BUSY == tmrStatus)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= tmrStatus)
{
    // Handle error
}
```

Remarks: This operation uses the same initialization data structure as the SYS_TMR_Initialize operation. This operation can be used to change the power state of a TMR module. This function can be called multiple times to reinitialize the module. This operation uses the same initialization data structure as the Initialize operation. This operation can also be used to refresh the hardware registers as defined by the initialization data.

5.7.8.7.1.4 SYS_TMR_Status Function

C

```
SYS_STATUS SYS_TMR_Status(  
    SYS_MODULE_OBJ object  
);
```

Description

This function returns the status of the specific module instance disabling its operation (and any hardware for driver modules).

Preconditions

The [SYS_TMR_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
object	SYS TMR object handle, returned from SYS_TMR_Initialize

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to [accept](#) new operations

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been de-initialized

This value is less than SYS_STATUS_ERROR.

Example

```
SYS_MODULE_OBJ    object;    // Returned from SYS_TMR_Initialize  
SYS_STATUS        tmrStatus;  
  
tmrStatus = SYS_TMR_Status (object);  
else if (SYS_STATUS_ERROR >= tmrStatus)  
{  
    // Handle error  
}
```

Remarks: Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

5.7.8.7.1.5 SYS_TMR_Tasks Function

C

```
void SYS_TMR_Tasks(  
    SYS_MODULE_OBJ object  
);
```

Description

This function is used to maintain the system timer's internal state machine and implement its ISR for interrupt-driven implementations.

Preconditions

The [SYS_TMR_Initialize](#) function must have been called for the specified TMR driver instance.

Parameters

Parameters	Description
object	SYS TMR object handle, returned from SYS_TMR_Initialize

Returns

None

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine ([SYS_Tasks](#)) or by the appropriate raw ISR.

Example

```
SYS_MODULE_OBJ    object;    // Returned from SYS_TMR_Initialize

while (true)
{
    SYS_TMR_Tasks (object);

    // Do other tasks
}
```

5.7.8.7.2 Timed Callback Functions

5.7.8.7.2.1 SYS_TMR_CallbackPeriodic Function

C

```
SYS_TMR_HANDLE SYS_TMR_CallbackPeriodic(
    unsigned int period,
    SYS_TMR_CALLBACK callback
);
```

Description

This function registers the periodic callback service corresponding to the configuration parameters passed by the user.

Preconditions

The [SYS_TMR_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
period	Periodic delay in milliseconds
callback	Pointer to a callback routine that will be called periodically

Returns

A valid event handler of type [SYS_TMR_HANDLE](#) is returned.

Example

```
SYS_TMR_HANDLE handle;
unsigned int period = 20;

void Test_Callback1 (void);

handle = SYS_TMR_CallbackPeriodic (period, &Test_Callback1);
```

Remarks: The Period chosen should be a multiple of the driver's alarm period. For example, if the alarm period is 10 ms, the period should be a multiple of 10.

5.7.8.7.2.2 SYS_TMR_CallbackSingle Function

C

```
SYS_TMR_HANDLE SYS_TMR_CallbackSingle(  
    unsigned int period,  
    SYS_TMR_CALLBACK callback  
);
```

Description

This function registers a one-shot/single callback service corresponding to the configuration parameters passed by the user.

Preconditions

The [SYS_TMR_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
period	Periodic delay in milliseconds
callback	Pointer to a callback routine which will be called periodically

Returns

A valid event handler of type [SYS_TMR_HANDLE](#) is returned.

Example

```
SYS_TMR_HANDLE handle;  
unsigned int period = 20;  
void Test_Callback1 (void);
```

```
handle = SYS_TMR_CallbackSingle (period, &Test_Callback1);  
Remarks: None.
```

5.7.8.7.2.3 SYS_TMR_CallbackStop Function

C

```
void SYS_TMR_CallbackStop(  
    SYS_TMR_HANDLE handle  
);
```

Description

This function stops the previously registered periodic callbacks service.

Preconditions

The [SYS_TMR_CallbackPeriodic](#) function should have been called before calling this function.

Parameters

Parameters	Description
handle	A valid callback service handle, returned from the system API SYS_TMR_CallbackPeriodic

Returns

None.

Example

```
SYS_TMR_HANDLE handle;  
uint16_t period = 20;
```

```
void Test_Callback1 (void);  
handle = SYS_TMR_CallbackPeriodic (period, &Test_Callback1);  
...  
...  
SYS_TMR_CallbackStop (handle);  
Remarks:    None.
```

5.7.8.7.3 Timed Delay Functions

5.7.8.7.3.1 SYS_TMR_DelayMS Function

C

```
SYS_TMR_HANDLE SYS_TMR_DelayMS(  
    unsigned int delay  
);
```

Description

This function registers an event for the specified delay.

Preconditions

The [SYS_TMR_Initialize](#) function should have been called before calling this function.

Parameters

Parameters	Description
delay	Indicates the delay required in milliseconds

Returns

A valid event handler of type [SYS_TMR_HANDLE](#) is returned.

Example

```
SYS_TMR_DelayStart (50);  
Remarks:    Works only in Interrupt mode.
```

5.7.8.7.3.2 SYS_TMR_DelayStatusGet Function

C

```
SYS_TMR_DELAY_STATUS SYS_TMR_DelayStatusGet(  
    SYS_TMR_HANDLE handle  
);
```

Description

This function intimates if the previously configured delay has elapsed.

Preconditions

The [SYS_TMR_DelayStart](#) function should have been called before calling this function.

Parameters

Parameters	Description
handle	A valid callback service handle, returned from the system API SYS_TMR_DelayMS

Returns

One of the possible values of [SYS_TMR_DELAY_STATUS](#).

Example

```
if (SYS_TMR_DELAY_EXPIRED == SYS_TMR_DelayStatusGet ())  
    // Do Something  
Remarks:    None.
```

5.7.8.7.4 Miscellaneous Functions

5.7.8.7.4.1 SYS_TMR_AlarmPeriodGet Function

C

```
uint32_t SYS_TMR_AlarmPeriodGet();
```

Description

This function obtains the previously configured alarm period.

Preconditions

The [SYS_TMR_Initialize](#) function should have been called before calling this function.

Returns

uint32_t - Alarm period in milliseconds

Example

```
uint32_t period;  
period = SYS_TMR_AlarmPeriodGet();  
Remarks:    None.
```

5.7.8.7.4.2 SYS_TMR_TickCountGet Function

C

```
uint32_t SYS_TMR_TickCountGet();
```

Description

This function provides the current counter value.

Preconditions

The [SYS_TMR_CallbackPeriodic](#) function should have been called before calling this function.

Returns

uint32_t - Current system timer tick count value

Example

```
uint32_t count;  
count = SYS_TMR_TickCountGet ();  
Remarks:    None.
```

5.7.8.7.5 Data Types and Constants

5.7.8.7.5.1 SYS_TMR_CALLBACK Type

C

```
typedef void (* SYS_TMR_CALLBACK)(void);
```

Description

SYS TMR Callback Function

This data type defines a pointer callback function.

Remarks

None.

5.7.8.7.5.2 SYS_TMR_DELAY_STATUS Enumeration

C

```
typedef enum {
    SYS_TMR_DELAY_IDLE,
    SYS_TMR_DELAY_EXPIRED,
    SYS_TMR_DELAY_NOT_EXPIRED,
    SYS_TMR_DELAY_ERROR
} SYS_TMR_DELAY_STATUS;
```

Description

Callback service type enumeration

This data type defines the callback service type.

Members

Members	Description
SYS_TMR_DELAY_IDLE	Delay Idle
SYS_TMR_DELAY_EXPIRED	Delay Expired
SYS_TMR_DELAY_NOT_EXPIRED	Delay not yet expired
SYS_TMR_DELAY_ERROR	delay error

Remarks

None.

5.7.8.7.5.3 SYS_TMR_HANDLE Type

C

```
typedef int8_t SYS_TMR_HANDLE;
```

Description

SYS TMR Handle

This event handle identifies a registered instance of an event. Every time the application that tries to access the paramateres with respect to a particular event, shall used this event handle to refer to that event.

Remarks

None.

5.7.8.7.5.4 SYS_TMR_INIT Structure

C

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    SYS_MODULE_INDEX drvIndex;
    uint32_t alarmPeriod;
} SYS_TMR_INIT;
```

Description

SYS TMR Initialize structure

This structure identifies the system timer initialize structure.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX drvIndex;	Driver Module index
uint32_t alarmPeriod;	Alarm period in ms

Remarks

None.

5.7.8.7.5.5 SYS_TMR_HANDLE_INVALID Macro**C**

```
#define SYS_TMR_HANDLE_INVALID ((int8_t)-1)
```

Description

SYS Timer invalid handle macro definition

This enumeration identifies the invalid handle of the system timer.

Remarks

None.

5.7.8.7.5.6 SYS_TMR_INDEX_0 Macro**C**

```
#define SYS_TMR_INDEX_0 0
```

Description

SYS Timer Module Index Numbers

These constants provide Timer System Service index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

5.7.8.7.5.7 SYS_TMR_RUNNING Macro**C**

```
#define SYS_TMR_RUNNING (SYS_STATUS_READY + 1)
```

Description

SYS Timer Module specific Status

This enumeration identifies the current status/state of the system timer.

Remarks

This enumeration is the return type for the system-level status routine defined by each device driver or system module (for example: [SYS_TMR_Status](#)).

5.7.8.8 Files

Files

Name	Description
sys_tmr.h	Timer System Service interface definition.
sys_tmr_config.h	Contains configuration definitions that are common to timer system services and aggregates the configuration files for the system services.

Description

5.7.8.8.1 sys_tmr.h

Timer System Service Interface Definition

This file contains the interface definition for the Timer System Service. It provides a way to interact with the Timer subsystem to manage the timing requests supported by the system.

Enumerations

	Name	Description
	SYS_TMR_DELAY_STATUS	Defines the callback service type.

Functions

	Name	Description
	SYS_TMR_AlarmPeriodGet	Returns the previously configured alarm period.
	SYS_TMR_CallbackPeriodic	The Periodic callback is registered and started using this function.
	SYS_TMR_CallbackSingle	The one-shot/single callback is registered and started using this function.
	SYS_TMR_CallbackStop	Stops the periodic callback service.
	SYS_TMR_Deinitialize	Deinitializes the specific module instance of the TMR module
	SYS_TMR_DelayMS	Registers an event for the specified delay.
	SYS_TMR_DelayStatusGet	Intimates if the previously configured delay has elapsed.
	SYS_TMR_Initialize	Initializes hardware and data for the instance of the Timer module and opens the specific module instance.
	SYS_TMR_Reinitialize	Reinitializes and refreshes the hardware for the instance of the Timer module.
	SYS_TMR_Status	Returns status of the specific instance of the Timer module.
	SYS_TMR_Tasks	Maintains the system timer's state machine and implements its ISR.
	SYS_TMR_TickCountGet	Provides the current counter value.

Macros

	Name	Description
	SYS_TMR_HANDLE_INVALID	Identifies the invalid handle of the system timer.
	SYS_TMR_INDEX_0	Timer System Service index definitions.
	SYS_TMR_RUNNING	Identifies the current status/state of the system timer.

Structures

	Name	Description
	SYS_TMR_INIT	Identifies the system timer initialize structure.

Types

	Name	Description
	<u>SYS_TMR_CALLBACK</u>	Defines a pointer to a callback function.
	<u>SYS_TMR_HANDLE</u>	Identifies a particular registered event instance.

File Name

sys_tmr.h

Company

Microchip Technology Inc.

5.7.8.8.2 sys_tmr_config.h

Timer System Service Configuration Definitions for the Template Version

This file contains configuration definitions that are common to timer drivers and aggregates the configuration files for the system services.

Macros

	Name	Description
	<u>SYS_TMR_ERROR_TOLERANCE</u>	Sets the error tolerance in milliseconds.
	<u>SYS_TMR_INTERRUPT_MODE</u>	Activates the Interrupt mode.
	<u>SYS_TMR_MAX_PERIODIC_EVENTS</u>	Sets the maximum periodic multi-event callbacks supported.
	<u>SYS_TMR_SINGLE_PERIODIC_EVENT</u>	Sets the type of the periodic event.

File Name

sys_tmr_config.h

Company

Microchip Technology Inc.

5.7.9 Watchdog Timer (WDT) System Service Library

5.7.9.1 Introduction

Watchdog Timer (WDT) System Service Library**for****Microchip Microcontrollers**

The primary function of the Watchdog Timer (WDT) is to reset the microcontroller, in the event of a software malfunction, by resetting the device if it has not been cleared by software. To ensure that application doesn't hang, the application is required to reset the timer periodically. It can also be used to wake the device from Sleep or Idle mode. The WDT is a free-running timer which uses the low-power RC oscillator and requires no external components. Therefore, the WDT will continue to operate even if the system's primary clock source is stopped.

This library provides a low-level abstraction of the Watchdog Timer System Service Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

5.7.9.2 Release Notes

MPLAB Harmony Version: v0.70b **Watchdog Timer (WDT) System Service Library Version :** 0.1 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.7.9.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.7.9.4 Using the Library

This topic describes the basic architecture of the Watchdog Timer System Service Library and provides information and examples on how to use it.

Interface Header File: [sys_wdt.h](#)

The interface to the Watchdog Timer System Service library is defined in the "[sys_wdt.h](#)" header file. This file is included by the "sys.h" file. Any C language source (.c) file that uses the Watchdog Timer System Service library should include "sys.h".

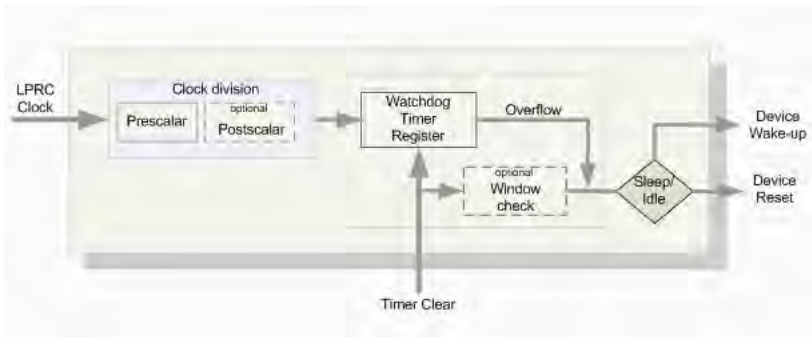
Library File:

The Watchdog Timer System Service library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the System Service interacts with the framework.

5.7.9.4.1 Abstraction Model

Watchdog Timer Software Abstraction Block Diagram



Watchdog timer uses internal Low Power RC(LPRC) oscillator as the source of clock. The clock is divided by the configured prescaler value. There may be one more postscaler divisors and these should be set through the 'configuration bits'. The divided clock is then used to increment a counter. If the software does not clear the counter in time the counter overflows and that will result in reset in normal mode. In case of Sleep or Idle mode the overflow will result in device wakeup.

In case of windowed mode, resetting the counter when the count is not in the window specified will also lead to a reset.

5.7.9.4.2 Library Overview

Refer to the [System Services Overview](#) section for how the system services operates in a system.

The library interface routines are divided into various sub-sections, each sub-section addresses one of the blocks of the overall operation of the Watchdog Timer module.

Library Interface Section	Description
Enable and Disable Functions	APIs to enable and disable the Watchdog Timer
Clearing the Timer Functions	API to clear the Watchdog Timer

5.7.9.4.3 How the Library Works

Initialization of Watchdog Timer:

In case of devices which has 'Window mode', use the argument to control the window mode. The argument will be discarded in case of devices which doesn't has 'Window mode'.

```
SYS_WDT_Enable(true);  
  
//Some code  
SYS_WDT_Enable(false);  
  
//Some code  
SYS_WDT_Disable();
```

Note: For all the above code to work, the watchdog timer should not be controlled through configuration bits.

Service the Watchdog Timer:

To ensure the normal operation of the system, the software must clear the Watchdog timer periodically.

```
SYS_WDT_TimerClear();
```

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

5.7.9.5 Configuring the Library

Macros

	Name	Description
	WDT_PLIB_ID	Configures the module's ID used by the peripheral library.

Description

The configuration of the Watchdog Timer System Service is based on the file `sys_config.h`

This header file contains the configuration selection for the Watchdog Timer System Service. Based on the selections made, the Watchdog Timer System Service will support or not support selected features. These configuration settings will apply to all instances of the Watchdog Timer System Service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.7.9.5.1 WDT_PLIB_ID Macro

C

```
#define WDT_PLIB_ID WDT_ID_1
```

Description

Watchdog timer module ID used by the Peripheral Library

This macro configures the module's ID used by the peripheral library.

Remarks

None.

5.7.9.6 Building the Library

This section list the files that are available in the `\src` of the Watchdog Timer System Service driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.7.9.7 Library Interface

Clearing the Timer Functions

	Name	Description
	SYS_WDT_TimerClear	Reset the WDT.

Enable and Disable Functions

	Name	Description
	SYS_WDT_Disable	Disables the WDT if it is enabled in software.
	SYS_WDT_Enable	Enables the WDT. The argument 'windowModeEnable' will be used only for those devices that support 'window mode'. Otherwise, it will be discarded.

Description

This section describes the Application Programming Interface (API) functions of the Watchdog Timer System Service.

Refer to each section for a detailed description.

5.7.9.7.1 Enable and Disable Functions

5.7.9.7.1.1 SYS_WDT_Disable Function

C

```
void SYS_WDT_Disable();
```

Description

This function disables the WDT if it is enabled in software. If the WDT is enabled through 'configuration bits' it cannot be disabled using this function.

Preconditions

The WDT should be disabled through 'configuration bits'.

Returns

None.

Remarks

The example code doesn't include the settings that should be done through configuration bits.

Example

```
SYS_WDT_Disable();
```

5.7.9.7.1.2 SYS_WDT_Enable Function

C

```
void SYS_WDT_Enable(  
    bool windowModeEnable  
);
```

Description

This function enables the WDT. The argument 'windowModeEnable' will be used only for those devices that support 'window mode'. Otherwise, the argument will be discarded. This function could be called multiple times to enable/disable the 'window mode'.

Preconditions

None.

Parameters

Parameters	Description

5.7.9.7.2 Clearing the Timer Functions

5.7.9.7.2.1 SYS_WDT_TimerClear Function

C

```
void SYS_WDT_TimerClear();
```

Description

This function clears the WDT counter. The WDT should be cleared periodically before the count overflows and forces the device to Reset.

Preconditions

None.

Returns

None.

Remarks

Clearing the WDT before the count reaches the window, will cause a reset in Windowed mode.

The example code doesn't include the settings that should be done through Configuration bits.

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is supported.

Example

```
//Application loop
while(1)
{
    SYS_WDT_TimerClear();
    //user code
}
```

5.7.9.8 Files

Files

Name	Description
sys_wdt.h	Watchdog Timer (WDT) System Service interface definition.
sys_wdt_config.h	Watchdog Timer (WDT) System Service interface definition.

Description

5.7.9.8.1 sys_wdt.h

Watchdog Timer System Service Interface Definition

This file contains the interface definition for the WDT System Service. It provides a way to interact with the WDT subsystem to manage the timing requests supported by the system.

Functions

	Name	Description
	SYS_WDT_Disable	Disables the WDT if it is enabled in software.
	SYS_WDT_Enable	Enables the WDT. The argument 'windowModeEnable' will be used only for those devices that support 'window mode'. Otherwise, it will be discarded.
	SYS_WDT_TimerClear	Reset the WDT.

File Name

sys_wdt.h

Company

Microchip Technology Inc.

5.7.9.8.2 sys_wdt_config.h

Watchdog Timer System Service Interface Definition

This file contains the interface definition for the WDT System Service. It provides a way to interact with the Watchdog timer subsystem to manage the timing requests supported by the system

Macros

	Name	Description
	<u>WDT_PLIB_ID</u>	Configures the module's ID used by the peripheral library.

File Name

sys_wdt.h

Company

Microchip Technology Inc.

5.8 TCP/IP Stack Library Help

5.8.1 TCP/IP Library Overview

5.8.1.1 Introduction

This topic provides an overview of the TCP/IP Stack in MPLAB Harmony.

Description

The MPLAB Harmony TCP/IP Stack provides a foundation for embedded network applications by handling most of the interaction required between the physical network port and your application. It includes modules for several commonly used application layers, including HTTP for serving web pages, SMTP for sending e-mails, SNMP for providing status and control, Telnet, TFTP, Serial-to-Ethernet and much more. In addition, the stack includes light-weight and high-performance implementations of the TCP and UDP transport layers, as well as other supporting modules such as IP, ICMP, DHCP, ARP, and DNS.

This help file serves two purposes. The first is to be a guide for first-time users of the TCP/IP Stack. The Getting Started section begins a series of pages to help you become familiar with the stack and configure it for use on a Microchip development board.

The second purpose is to serve as a programmer's reference guide to the features and APIs available in the TCP/IP Stack.

5.8.1.2 Release Notes

MPLAB Harmony Version: v0.70b **TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Description

Resource Usage and Network Metrics for the PIC32MX

Memory and Flash usage

Module	RAM No Optimization	Flash No Optimization	RAM Optimize for Size	Flash Optimize for Size	RAM Optimized for Speed	Flash Optimized for Speed
Core (arp, tcp manager, tcp helpers, hash functions, etc.)	356	51,380	352	24,772	352	34,812
Berkeley API	116	7,464	116	4,116	116	7,080

Commands for TCP Configuration	120	18,348	120	13,104	124	14,556
DHCP Client	40	11,328	40	6,036	44	7,676
DHCP Server	76	15,188	76	7,464	76	10,456
DNS Client	66	6,868	62	3,496	68	4,368
DNS Server	86	2,276	86	1,052	88	1,192
FTP Server	40	11,368	40	5,384	40	6,548
HTTP Server	24	19,964	24	10,404	24	13,128
ICMPv4 Server	20	2,472	20	1,112	20	1,264
ICMPv6 Server	20	11,852	20	6,332	20	7,336
IPerf	252	22,344	252	12,136	252	14,240
IPv4	40	4,572	40	2,348	40	2,640
IPv6	224	45,488	224	25,060	224	40,480
Microchip Embedded Ethernet Device Discoverer	36	3,280	36	1,764	36	2,164
NetBIOS Name Resolution	12	2,884	12	1,372	12	2,024
Reboot Server	12	1,048	12	576	12	764
SMTP Client	98	8,252	102	4,324	104	6,052
SNMP Server	72	30,616	68	16,244	72	28,900
SNMPv3 Server	864	32,600	864	15,580	864	27,052
SNTP Client	72	2,796	72	1,572	72	1,780
SSL	72	33,416	72	18,096	72	49,252
TCP Protocol	28	42,268	28	21,604	32	28,556
Telnet Server	28	3,708	28	1,908	28	2,236
UDP Protocol	36	17,052	36	9,308	40	11,884
Zero Config for Local Link	25	22,296	25	10,892	25	13,116

Network Performance measured with iPerf

The following commands were used to generate the performance numbers:

- UDP/IP Client:
 - **PIC32 Board:** iperf -c <PC IP Address> -u -b 100M -i 5 -t 20
 - **PC:** iperf -s -u -i 5
- UDP/IP Server:
 - **PIC32 Board:** iperf -s -u -i 5
 - **PC:** iperf -c mchpboard_e -u -b 100M -i 5 -t 20
- TCP/IP Client
 - **PIC32 Board:** iperf -c <PC IP Address> -t 10 -i 5 -l 1024 -x 100M -M 1460
 - **PC:** iperf -s -i 5 -m -w 10K -N
- TCP/IP Server
 - **PIC32 Board:** iperf -s -i 5 -x 100M

- **PC:** iperf -c mchpboard_e -t 10 -i 5 -l 1024

UDP/IP Client	UDP/IP Server	TCP/IP Client	TCP/IP Server
89.3 MBits/sec	95.7 MBits/sec	50.1 MBits/sec	60.3 MBits/sec

Resource Usage and Network Metrics for the PIC32MZ

Memory and Flash usage

Module	RAM No Optimization	Flash No Optimization	RAM Optimized for Size	Flash Optimized for Size	RAM Optimized for Speed	Flash Optimized for Speed
Core (arp, tcp manager, tcp helpers, hash functions, etc.)	356	51,112	352	24,868	352	34,632
Berkeley API	116	7,348	116	4,116	116	7,056
Commands for TCP Configuration	120	18,232	120	13,084	124	14,472
DHCP Client	40	11,276	40	6,040	44	7,660
DHCP Server	76	15,072	76	7,464	76	10,396
DNS Client	66	6,752	62	3,488	68	4,324
DNS Server	86	2,160	86	1,052	88	1,172
FTP Server	40	11,252	40	5,372	40	6,516
HTTP Server	24	19,848	24	10,284	24	12,988
ICMPv4 Server	20	2,356	20	1,112	20	1,240
ICMPv6 Server	20	12,336	20	6,668	20	7,572
IPerf	252	23,132	252	12,508	252	14,396
IPv4	40	4,576	40	2,368	40	2,628
IPv6	224	45,896	224	24,988	224	40,296
Microchip Embedded Ethernet Device Discoverer	36	3,164	36	1,748	36	1,980
NetBIOS Name Resolution	12	2,768	12	1,372	12	2,000
Reboot Server	12	932	12	572	12	744
SMTP Client	98	8,136	102	4,304	104	6,008
SNMP Server	72	30,500	68	16,188	72	28,500
SNMPv3 Server	864	32,368	864	15,396	864	26,804
SNTP Client	72	2,780	72	1,592	72	1,772
SSL	72	33,068	72	18,068	72	49,156
TCP Protocol	28	42,496	28	21,604	32	28,620
Telnet Server	28	3,592	28	1,896	28	2,156
UDP Protocol	36	17,320	36	9,360	40	11,812
Zero Config for Local Link	25	22,488	25	11,000	25	13,156

Network Performance measured with iPerf

The following commands were used to generate the performance numbers:

- UDP/IP Client:
 - **PIC32 Board:** iperf -c <PC IP Address> -u -b 100M -i 5 -t 20
 - **PC:** iperf -s -u -i 5
- UDP/IP Server:
 - **PIC32 Board:** iperf -s -u -i 5
 - **PC:** iperf -c mchpboard_e -u -b 100M -i 5 -t 20
- TCP/IP Client
 - **PIC32 Board:** iperf -c <PC IP Address> -t 10 -i 5 -l 1024 -x 100M -M 1460
 - **PC:** iperf -s -i 5 -m -w 10K -N
- TCP/IP Server
 - **PIC32 Board:** iperf -s -i 5 -x 100M
 - **PC:** iperf -c mchpboard_e -t 10 -i 5 -l 1024

UDP/IP Client	UDP/IP Server	TCP/IP Client	TCP/IP Server
71.6 MBits/sec	95.7 MBits/sec	46.6 MBits/sec	44.5 MBits/sec

5.8.1.3 SW License Agreement

MICROCHIP IS WILLING TO LICENSE THE ACCOMPANYING SOFTWARE AND DOCUMENTATION TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "I DO NOT ACCEPT," AND DO NOT DOWNLOAD OR INSTALL THIS SOFTWARE.

NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT

This Nonexclusive Software License Agreement ("Agreement") is a contract between you, your heirs, successors and assigns ("Licensee") and Microchip Technology Incorporated, a Delaware corporation, with a principal place of business at 2355 W. Chandler Blvd., Chandler, AZ 85224-6199, and its subsidiary, Microchip Technology (Barbados) II Incorporated (collectively, "Microchip") for the accompanying Microchip software including, but not limited to, Graphics Library Software, IrDA Stack Software, MCHPFSUSB Stack Software, Memory Disk Drive File System Software, mTouch(TM) Capacitive Library Software, Smart Card Library Software, TCP/IP Stack Software, MiWi(TM) DE Software, Security Package Software, and/or any PC programs and any updates thereto (collectively, the "Software"), and accompanying documentation, including images and any other graphic resources provided by Microchip ("Documentation").

1. Definitions. As used in this Agreement, the following capitalized terms will have the meanings defined below: a. "Microchip Products" means Microchip microcontrollers and Microchip digital signal controllers.

b. "Licensee Products" means Licensee products that use or incorporate Microchip Products.

c. "Object Code" means the Software computer programming code that is in binary form (including related documentation, if any), and error corrections, improvements, modifications, and updates.

d. "Source Code" means the Software computer programming code that may be printed out or displayed in human readable form (including related programmer comments and documentation, if any), and error corrections, improvements, modifications, and updates.

e. "Third Party" means Licensee's agents, representatives, consultants, clients, customers, or contract manufacturers. f. "Third Party Products" means Third Party products that use or incorporate Microchip Products.

2. Software License Grant. Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to: a. use the Software in connection with Licensee Products and/or Third Party Products;
- b. if Source Code is provided, modify the Software; provided that Licensee clearly notifies Third Parties regarding the source of such modifications;
- c. distribute the Software to Third Parties for use in Third Party Products, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to [accept](#) (see page 174)") and this Agreement accompanies such distribution;
- d. sublicense to a Third Party to use the Software, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to [accept](#) (see page 174)");
- e. with respect to the TCP/IP Stack Software, Licensee may port the ENC28J60.c, ENC28J60.h, ENC24J600.c, and ENC24J600.h driver source files to a non-Microchip Product used in conjunction with a Microchip ethernet controller;
- f. with respect to the MiWi (TM) DE Software, Licensee may only exercise its rights when the Software is embedded on a Microchip Product and used with a Microchip radio frequency transceiver or UBEC UZ2400 radio frequency transceiver which are integrated into Licensee Products or Third Party Products.

For purposes of clarity, Licensee may NOT embed the Software on a non-Microchip Product, except as described in this Section.

3. Documentation License Grant. Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to use the Documentation in support of Licensee's authorized use of the Software

4. Third Party Requirements. Licensee acknowledges that it is Licensee's responsibility to comply with any third party license terms or requirements applicable to the use of such third party software, specifications, systems, or tools. This includes, by way of example but not as a limitation, any standards setting organizations requirements and, particularly with respect to the Security Package Software, local encryption laws and requirements. Microchip is not responsible and will not be held responsible in any manner for Licensee's failure to comply with such applicable terms or requirements.

5. Open Source Components. Notwithstanding the license grant in Section 1 above, Licensee further acknowledges that certain components of the Software may be covered by so-called "open source" software licenses ("Open Source Components"). Open Source Components means any software licenses approved as open source licenses by the Open Source Initiative or any substantially similar licenses, including without limitation any license that, as a condition of distribution of the software licensed under such license, requires that the distributor make the software available in source code format. To the extent required by the licenses covering Open Source Components, the terms of such license will apply in lieu of the terms of this Agreement. To the extent the terms of the licenses applicable to Open Source Components prohibit any of the restrictions in this Agreement with respect to such Open Source Components, such restrictions will not apply to such Open Source Component.

6. Licensee Obligations. Licensee will not: (a) engage in unauthorized use, modification, disclosure or distribution of Software or Documentation, or its derivatives; (b) use all or any portion of the Software, Documentation, or its derivatives except in conjunction with Microchip Products, Licensee Products or Third Party Products; or (c) reverse engineer (by disassembly, decompilation or otherwise) Software or any portion thereof. Licensee may not remove or alter any Microchip copyright or other proprietary rights notice posted in any portion of the Software or Documentation. Licensee will defend, indemnify and hold Microchip and its subsidiaries harmless from and against any and all claims, costs, damages, expenses (including reasonable attorney's fees), liabilities, and losses, including without limitation: (x) any claims directly or indirectly arising from or related to the use, modification, disclosure or distribution of the Software, Documentation, or any intellectual property rights related thereto; (y) the use, sale and distribution of Licensee Products or Third Party Products; and (z) breach of this Agreement.

7. Confidentiality. Licensee agrees that the Software (including but not limited to the Source Code, Object Code and library files) and its derivatives, Documentation and underlying inventions, algorithms, know-how and ideas relating to the Software and the Documentation are proprietary information belonging to Microchip and its licensors ("Proprietary Information"). Except as expressly and unambiguously allowed herein, Licensee will hold in confidence and not use or disclose any Proprietary Information and will similarly [bind](#) its employees and Third Party(ies) in writing. Proprietary Information will not include information that: (i) is in or enters the public domain without breach of this Agreement and through no fault of the receiving party; (ii) the receiving party was legally in possession of prior to receiving it; (iii) the receiving party can demonstrate was developed by

the receiving party independently and without use of or reference to the disclosing party's Proprietary Information; or (iv) the receiving party receives from a third party without restriction on disclosure. If Licensee is required to disclose Proprietary Information by law, court order, or government agency, Licensee will give Microchip prompt notice of such requirement in order to allow Microchip to object or limit such disclosure. Licensee agrees that the provisions of this Agreement regarding unauthorized use and nondisclosure of the Software, Documentation and related Proprietary Rights are necessary to protect the legitimate business interests of Microchip and its licensors and that monetary damage alone cannot adequately compensate Microchip or its licensors if such provisions are violated. Licensee, therefore, agrees that if Microchip alleges that Licensee or Third Party has breached or violated such provision then Microchip will have the right to injunctive relief, without the requirement for the posting of a bond, in addition to all other remedies at law or in equity.

8. Ownership of Proprietary Rights. Microchip and its licensors retain all right, title and interest in and to the Software and Documentation including, but not limited to all patent, copyright, trade secret and other intellectual property rights in the Software, Documentation, and underlying technology and all copies and derivative works thereof (by whomever produced). Licensee and Third Party use of such modifications and derivatives is limited to the license rights described in this Agreement.

9. Termination of Agreement. Without prejudice to any other rights, this Agreement terminates immediately, without notice by Microchip, upon a failure by Licensee or Third Party to comply with any provision of this Agreement. Upon termination, Licensee and Third Party will immediately stop using the Software, Documentation, and derivatives thereof, and immediately destroy all such copies.

10. Warranty Disclaimers. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. MICROCHIP AND ITS LICENSORS ASSUME NO RESPONSIBILITY FOR THE ACCURACY, RELIABILITY OR APPLICATION OF THE SOFTWARE OR DOCUMENTATION. MICROCHIP AND ITS LICENSORS DO NOT WARRANT THAT THE SOFTWARE WILL MEET REQUIREMENTS OF LICENSEE OR THIRD PARTY, BE UNINTERRUPTED OR ERROR-FREE. MICROCHIP AND ITS LICENSORS HAVE NO OBLIGATION TO CORRECT ANY DEFECTS IN THE SOFTWARE.

11. Limited Liability. IN NO EVENT WILL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER ANY LEGAL OR EQUITABLE THEORY FOR ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS. The aggregate and cumulative liability of Microchip and its licensors for damages hereunder will in no event exceed \$1000 or the amount Licensee paid Microchip for the Software and Documentation, whichever is greater. Licensee acknowledges that the foregoing limitations are reasonable and an essential part of this Agreement.

12. General. THIS AGREEMENT WILL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF ARIZONA AND THE UNITED STATES WITHOUT REGARD TO CONFLICTS OF LAWS PROVISIONS. Licensee agrees that any disputes arising out of or related to this Agreement, Software or Documentation will be brought exclusively in either the U.S. District Court for the District of Arizona, Phoenix Division, or the Superior Court of Arizona located in Maricopa County, Arizona. This Agreement will constitute the entire agreement between the parties with respect to the subject matter hereof. It will not be modified except by a written agreement signed by an authorized representative of Microchip. If any provision of this Agreement will be held by a court of competent jurisdiction to be illegal, invalid or unenforceable, that provision will be limited or eliminated to the minimum extent necessary so that this Agreement will otherwise remain in full force and effect and enforceable. No waiver of any breach of any provision of this Agreement will constitute a waiver of any prior, concurrent or subsequent breach of the same or any other provisions hereof, and no waiver will be effective unless made in writing and signed by an authorized representative of the waiving party. Licensee agrees to comply with all import and export laws and restrictions and regulations of the Department of Commerce or other United States or foreign agency or authority. The indemnities, obligations of confidentiality, and limitations on liability described herein, and any right of action for breach of this Agreement prior to termination, will survive any termination of this Agreement. Any prohibited assignment will be null and void. Use, duplication or disclosure by the United States Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause of FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in

Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Contractor/manufacturer is Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199.

If Licensee has any questions about this Agreement, please write to Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199 USA. ATTN: Marketing.

Copyright (c) 2012 Microchip Technology Inc. All rights reserved.

License Rev. No. 05-012412

5.8.1.4 Getting Help

This topic provides information for requesting support assistance with the TCP/IP Stack.

Description

The TCP/IP Stack is supported through Microchip's standard support channels. If you encounter difficulties, you may submit ticket requests at <http://support.microchip.com>.

The Microchip forums are also an excellent source of information, with a very lively community dedicated specifically to Ethernet and TCP/IP discussions at <http://forum.microchip.com>.

Microchip also offers embedded network classes through Regional Training Centers. For more information, visit <http://www.microchip.com/rtc>.

5.8.1.5 Utilities

This section discusses the computer software applications included with the TCP/IP Stack.

These tools are implemented using the C# or Java programming languages, or both. The C# tools (*.exe) will require the Microsoft® .NET Framework v2.0 to be installed on the local PC. The Java tools (*.jar) require Java Runtime Environment (JRE) 1.6 or later to be installed on the target computer.

5.8.1.5.1 MPFS2 Utility

This topic provides a description of the MPFS2 Utility.

Description

The MPFS2 Utility packages web pages into a format for efficient storage in an embedded system. It is a graphical application for PCs that can generate MPFS2 images for storage in external storage or internal Flash program memory.

When used to build MPFS2 images, the MPFS2 Utility also indexes the dynamic variables found. It uses this information to generate HTTPPrint.h, which ensures that the proper callback functions are invoked as necessary. It also stores this index information along with the file in the MPFS2 image, which alleviates the task of searching from the embedded device.

Finally, when developing an application that uses external storage, the MPFS2 Utility can upload images to the external storage device using the upload functionality built into the HTTP2 web server or FTP server.

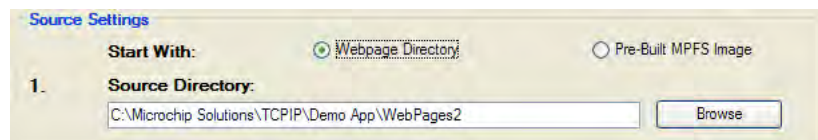
The source code for this application is included in the Microchip Applications Libraries installer.

5.8.1.5.1.1 Building MPFS2 Images

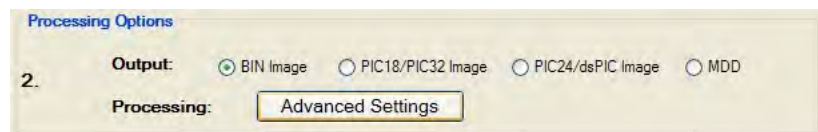
This topic provides information for building MPFS2 images.

Description

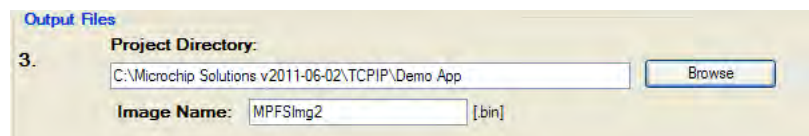
The MPFS2 Utility has four steps, which are denoted on the left hand side of the dialog. To build an MPFS image, select **Start With: Webpage Directory** in step 1 and choose the directory in which the web pages are stored.



Step 2 selects the output format. If storing the web pages in external EEPROM or serial Flash, choose the **BIN Image** output format. If internal program memory will be used, select **C18/C32 Image** for use with 8-bit and 32-bit parts, or **ASM30 Array** for 16-bit targets. To store the web pages on a device formatted with the FAT file system without compressing them into an MPFS image, select **MDD**.

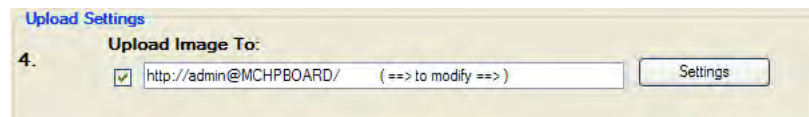


Step 3 asks for the MPLAB IDE project directory. The MPFS tool will write the image file to the project directory, and will also update the HTTPPrint.h file there if needed. Select the correct directory so that the right files are modified.



Step 4 controls the upload settings. When external EEPROM or serial flash is used for storage, the option to upload the newly created image to the board is available. Check the box next to **Upload Image To** to enable this feature. The target host name (or IP address), upload protocol, and upload path may need to be changed to the one chosen when the board was first configured. You may also need to modify the user name and password used to access the secured functionality in your application, like web page upload. Use the **Settings** button to edit these values.

If internal program memory is being used, the image will be compiled in with the project and so direct uploads are not available. Make sure to include the output source file indicated in step 3 as part of the project.



Once all the correct settings have been chosen, click the **Generate** button to create the image. If uploads are enabled, this will also attempt to upload the file to the device.

5.8.1.5.1.2 Uploading Pre-built MPFS2 Images

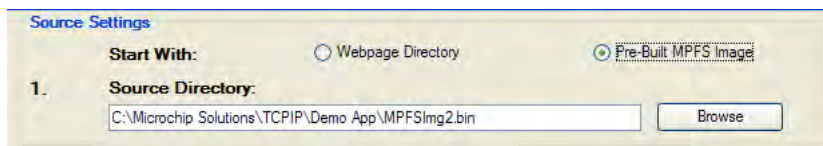
This topic provides information on uploading pre-built MPFS2 images.

Description

There are two ways to upload a pre-built image to external storage. The first is described in the Getting Started (see page 79) section, and involves uploading from the browser directly. The second is to use the MPFS2 Utility to upload the image. You can select HTTP or FTP uploading to match the protocol that your application uses.

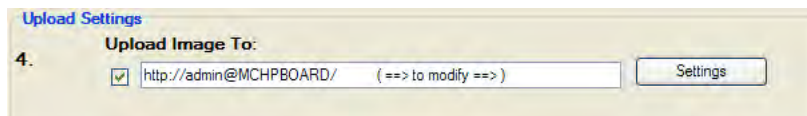
To use the MPFS2 Utility to upload an image, begin by selecting Start With: Pre-Build MPFS Image in step 1 at the top.

Choose the image file to upload.



Steps 2 and 3 are not required for pre-built images. Proceed directly to step 4 and verify that the upload settings are correct. The target host name (or IP address), upload protocol, and upload path may need to be changed to the one chosen when the board was first configured. You may also need to modify the user name and password used to access the secured functionality in your application, like web page upload. Use the Settings button to edit these values.

Once all the settings are correct, click the Upload button. The image will be uploaded to the board.

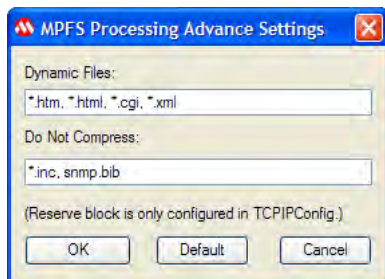


5.8.1.5.1.3 Advanced MPFS2 Settings

This topic provides information on advanced MPFS2 settings.

Description

The Advanced Settings dialog found in step 2 provides greater control over how files are processed.



The **Dynamic Files** list indicates which file types to parse for dynamic variables. By default, all files with the extensions htm, html, cgi, or xml are parsed. If an application has dynamic variables in other file types, these types must be added to the list. This field must be a comma-separated list of extensions and file names.

The **Do Not Compress** field indicates which file types should never be compressed. Compressing files with GZIP saves both storage space and transmission time. However, this is only suitable for static content such as CSS or JavaScript. Any files with dynamic variables will automatically be excluded. In addition, any file that the PIC may need to process internally should be excluded. Files included via ~inc:filename~ should not be compressed, nor should any BIB file used for the SNMP module (if present). Additional file types can be added to this list if a custom application will be accessing the MPFS.

The GZIP compressor will attempt to shrink all files. In some cases, especially with images, little or no compression is achieved. When this occurs the file is stored as-is in the MPFS image.

5.8.1.5.1.4 MPFS2 Command Line Options

This topic provides a description of the MPFS2 command line options.

Description

To facilitate batch files and automation, the MPFS2 Utility also supports execution from the command line. The syntax is as follows:

```
MPFS2.jar [options] <SourceDir> <ProjectDir> <OutputFile>
```


The SourceDir, ProjectDir, and OutputFile options are required and should be enclosed in quotation marks. The OutputFile option will be relative to ProjectDir, and cannot be a full path name. The various option switches are described in the following table.

Switch	Short	Description
/BIN	/b	Output a BIN image (Default)
/C18_C32	/c	Output a C18 or XC32 image
/ASM16	/s	Output an ASM16 image
/mpfs2	/2	Use the MPFS2 format (Default)
/html "..."	/h "..."	File types to be parsed for dynamic variables (Default: "*.htm, *.html, *.cgi, *.xml")
/xgzip "..."	/z "..."	File types to be excluded from GZIP compression (Default: "*.bib, *.inc")

The command-line interface does not support image uploads. For batch or production uploads, use a tool such as wget to upload the generated BIN image.

5.8.1.5.2 TCP/IP Discoverer

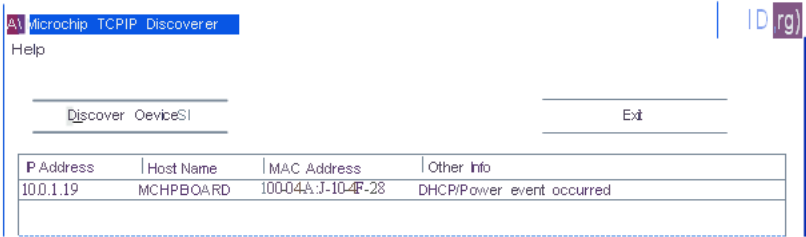
This topic provides a description of the TCP/IP Discoverer.

Description

The TCP/IP Discoverer PC project (formerly known as the Embedded Ethernet Device Discoverer) will aid in embedded product device discovery (with the Announce (see page 160) protocol) and will demonstrate how to write PC applications to communicate to embedded devices.

When the "Discover Devices" button is clicked, this application will transmit a broadcast UDP packet containing the message, "Discovery: Who is out there?" on the local network to port 30303. If any embedded devices with the Announce (see page 160) protocol enabled are connected to the network, they will respond with a UDP packet containing their host name (NBNS (see page 295)) and MAC address.

The Java source code for this application is also included. This source code should provide a rough idea of how to write a PC-based application to communicate with your embedded devices.



5.8.2 TCP/IP Stack Porting Guide

5.8.2.1 Introduction

Porting to the MPLAB Harmony TCP/IP Stack

This help file provides information for porting from a previous version of the TCP/IP Stack to the TCP/IP Stack within MPLAB Harmony.

5.8.2.2 Release Notes

MPLAB Harmony Version: v0.70b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.2.3 SW License Agreement

© 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS." MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.2.4 Upgrading from the V5 TCP/IP Stack to the MPLAB Harmony TCP/IP Stack

The MPLAB Harmony TCP/IP Stack is a completely new distribution, which is included as part of the MPLAB Harmony installation. No files need to be maintained from an existing V5 TCP/IP Stack installation.

5.8.2.4.1 MPLAB Harmony TCP/IP Stack Key Features

This topic provides a description of the key features of the MPLAB Harmony TCP/IP Stack.

Description

Note: The PIC18 device family is not supported by the MPLAB Harmony TCP/IP Stack.

The following are the key features of the MPLAB Harmony TCP/IP Stack:

- Multiple interfaces
- IPV6 support
- Fully dynamic:
 - Stack initialization/deinitialization
 - Up/down interface
 - Resource management
 - Module configuration
- Improved modularity and stack layout
- Run-time configuration through the TCP/IP console
- Interrupt driven operation
- RTOS friendly, with easy RTOS integration

5.8.2.4.2 MPLAB Harmony TCP/IP Stack Structure

This topic describes the structure of the MPLAB Harmony TCP/IP Stack.

Description

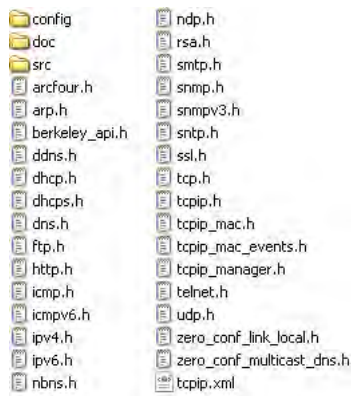
The MPLAB Harmony TCP/IP Stack consists of the following structure:

- Lowercase file names
- Underscores are used (e.g., `tcpip_manager.h`)
- Private headers are no longer exposed and have been moved to the source folder
- System services have been removed from the stack, such as:
 - Interrupts
 - Drivers
 - Timer services
 - File system
 - Board Support Package (BSP)

By default, the TCP/IP Stack is placed into the following location during installation of MPLAB Harmony on a Windows[®] system:

C:\Microchip\harmony\<version>\framework\tcpip

The following figure shows the list of files in the tcpip folder.



5.8.2.4.3 MPLAB Harmony TCP/IP Stack Design

This topic discusses the design of the MPLAB Harmony TCP/IP Stack.

Description

The MPLAB Harmony TCP/IP Stack is designed as a part of a system running other applications, middleware, etc. Therefore, it makes use of the system services that are available to other modules in the system or to the application, such as:

- File system
- System interrupts
- System driver services
- System timers
- System device drivers
- System command processor
- System console
- System debug services

Refer to the `./framework/tcpip/src/system` folder for the header files that expose the system wide available API.

5.8.2.4.4 MPLAB Harmony TCP/IP Stack API Changes

This topic discusses the reasons behind the API changes to the MPLAB Harmony TCP/IP Stack.

Description

For most modules, the API should be backward compatible to the V5 TCP/IP Stack.

The changes in the stack API have been minimized so that the porting process is straightforward. However, new functionality has been added because of new features, such as IPV6 support, multiple network interfaces, and the dynamic configuration of the stack. Other than these new features, the changes to the API were only made when an existing V5 TCP/IP Stack function did not support the required parameters/flexibility, or they were confusing.

The API changes are at the TCP, UDP, ARP, HTTP, SMTP, and stack access and initialization level.

5.8.2.4.4.1 TCP Changes

This topic discusses the changes made to the [tcp.h](#) file.

Description

TCP Changes

The TCP/IP Stack include header file, [tcp.h](#), has been updated as follows.

- *DNS Resolution* - DNS resolution is no longer performed automatically when opening a TCP [socket](#). The TCP layer takes only an IP address as an input parameter. If you need a DNS resolution, you have to perform it before opening a [socket](#) (see [dns.h](#)).
- *IPv6 Support* - Support for IPV6 has been added

Opening TCP Sockets

To open TCP sockets, the required functions are now:

[TCP_SOCKET](#) [TCPIP_TCP_ServerOpen](#)(IP_ADDRESS_TYPE addType, [TCP_PORT](#) localPort, IP_MULTI_ADDRESS* localAddress)

and

[TCP_SOCKET](#) [TCPIP_TCP_ClientOpen](#)(IP_ADDRESS_TYPE addType, [TCP_PORT](#) remotePort, IP_MULTI_ADDRESS* remoteAddress)

These two new functions replace the V5 TCP/IP Stack TCPOpen function. The application code must replace the calls to TCPOpen with [TCP_SOCKET](#) [TCPIP_TCP_ServerOpen](#) or [TCP_SOCKET](#) [TCPIP_TCP_ClientOpen](#), as appropriate.

The new calls add the possibility to use IPV6 addresses and make clear the choice of parameters for server or client sockets. The parameters are:

- typedef union


```
{
IPV4_ADDR v4Add;
IPV6_ADDR v6Add;
}IP_MULTI_ADDRESS;
```
- typedef enum


```
{
IP_ADDRESS_TYPE_ANY = 0, // either IPv4 or IPv6; unspecified;
IP_ADDRESS_TYPE_IPV4 = 1, // IPv4 address type
IP_ADDRESS_TYPE_IPV6 // IPv6 address type
}IP_ADDRESS_TYPE;
```

Note: Currently, IP_ADDRESS_TYPE_ANY is only supported for server sockets.

Disconnecting a Socket

The functions to disconnect a [socket](#) have changed to include added functionality:

- [TCPIP_TCP_Disconnect](#)([TCP_SOCKET](#) hTCP)

This function closes the TX side of a connection by sending a FIN (if currently connected) to the remote node of the connection. It no longer sends a RST signal to the remote node so that a sequence of two sequential calls to the function is no longer needed.

If the [socket](#) has the Linger option set (default), the queued TX data transmission will be attempted before sending the FIN. If the Linger option is off, the queued TX data will be discarded.

No more data can be sent by this [socket](#), but more data can be received (the [socket](#) will be eventually closed when a FIN is received from the remote node or by a time-out dictated by the [TCP_FIN_WAIT_2_TIMEOUT](#) value in [tcp_config.h](#)).

Please note that this call may fail in which case it can be reissued.

Setting the [socket](#) options is done using the function [TCPIP_TCP_OptionsSet\(\)](#).

- [TCPIP_TCP_Abort](#)([TCP_SOCKET](#) hTCP, bool killSocket)

This function aborts a connection to a remote node by sending a RST (if currently connected). Any pending TX/RX data is discarded.

A client [socket](#) will always be closed and the associated resources released. The [socket](#) cannot be used again after this call.

A server [socket](#) will abort the current connection if:

- killSocket == false (the [socket](#) will remain listening)
- killSocket == true (the [socket](#) will be closed and all associated resources released. The [socket](#) cannot be used again after this call.)
- [TCPIP_TCP_Close](#)([TCP_SOCKET](#) hTCP)

This function disconnects an open [socket](#) and destroys the [socket](#) handle, releasing the associated resources.

If the graceful option is set for the [socket](#) (default) a [TCPIP_TCP_Disconnect](#) will be tried (FIN will be sent):

- If the linger option is set (default) the [TCPIP_TCP_Disconnect](#) will attempt to [send](#) any queued TX data before issuing the FIN
- If the FIN [send](#) operation fails or the [socket](#) is not connected, an abort is generated

If the graceful option is not set, or the previous step could not [send](#) the FIN:

- A [TCPIP_TCP_Abort\(\)](#) is called, sending a RST to the remote node and communication is closed. The [socket](#) is no longer valid and the associated resources are freed.

Setting the [socket](#) options is done using the call [TCPIP_TCP_OptionsSet\(\)](#);

Adjusting Socket Size

The function to adjust the size of a [socket](#)'s RX and TX buffers has changed to include added functionality:

- [TCPIP_TCP_FifoSizeAdjust](#)([TCP_SOCKET](#) hTCP, uint16_t wMinRXSize, uint16_t wMinTXSize, [TCP_ADJUST_FLAGS](#) vFlags)

The TX and RX FIFOs (buffers) associated with a [socket](#) are now completely separate and independent.

Two new flags, [TCP_ADJUST_TX_ONLY](#) and [TCP_ADJUST_RX_ONLY](#), have been added, which allow changing the size of TX and RX buffers independently. This is the preferred option.

However, when either flag is not set, for the purpose of this function, the TX and RX FIFOs are considered to be contiguous so that the total FIFO space is divided between the TX and RX FIFOs. This provides backward compatibility with previous versions of this function.

Note: The TX or RX associated buffer sizes can be independently changed too using the [socket](#) options. See the [TCPIP_TCP_OptionsSet](#) help for more information.

5.8.2.4.4.2 UDP Changes

This topic discusses the changes made to the [udp.h](#) file.

Description

UDP Changes

The TCP/IP Stack include header file, [udp.h](#), has been updated as follows.

- *DNS Resolution* - DNS resolution is no longer performed automatically when opening a UDP [socket](#). The UDP layer takes only an IP address as an input parameter. If you need a DNS resolution, you have to perform it before opening a [socket](#) (see [dns.h](#)).
- *IPv6 Support* - Support for IPV6 has been added

Opening UDP Sockets

To open UDP sockets, the required functions are now:

```
UDP\_SOCKET TCPIP\_UDP\_ServerOpen(IP_ADDRESS_TYPE addType, UDP\_PORT localPort, IP_MULTI_ADDRESS* localAddress);
```

and

```
UDP\_SOCKET TCPIP\_UDP\_ClientOpen(IP_ADDRESS_TYPE addType, UDP\_PORT remotePort, IP_MULTI_ADDRESS* remoteAddress);
```

These two new functions replace the V5 TCP/IP Stack UDPOpen function.

The application code must replace any calls to UDPOpen with [UDP_SOCKET TCPIP_UDP_ServerOpen\(\)](#) or [UDP_SOCKET TCPIP_UDP_ClientOpen\(\)](#), as appropriate.

Note: Currently, IP_ADDRESS_TYPE_ANY is only supported for server sockets.

5.8.2.4.4.3 ARP Changes

This topic discusses the changes made to the [arp.h](#) file.

Description

ARP Changes

The header file, [arp.h](#), has been updated as follows.

The ARP module has been redesigned to support multiple network interfaces and to implement internal storage (caches) per interface for eliminating the need for frequent access to the network.

Some of the most important changes include:

- Manipulation/control of the cache entries ([TCPIP_ARP_EntryGet](#), [TCPIP_ARP_EntrySet](#), [TCPIP_ARP_EntryRemove](#)) with permanent entries support
- Dynamic notification mechanism ([TCPIP_ARP_HandlerRegister](#), [TCPIP_ARP_HandlerDeRegister](#)) for signaling of the ARP events
- Resolution calls ([TCPIP_ARP_Resolve](#), [TCPIP_ARP_Probe](#), [TCPIP_ARP_IsResolved](#)).

Normally, the application does not need access to the ARP module. The address resolution is performed internally by the stack.

The ARP module cache manipulation access is meant for TCP/IP Stack control and configuration depending on the actual network topology.

5.8.2.4.4.4 HTTP Changes

This topic discusses the changes made to the [http.h](#) file.

Note: This file was named http2.h in the V6 Beta TCP/IP Stack.

Description

HTTP Changes

The TCP/IP Stack include header file, [http.h](#), has been updated as follows.

All of the functions that the HTTP exposes as its API now take a first parameter, a HTTP connection handle, named [HTTP_CONN_HANDLE](#). This allows a cleaner and better behavior in both multi-threaded environments and in the situation where we it may be desired to run multiple instances of the HTTP process itself, allowing for serving multiple independent connections.

For example, the HTTP functions now appear as:

- [TCPIP_HTTP_FileInclude](#)([HTTP_CONN_HANDLE](#) connHandle, const uint8_t* cFile) ;
- [HTTPExecuteGet](#)([HTTP_CONN_HANDLE](#) connHandle);
- [HTTPExecutePost](#)([HTTP_CONN_HANDLE](#) connHandle);
- [TCPIP_HTTP_CurrentConnectionFileGet](#)([HTTP_CONN_HANDLE](#) connHandle);
- [TCPIP_HTTP_CurrentConnectionCallbackPosGet](#)([HTTP_CONN_HANDLE](#) connHandle);

Support has been added for the HTTP module client to store/retrieve its own connection related data. For example:

- [TCPIP_HTTP_CurrentConnectionUserDataSet](#)([HTTP_CONN_HANDLE](#) connHandle, const void* uData);
- [TCPIP_HTTP_CurrentConnectionUserDataGet](#)([HTTP_CONN_HANDLE](#) connHandle);

HTTP Print

Also, the functions exposed in [http_print.h](#) take the connection handle parameter. For example:

- [HTTPPrint](#)([HTTP_CONN_HANDLE](#) connHandle, uint32_t callbackID);
- [HTTPPrint_hellomsg](#)([HTTP_CONN_HANDLE](#) connHandle);

These changes affect all of the function calls in the file, [custom_http_app.c](#), such as:

- [HTTPExecuteGet](#)([HTTP_CONN_HANDLE](#) connHandle);
- [HTTPExecutePost](#)([HTTP_CONN_HANDLE](#) connHandle);

To port an existing application, the extra parameter will have to be added. The connection handle is passed to the application as part of the HTTP callback. The modifications should be minimal and possible using an editor without any other impact.

Please note that the MPFS2 generator tool ([mpfs2.jar](#)) has been updated to support the new HTTP API.

See the list of the complete API changes in [http.h](#) and [http_print.h](#).

5.8.2.4.4.5 MPLAB Harmony TCP/IP Stack Storage Changes

This topic describes the storage changes in the MPLAB Harmony TCP/IP Stack.

Description

MPLAB Harmony TCP/IP Stack Storage Changes

The V5 TCP/IP Stack include header file, [tcpip_storage.h](#), *does not* exist in the MPLAB Harmony TCP/IP Stack.

The TCP/IP Stack storage has become obsolete and is no longer maintained. This is mainly due to the way the dynamic initialization of the stack is done in the V5 TCP/IP Stack:

- Each module has its own initialization data
- There are many parameters that could be relevant for an application and that may require storage besides the IP address, IP Mask, or SSID, etc.
- The data is passed dynamically at the stack initialization. There is no more “build time” set of parameters against which to

check at stack configuration (although support for a default configuration set at build time is present). The actual data contents are the application responsibility.

- A system database service will be added that will use the File System. The database service will maintain configurations for all the modules in the system, including the TCP/IP. The system database service API will be available to the applications as well for storing/retrieving proprietary information.
- The parameter, `TCPIP_STACK_USE_STORAGE`, in the file, `tcPIP_config.h`, should not be enabled. Future plans call for this service to be removed from the distribution.

5.8.2.4.4.6 MPLAB Harmony TCP/IP Stack Configuration Changes

This topic describes the configuration changes in the MPLAB Harmony TCP/IP Stack.

Description

The stack configuration has a completely *new* structure. Each project has its own configuration folder that stores multiple configurations based on CPU platforms and hardware boards.

Note: V5 TCP/IP Stack configuration files *cannot* be reused in the MPLAB Harmony TCP/IP Stack.

The following is a list of the most important features:

- `system_config` folder for projects – all profiles are stored in this location
- Multiple profiles per project, which are separated by CPU platform. The TCP/IP profile is usually common.
- `tcPIP_config.h` – selects modules that are part of the build
- Each module has its own profile: [arp_config.h](#), [tcp_config.h](#), [ssl_config.h](#), etc.
- BSP profiles per development board containing:
 - `hardware_config.h` – the BSP specific hardware settings
 - `media_storage_config.h` – storage media and partitioning settings
 - `sys_fs_config.h` – file system configuration and settings
 - `mpfs_config.h` – MPFS file system settings
 - `network_config.h` – configuration for every network interface: NetBIOS names, default IP addresses, etc.
 - `system_config.h` – system configuration (i.e, system tick, system console selection, system debug services, etc.)

5.8.2.4.4.6.1 MPLAB Harmony TCP/IP Stack Configuration

This topic provides information on the configuration of the MPLAB Harmony TCP/IP Stack.

Description

MPLAB Harmony TCP/IP Stack Configuration

The presented structure is simply a model and a different layout can be chosen. The only requirement is to use the proper path in your project so that the necessary configuration files are found at build time.

For example, depending on your exact hardware platform or development board, a path similar to the following can be added to your project path:

```
..\harmony\<version>\apps\tcpip\tcpip_web_server_demo_app\firmware\src\system_config\pic32_eth_sk_int_dyn\tcpip_profile
```

5.8.2.4.4.6.2 MPLAB Harmony TCP/IP Stack Heap Configuration

This topic provides information on the heap configuration of the MPLAB Harmony TCP/IP Stack.

Description

MPLAB Harmony TCP/IP Stack Heap Configuration

The MPLAB Harmony TCP/IP Stack uses dynamic memory for both initialization and run time buffers. Therefore, there are two requirements for a project containing the stack to work properly.

The first requirement is that the TCP/IP stack should be configured with enough heap space. The amount of heap used by the stack is specified by the `TCPIP_STACK_DRAM_SIZE` parameter in the `tcip_config.h` file.

The value of the required TCP/IP heap for a project is application dependent. Some of the most important factors that impact the heap size are:

- Number of TCP sockets
 - By default, each TCP [socket](#) requires 512 bytes for a RX buffer and 512 bytes for a TX buffer. These parameters can be adjusted by modifying the values specified in [tcp_config.h](#) or dynamically through the [TCPIP_TCP_OptionsSet\(\)](#) function.
- Number of UDP sockets:
 - For each UDP [socket](#) that needs to initiate a transmission, the IP layer will have to allocate the required space (suggested by the functions, `UDPv4IsTxPutReady` or `UDPv4IsTxPutReady`)
 - Once the UDP buffering will be added, each [socket](#) will have its own RX and TX buffers. These parameters can be adjusted by modifying the values specified in the file, [udp_config.h](#), or dynamically through the [TCPIP_UDP_OptionsSet\(\)](#) function
- The type of Ethernet MAC that is used:
 - The PIC32MX6XX/7XX devices with a built-in 100 Mbps Ethernet controller use system memory for buffering incoming RX traffic. For sustained 100 Mbps operation, adequate RX buffer space must be provided. This parameter can be adjusted by modifying the values specified in the file, [network_config.h](#), using the array, `TCPIP_HOSTS_CONFIGURATION[]`.
- If SSL is enabled, it will require additional buffering for encryption/decryption at run-time. The SSL module buffering requirement is dependent on the RSA key length specified in [ssl_config.h](#).

The second requirement is that the project that includes the MPLAB Harmony TCP/IP Stack must be built with sufficient heap size to accommodate the stack (at a minimum, the project needs the `TCPIP_STACK_DRAM_SIZE` bytes of heap). This parameter is adjusted on the Linker tab in the project properties.

In general, for a TCP/IP project running on a PIC32MX device with an embedded Ethernet controller, at least 8 KB of heap space is needed by the stack. However, the following is implied:

- 100 Mbps traffic is not sustained
- No SSL
- Relatively few sockets

A typical value for comfortably handling 100 Mbps Ethernet traffic would be 40 KB of TCP/IP heap space.

The amount of the required heap is less for an external Ethernet MAC.

Keep in mind the following when assigning heap space:

- If there is not enough heap space the stack initialization may fail
- If there is not enough heap space some run time buffer allocation may fail and some packets transmission will have to be deferred until a later time, thus impacting the stack performance
- It is always a good idea to have a reserve, at least an extra 2 KB of heap space above the total amount of space that is used
- A very useful tool in understanding the heap allocation and how the space is distributed among the stack modules is the TCP/IP command processor

By enabling the parameter, `TCPIP_STACK_DRAM_DEBUG_ENABLE`, in the `tcip_config.h` file, the stack will output debug messages when it runs out of memory. Then, using the `heapinfo` command at the TCP/IP command processor prompt will

return a snapshot of the current TCP/IP heap status and can help in early detection of problems.

Optionally, enabling the parameter, `TCPIP_STACK_DRAM_TRACE_ENABLE` in the `tcip_config.h` file, will instruct the TCP/IP heap allocation module to store trace information that will be displayed with the `heapinfo` command.

5.8.2.4.4.7 New MPLAB Harmony TCP/IP Stack API Functions

This topic describes some of the important new API functions in the MPLAB Harmony TCP/IP Stack.

Description

There are many new functions available that have been introduced to take advantage of new features, such as IPV6 and support of multiple interfaces. However, there is no concern for the porting process regarding the new API calls as the previous stack implementation did not support this kind of service.

A new API function should be added to the application only when the access to the new feature is needed.

Existent applications should port easily without using the new API.

For reference, the following are a few of the most important new API functions that could prove useful in the porting of your application:

- Initialization of the stack network interfaces:
 - [`TCPIP\_STACK\_NetUp\(\)`](#)
 - [`TCPIP\_STACK\_NetDown\(\)`](#)
- Default interface selection:
 - [`TCPIP\_STACK\_NetDefaultGet\(\)`](#)
 - [`TCPIP\_STACK\_NetDefaultSet\(\)`](#)
- TCP/UDP multiple interface support:
 - [`TCPIP\_TCP\_Bind\(\)`](#)
 - [`TCPIP\_TCP\_RemoteBind\(\)`](#)
 - [`TCPIP\_TCP\_SocketNetGet\(\)`](#)
 - [`TCPIP\_TCP\_SocketNetSet\(\)`](#)
 - [`TCPIP\_UDPBind\(\)`](#)
 - [`TCPIP\_UDP\_RemoteBind\(\)`](#)
 - [`TCPIP\_UDP\_SocketNetGet\(\)`](#)
 - [`TCPIP\_UDP\_SocketNetSet\(\)`](#)

Refer to the file, [`tcip\_manager.h`](#), for a complete list of all new network interface APIs.

5.8.2.4.4.7.1 Main Program Changes

This topic describes the main program changes in the MPLAB Harmony TCP/IP Stack.

Description

Main Program Changes

There are few changes that the main program loop has to take care of. A full working example is given with the file, `main_demo.c`, which is provided with the stack distribution.

The main program should call the following functions:

- [`SYS\_Initialize\(\)`](#) – This function is what initializes the system and runs the Board Support Package (BSP) specific code

- [SYS_Tasks\(\)](#) – This is the function that takes care of the system tasks and must be called periodically
- [TCPIP_STACK_Task\(\)](#) – This is the TCP/IP stack tasks function that runs all of the state machines that are part of the TCP/IP stack. This function must be called periodically. Please note the name change.

To gain access to a network interface the `TCPIP_STACK_NetHandle("iname")` function should be used. Examples are provided in the previous section.

5.8.2.4.4.8 MPLAB Harmony TCP/IP Stack Initialization Changes

This topic describes the initialization updates to the MPLAB Harmony TCP/IP Stack.

Description

The TCP/IP Stack include header file, [tcpip_manager.h](#), has been updated as follows.

To initialize the stack the call is now:

```
TCPIP\_STACK\_Initialize(const          TCPIP_NETWORK_CONFIG*          pNetConf,          int          nNets,
TCPIP_STACK_MODULE_CONFIG* pModConfig, int nModules);
```

Where:

- `pNetConf` – is a pointer to an array of configurations for all the initialized network interfaces
- `nNets` – is the number of the network interfaces that the stack is to support
- `pModConfig` – is a pointer to a table storing configuration data for the TCP/IP stack modules
- `nModules` – is the number of entries in this table, (i.e., the number of initialized modules)

The default TCP/IP stack configuration is provided with the stack distribution and consists of:

- `network_config.h`: `TCPIP_HOSTS_CONFIGURATION[]` – an array of `TCPIP_NETWORK_CONFIG` entries describing the default configuration of each network interface
- `tcpip_modules_config.h`: `TCPIP_STACK_MODULE_CONFIG_TBL[]` – an array of `TCPIP_STACK_MODULE_CONFIG` entries storing the default configuration data for each module of the TCP/IP stack. This is just an aggregate of all the default configurations per module found in the module configuration header files, such as [tcp_config.h](#), [udp_config.h](#), [arp_config.h](#), and so on.

These tables are defined by default in the configuration files. See MPLAB Harmony TCP/IP Stack Configuration Changes for details.

You can change the parameters of these structures as appropriate.

An example of the TCP/IP initialization is part of the distributed file, `main_demo.c`. This file is located in the following Windows[®] path:

```
.\harmony<version>\apps\tcpip\tcpip_web_server_demo_app\firmware\src
```

Note: The stack initialization may fail. The application code must check the result of this call to detect if the stack failed to initialize properly.

5.8.2.4.4.9 MPLAB Harmony TCP/IP Stack Access Changes

This topic describes the stack access updates that were made to the MPLAB Harmony TCP/IP Stack.

Description

The TCP/IP Stack include header file, [tcpip_manager.h](#), has been updated as follows.

Direct access to the internally maintained stack structures has been removed. To access the information for a specific network interface a handle must be obtained using the interface name. For example:

```
TCPIP\_NET\_HANDLE hNet = TCPIP\_STACK\_NetHandleGet("PIC32INT");
```

or

```
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("MRF24W");
```

Once a handle to the interface has been obtained, different parameters of that network interface can be queried and set. For example:

- `TCPIP_STACK_NetAddress(TCPIP_NET_HANDLE netH)` – returns the network interface address
- `TCPIP_STACK_NetAddressGateway(TCPIP_NET_HANDLE netH)` – returns the interface gateway address

Refer to the `tcpip_manager.h` file for a complete list of functions.

Note: The `TCPIP_NET_HANDLE` is an opaque data type.

The well known names of the network interfaces are currently in the file, `tcpip.h`.

The exact header file that exposes the network interfaces names may change in the future; however, the names of the supported network interfaces will be retained.

Use the file, `network_config.h`, as an example.

Refer to the file, `main_demo.c`, for an example of how to use the interface handle functions.

5.8.2.5 MPLAB Harmony TCP/IP Stack Utilities

This topic describes the MPLAB Harmony TCP/IP Stack utilities.

Description

The MPLAB Harmony TCP/IP Stack includes the following utilities:

- MPFS2 - The MPFS2 Java utility that assists the application in generating the MPFS image of the files needed by the Web server has been updated. This utility supports both the V5 and MPLAB Harmony versions of the TCP/IP stack with improved functionality. The V5 TCP/IP Stack utility can still be used; however, some minor adjustments may need to be done manually.
- TCP/IP Discoverer - The TCP/IP Discoverer Java utility has been updated to support not only the MPLAB Harmony TCP/IP Stack, but also IPV6. This utility is up and running and is backward compatible with the V5 TCP/IP Stack.
- TCP/IP Configuration - The TCP/IP Configuration utility is currently not supported. The need of individually configuring TCP and UDP sockets is no longer needed, as all sockets behave similarly and they are allocated dynamically as needed.

5.8.2.6 MPLAB Harmony TCP/IP Stack SSL/RSA Usage

This topic provides information regarding the usage of SSL and RSA.

Description

The SSL design has been updated to support multiple interfaces. The same is true for the RSA engine. However, the SSL module does not currently use the Microchip Crypto Library. Until the proper Crypto Library support is added to SSL, the stack is distributed with the RSA and RC4 code embedded into the stack.

In the future, the Crypto Library will be completely removed from the TCP/IP stack, and the stack will be just a client to the Crypto Library.

5.8.2.7 Porting Applications from the V6 Beta TCP/IP Stack to the MPLAB Harmony TCP/IP Stack

Porting an application from the V6 Beta TCP/IP Stack into the TCP/IP Stack within MPLAB Harmony consists of updating

existing TCP/IP Stack function names in your application to the new MPLAB Harmony function names.

5.8.2.7.1 MPLAB Harmony TCP/IP Stack Function Name Compliance

This topic provides header file function mapping tables that show the V6 Beta TCP/IP Stack function name and the compliant name within the MPLAB Harmony TCP/IP Stack.

Description

The replacement names chosen to make existing TCP/IP Stack functions compliant with MPLAB Harmony are shown in individual tables per header file name.

The files listed in the tables are located in the following two MPLAB Harmony installation folders:

```
. \harmony\<version>\framework\tcpip
. \harmony\<version>\framework\tcpip\src
```

Each table shows the original and new names used. Note that other than name changes, no other modifications were made to the middleware functions.

Header File: **arp.h**

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
ARP_ENTRY_QUERY	TCPIP_ARP_ENTRY_QUERY
ARP_ENTRY_TYPE	TCPIP_ARP_ENTRY_TYPE
ARP_EVENT_HANDLER	TCPIP_ARP_EVENT_HANDLER
ARP_EVENT_TYPE	TCPIP_ARP_EVENT_TYPE
ARP_HANDLE	TCPIP_ARP_HANDLE
ARP_OPERATION_TYPE	TCPIP_ARP_OPERATION_TYPE
ARP_RESULT	TCPIP_ARP_RESULT
ARPCacheGetEntriesNo	TCPIP_ARP_CacheEntriesNoGet
ARPCacheSetThreshold	TCPIP_ARP_CacheThresholdSet
ARPEntryGet	TCPIP_ARP_EntryGet
ARPEntryQuery	TCPIP_ARP_EntryQuery
ARPEntryRemove	TCPIP_ARP_EntryRemove
ARPEntryRemoveAll	TCPIP_ARP_EntryRemoveAll
ARPEntryRemoveNet	TCPIP_ARP_EntryRemoveNet
ARPEntrySet	TCPIP_ARP_EntrySet
ARPDeRegisterHandler	TCPIP_ARP_HandlerDeRegister
ARPIsResolved	TCPIP_ARP_IsResolved
ARPProbe	TCPIP_ARP_Probe
ARPRegisterHandler	TCPIP_ARP_HandlerRegister
ARPResolve	TCPIP_ARP_Resolve
ARPRegisterCallbacks	TCPIP_ARP_CallbacksRegister

Header File: [dhcp.h](#)

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DHCPDeRegisterHandler	TCPIP_DHCP_HandlerDeRegister
DHCPRegisterHandler	TCPIP_DHCP_HandlerRegister
DHCPSetRequestTimeout	TCPIP_DHCP_RequestTimeoutSet
DHCP_EVENT_HANDLER	TCPIP_DHCP_EVENT_HANDLER
DHCP_EVENT_TYPE	TCPIP_DHCP_EVENT_TYPE
DHCP_HANDLE	TCPIP_DHCP_HANDLE
DHCPDisable	TCPIP_DHCP_Disable
DHCPEnable	TCPIP_DHCP_Enable
DHCPisBound	TCPIP_DHCP_IsBound
DHCPisEnabled	TCPIP_DHCP_IsEnabled
DHCPisServerDetected	TCPIP_DHCP_IsServerDetected

Header File: dhcps.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DHCPServerIsEnabled	TCPIP_DHCPS_IsEnabled
DHCPServerLeaseEntryGet	TCPIP_DHCPS_LeaseEntryGet
DCHPServerGetPoolEntries	TCPIP_DHCPS_GetPoolEntries
DCHPServerRemovePoolEntries	TCPIP_DHCPS_RemovePoolEntries
DHCPServerDisable	TCPIP_DHCPS_Disable
DHCPServerEnable	TCPIP_DHCPS_Enable
DHCPServerIsEnabled	TCPIP_DHCPS_IsEnabled

Header File: ddns.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DDNSForceUpdate	TCPIP_DDNS_UpdateForce
DDNSSetService	TCPIP_DDNS_ServiceSet
DDNSGetLastIP	TCPIP_DDNS_LastIPGet
DDNSGetLastStatus	TCPIP_DDNS_LastStatusGet

Header File: dns.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DNSBeginUsage	TCPIP_DNS_UsageBegin
DNSEndUsage	TCPIP_DNS_UsageEnd
DNSResolve	TCPIP_DNS_Resolve
DNSIsResolved	TCPIP_DNS_IsResolved

Header File: http.h (formerly http2.h)

V6 Beta TCP/IP Stack Name (http2.h)	MPLAB Harmony Compliant Name (http.h)
HTTPURLDecode	TCPIP_HTTP_URLDecode

HTTPGetArg	TCPIP_HTTP_ArgGet
HTTPReadPostName	TCPIP_HTTP_PostNameRead
HTTPReadPostValue	TCPIP_HTTP_PostValueRead
HTTPIncFile	TCPIP_HTTP_FileInclude
HTTPCurConnectionFileGet	TCPIP_HTTP_CurrentConnectionFileGet
HTTPCurConnectionPostSmGet	TCPIP_HTTP_CurrentConnectionPostSmGet
HTTPCurConnectionPostSmSet	TCPIP_HTTP_CurrentConnectionPostSmSet
HTTPCurConnectionDataBufferGet	TCPIP_HTTP_CurrentConnectionDataBufferGet
HTTPCurConnectionCallbackPosGet	TCPIP_HTTP_CurrentConnectionCallbackPosGet
HTTPCurConnectionCallbackPosSet	TCPIP_HTTP_CurrentConnectionCallbackPosSet
HTTPCurConnectionStatusSet	TCPIP_HTTP_CurrentConnectionStatusSet
HTTPCurConnectionHasArgsSet	TCPIP_HTTP_CurrentConnectionHasArgsSet
HTTPCurConnectionByteCountGet	TCPIP_HTTP_CurrentConnectionByteCountGet
HTTPCurConnectionByteCountSet	TCPIP_HTTP_CurrentConnectionByteCountSet
HTTPCurConnectionByteCountDec	TCPIP_HTTP_CurrentConnectionByteCountDec
HTTPCurConnectionSocketGet	TCPIP_HTTP_CurrentConnectionSocketGet

Header File: icmp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
ICMPDeRegisterCallback	TCPIP_ICMP_CallbackDeregister
ICMPRegisterCallback	TCPIP_ICMP_CallbackRegister
ICMPSEndEchoRequest	TCPIP_ICMP_EchoRequestSend

Header File: icmpv6.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_ICMPV6_DeRegisterCallback	TCPIP_ICMPV6_CallbackDeregister
TCPIP_ICMPV6_PutHeaderEchoRequest	TCPIP_ICMPV6_HeaderEchoRequestPut
TCPIP_ICMPV6_RegisterCallback	TCPIP_ICMPV6_CallbackRegister
TCPIP_IPV6_GetDestAddress	TCPIP_IPV6_AddressDestGet
TCPIP_IPV6_GetSourceAddress	TCPIP_IPV6_AddressSourceGet
TCPIP_IPV6_SetDestAddress	TCPIP_IPV6_AddressDestSet
TCPIP_IPV6_SetSourceAddress	TCPIP_IPV6_AddressSourceSet
TCPIP_IPV6_DAS_SelectSourceAddress	TCPIP_IPV6_DAS_AddressSourceSelect
TCPIP_IPV6_AddUnicastAddress	TCPIP_IPV6_AddressUnicastAdd
TCPIP_IPV6_RemoveUnicastAddress	TCPIP_IPV6_AddressUnicastRemove
TCPIP_IPV6_AddMulticastListener	TCPIP_IPV6_MulticastListenerAdd
TCPIP_IPV6_RemoveMulticastListener	TCPIP_IPV6_MulticastListenerRemove
TCPIP_IPV6_RegisterHandler	TCPIP_IPV6_HandlerRegister
TCPIP_IPV6_DeRegisterHandler	TCPIP_IPV6_HandlerDeRegister

Header File: ipv6.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_IPV6_InterfacelsReady	Not Applicable
TCPIP_IPV6_Put	Not Applicable
TCPIP_IPV6_AddressIsSolicitedNodeMulticast	Not Applicable
TCPIP_IPV6_Flush	Not Applicable
TCPIP_IPV6_PutArrayHelper	TCPIP_IPV6_ArrayPutHelper
TCPIP_IPV6_DAS_SelectSourceAddress	TCPIP_IPV6_DASSourceAddressSelect
TCPIP_IPV6_GetDestAddress	TCPIP_IPV6_DestAddressGet
TCPIP_IPV6_SetDestAddress	TCPIP_IPV6_DestAddressSet
TCPIP_IPV6_DeRegisterHandler	TCPIP_IPV6_HandlerDeregister
TCPIP_IPV6_RegisterHandler	TCPIP_IPV6_HandlerRegister
TCPIP_IPV6_AddMulticastListener	TCPIP_IPV6_MulticastListenerAdd
IPv6RemoveMulticastListener	TCPIP_IPV6_MulticastListenerRemove
TCPIP_IPV6_FreePacket	TCPIP_IPV6_PacketFree
TCPIP_IPV6_SetPayload	TCPIP_IPV6_PayloadSet
TCPIP_IPV6_GetSourceAddress	TCPIP_IPV6_SourceAddressGet
TCPIP_IPV6_SetSourceAddress	TCPIP_IPV6_SourceAddressSet
TCPIP_IPV6_IsTxPutReady	TCPIP_IPV6_TxIsPutReady
TCPIP_IPV6_AllocateTxPacket	TCPIP_IPV6_TxPacketAllocate
TCPIP_IPV6_AddUnicastAddress	TCPIP_IPV6_UnicastAddressAdd
TCPIP_IPV6_GetArray	TCPIP_IPV6_ArrayGet
IPv6RemoveUnicastAddress	TCPIP_IPV6_UnicastAddressRemove

Header File: ndp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_NDP_NeighborConfirmReachability	TCPIP_NDP_NborReachConfirm

Header File: smtp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
SMTPPutArray	TCPIP_SMTP_ArrayPut
SMTPFlush	TCPIP_SMTP_Flush
SMTPIsBusy	TCPIP_SMTP_IsBusy
SMTPIsPutReady	TCPIP_SMTP_IsPutReady
SMTPSendMail	TCPIP_SMTP_MailSend
SMTPPut	TCPIP_SMTP_Put
SMTPPutDone	TCPIP_SMTP_PutIsDone
SMTPPutString	TCPIP_SMTP_StringPut
SMTPBeginUsage	TCPIP_SMTP_UsageBegin

SMTPEndUsage	TCPIP_SMTP_UsageEnd

Header File: snmp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
SNMPNotify	TCPIP_SNMP_Notify
SNMPIsNotifyReady	TCPIP_SNMP_NotifyIsReady
SNMPNotifyPrepare	TCPIP_SNMP_NotifyPrepare
SNMPGetPktProcessingDynMemStubPtrs	TCPIP_SNMP_PacketProcStubPtrsGet
SNMPGetProcessBuffData	TCPIP_SNMP_ProcessBufferDataGet
SNMPPutDataToProcessBuff	TCPIP_SNMP_DataCopyToProcessBuffer
SNMPRetrieveReadCommunity	TCPIP_SNMP_ReadCommunityGet
SNMPGetTrapTime	TCPIP_SNMP_TrapTimeGet
SNMPUdpClientGetNet	TCPIP_SNMP_ClientGetNet
SNMPRetrieveWriteCommunity	TCPIP_SNMP_WriteCommunityGet

Header File: sntp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
SNTPGetUTCSeconds	TCPIP_SNTP_UTCSecondsGet

Header File: tcpip_manager.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_STACK_SetNetAddress	TCPIP_STACK_NetAddressSet
TCPIP_STACK_SetNetGatewayAddress	TCPIP_STACK_NetAddressGatewaySet
TCPIP_STACK_SetNetPriDNSAddress	TCPIP_STACK_NetAddressDnsPrimarySet
TCPIP_STACK_SetNetSecondDNSAddress	TCPIP_STACK_NetAddressDnsSecondSet
TCPIP_STACK_SetNetMask	TCPIP_STACK_NetMaskSet
TCPIP_STACK_SetNetMacAddress	TCPIP_STACK_NetAddressMacSet
TCPIP_STACK_NetGatewayAddress	TCPIP_STACK_NetAddressGateway
TCPIP_STACK_NetPriDNSAddress	TCPIP_STACK_NetAddressDnsPrimary
TCPIP_STACK_NetSecondDNSAddress	TCPIP_STACK_NetAddressDnsSecond
TCPIP_STACK_NetMacAddress	TCPIP_STACK_NetAddressMac
TCPIP_STACK_NetBcastAddress	TCPIP_STACK_NetAddressBcast
TCPIP_STACK_SetNetBIOSName	TCPIP_STACK_NetBiosNameSet
TCPIP_STACK_GetDefaultNet	TCPIP_STACK_NetDefaultGet
TCPIP_STACK_SetDefaultNet	TCPIP_STACK_NetDefaultSet
TCPIP_STACK_IsNetLinked	TCPIP_STACK_NetIsLinked
TCPIP_STACK_CreateNetInfo	TCPIP_STACK_NetInfoCreate
TCPIP_STACK_IsNetUp	TCPIP_STACK_NetIsUp

Header File: tcp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIsGetReady	TCPIP_TCP_GetIsReady
TCPGetArray	TCPIP_TCP_ArrayGet
TCPPeek	TCPIP_TCP_Peek
TCPPeekArray	TCPIP_TCP_ArrayPeek
TCPGetRxFIFOFree	TCPIP_TCP_FifoRxFreeGet
TCPGetTxFIFOFull	TCPIP_TCP_FifoTxFullGet
TCPDiscard	TCPIP_TCP_Discard
TCPGet	TCPIP_TCP_Get
TCPFind	TCPIP_TCP_Find
TCPFindArray	TCPIP_TCP_ArrayFind
TCPAdjustFIFOSize	TCPIP_TCP_FifoSizeAdjust
TCPOpenServer	TCPIP_TCP_ServerOpen
TCPOpenClient	TCPIP_TCP_ClientOpen
TCPBind	TCPIP_TCP_Bind
TCPRemoteBind	TCPIP_TCP_RemoteBind
TCPSetOptions	TCPIP_TCP_OptionsSet
TCPGetOptions	TCPIP_TCP_OptionsGet
TCPIsConnected	TCPIP_TCP_IsConnected
TCPWasReset	TCPIP_TCP_WasReset
TCPDisconnect	TCPIP_TCP_Disconnect
TCPClose	TCPIP_TCP_Close
TCPGetSocketInfo	TCPIP_TCP_SocketInfoGet
TCPSocketSetNet	TCPIP_TCP_SocketNetSet
TCPSocketGetNet	TCPIP_TCP_SocketNetGet
TCPIsPutReady	TCPIP_TCP_PutIsReady
TCPPut	TCPIP_TCP_Put
TCPPutArray	TCPIP_TCP_ArrayPut
TCPPutString	TCPIP_TCP_StringPut
TCPFlush	TCPIP_TCP_Flush
TCPGetRxFIFOFull	TCPIP_TCP_FifoRxFullGet
TCPGetTxFIFOFree	TCPIP_TCP_FifoTxFreeGet
TCPAbort	TCPIP_TCP_Abort
TCPAddSSLListener	TCPIP_TCPSSL_ListenerAdd
TCPIsSSL	TCPIP_TCP_SocketIsSecuredBySSL
TCPSetDestinationIPAddress	TCPIP_TCP_DestinationIPAddressSet
TCPSetSourceIPAddress	TCPIP_TCP_SourceIPAddressSet

TCPSSLIsHandshaking	TCPIP_TCPSSL_StillHandshaking
TCpSSLMessageTransmit	TCPIP_TCPSSL_MessageTransmit
TCPSSLPutRecordHeader	TCPIP_TCPSSL_RecordHeaderPut
TCPStartSSLClient	TCPIP_TCPSSL_ClientStart
TCPStartSSLClientEx	TCPIP_TCPSSL_ClientBegin
TCPStartSSLServer	TCPIP_TCPSSL_ServerStart

Header File: tcpip_events.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_STACK_GetPendingEvents	TCPIP_STACK_EventsPendingGet
TCPIP_STACK_RegisterHandler	TCPIP_STACK_HandlerRegister
TCPIP_STACK_DeRegisterHandler	TCPIP_STACK_HandlerDeregister

Header File: ipv4.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_IPV4_GetArray	TCPIP_IPV4_ArrayGet
TCPIP_IPV4_PutArray	TCPIP_IPV4_ArrayPut
TCPIP_IPV4_PutArrayHelper	TCPIP_IPV4_ArrayPutHelper
TCPIP_IPV4_GetDestAddress	TCPIP_IPV4_DestAddressGet
TCPIP_IPV4_SetDestAddress	TCPIP_IPV4_DestAddressSet
TCPIP_IPV4_FreePacket	TCPIP_IPV4_PacketFree
TCPIP_IPV4_SetPayload	TCPIP_IPV4_PayloadSet
TCPIP_IPV4_GetSourceAddress	TCPIP_IPV4_SourceAddressGet
TCPIP_IPV4_SetSourceAddress	TCPIP_IPV4_SourceAddressSet
TCPIP_IPV4_IsTxPutReady	TCPIP_IPV4_TxIsPutReady
TCPIP_IPV4_IsTxReady	TCPIP_IPV4_TxIsReady
TCPIP_IPV4_AllocateTxPacket	TCPIP_IPV4_TxPacketAllocate

Header File: udp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
UDPIsGetReady	TCPIP_UDP_GetIsReady
UDPSetRxOffset	TCPIP_UDP_RxOffsetSet
UDPGetArray	TCPIP_UDP_ArrayGet
UDPDiscard	TCPIP_UDP_Discard
UDPGet	TCPIP_UDP_Get
UDPOpenServer	TCPIP_UDP_ServerOpen
UDPOpenClient	TCPIP_UDP_ClientOpen
UDPBind	TCPIP_UDP_Bind
UDPRemoteBind	TCPIP_UDP_RemoteBind

UDPSetOptions	TCPIP_UDP_OptionsSet
UDPGetOptions	TCPIP_UDP_OptionsGet
UDPIsConnected	TCPIP_UDP_IsConnected
UDPIsOpened	TCPIP_UDP_IsOpened
UDPClose	TCPIP_UDP_Close
UDPGetSocketInfo	TCPIP_UDP_SocketInfoGet
UDPSocketSetNet	TCPIP_UDP_SocketNetSet
UDPSocketGetNet	TCPIP_UDP_SocketNetGet
UDPSetTxOffset	TCPIP_UDP_TxOffsetSet
UDPIsTxPutReady	TCPIP_UDP_TxPutIsReady
UDPPutArray	TCPIP_UDP_ArrayPut
UDPPutString	TCPIP_UDP_StringPut
UDPGetTxCount	TCPIP_UDP_TxCountGet
UDPFlush	TCPIP_UDP_Flush
UDPPut	TCPIP_UDP_Put
UDPSetBcastIPv4Address	TCPIP_UDP_BcastIPv4AddressSet
UDPSetSourceIPAddress	TCPIP_UDP_SourceIPAddressSet
UDPSetDestinationIPAddress	TCPIP_UDP_DestinationIPAddressSet

5.8.2.7.2 Development Information (Advance Information)

This *advance information* is specifically provided to assist MPLAB Harmony driver development for porting the existing Ethernet Controller Library and MAC/PHY drivers to the new platform.

5.8.2.7.2.1 Porting the MPLAB Harmony Drivers and Peripheral Library

Note: This information is considered to be *Advance Information*.

This topic provides function mapping tables that show the V6 Beta TCP/IP Stack function name and the compliant name within the MPLAB Harmony TCP/IP Stack for the purpose of porting to the MPLAB Harmony Ethernet Drivers and Ethernet Peripheral Library.

Description

Note: This information is considered to be *Advance Information*.

Porting the MPLAB Harmony Ethernet Drivers and the MPLAB Harmony Ethernet Peripheral Library involves replacing the Ethernet Controller Library with calls to MPLAB Harmony equivalents. This information is listed in Table 1. Please note that function arguments were modified in some instances.

To understand how a function of the Ethernet Controller Library is different from its equivalent MPLAB Harmony function, the header of the MPLAB Harmony function should be examined. The MPLAB Harmony function header always lists the Ethernet Controller Library function(s) that the new MPLAB Harmony function replaces, including the arguments of the old function.

Table 1: TCP/IP Stack Function Mapping

Ethernet Controller Peripheral Library Name	MPLAB Harmony Name
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthEventsClr	PLIB_ETH_EventsClr
EthEventsEnableClr	PLIB_ETH_EventsEnableClr
EthEventsEnableGet	PLIB_ETH_EventsEnableGet
EthEventsEnableSet	PLIB_ETH_EventsEnableSet
EthEventsEnableWrite	PLIB_ETH_EventsEnableWrite
EthEventsGet	PLIB_ETH_EventsGet
EthInit	DRV_ETHMAC_LegacyInit
EthMACGetAddress	PLIB_ETH_MACGetAddress
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthMACSetAddress	PLIB_ETH_MACSetAddress
EthMACSetMaxFrame	PLIB_ETH_MACSetMaxFrame
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxFiltersClr	PLIB_ETH_RxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_RxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_RxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_RxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_RxFiltersSet
EthRxFiltersWrite	PLIB_ETH_RxFiltersWrite
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthRxSetBufferSize	PLIB_ETH_RxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_StatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_StatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_StatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_StatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_StatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_StatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_StatTxSColCnt
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

Table 2 shows the MPLAB Harmony Ethernet Controller Peripheral Library equivalents for functions in the MPLAB Harmony Ethernet MAC Driver and the MPLAB Harmony Ethernet PHY Driver.

The Ethernet Controller Library files are located in the following Windows® installation path:

C:\Program Files\Microchip\xc32\<version>\pic32-libs\peripheral\eth\source

Table 2: Ethernet Controller Peripheral Library Function Mapping

Ethernet Controller Peripheral Library Name	MPLAB Harmony Ethernet MAC Driver Name
EthClose	DRV_ETHMAC_LegacyClose
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthInit	DRV_ETHMAC_LegacyInit
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

Ethernet Controller Peripheral Library Name	MPLAB Harmony Ethernet Peripheral Library Name
EthEventsEnableClr	PLIB_ETH_EventsEnableClr
EthEventsEnableGet	PLIB_ETH_EventsEnableGet
EthEventsEnableSet	PLIB_ETH_EventsEnableSet
EthEventsEnableWrite	PLIB_ETH_EventsEnableWrite
EthEventsGet	PLIB_ETH_EventsGet
EthMACGetAddress	PLIB_ETH_MACGetAddress
EthMACSetAddress	PLIB_ETH_MACSetAddress
EthMACSetMaxFrame	PLIB_ETH_MACSetMaxFrame
EthRxFiltersClr	PLIB_ETH_RxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_RxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_RxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_RxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_RxFiltersSet
EthRxFiltersWrite	PLIB_ETH_RxFiltersWrite
EthRxSetBufferSize	PLIB_ETH_RxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_StatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_StatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_StatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_StatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_StatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_StatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_StatTxSColCnt

Table 3 shows the existing and new function names for the TCP/IP MAC driver.

Please note that the following v6 Beta TCP/IP Stack files are now obsolete:

- eth_pic32_ext_phy.c
- eth_pic32_ext_phy_*.c
- eth_pic32_int_mac.c
- mac_events_pic32.c

Table 3: TCP/IP MAC Driver (tcpip_mac.h) – V6 Beta TCP/IP Stack to MPLAB Harmony TCP/IP Stack Conversion

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
MAC_ADDR	TCPIP_MAC_ADDR
MAC_ETHERNET_HEADER	TCPIP_MAC_ETHERNET_HEADER
MAC_NOTIFY_HANDLER	TCPIP_MAC_NOTIFY_HANDLER
MAC_PACKET	TCPIP_MAC_PACKET
MAC_RX_PKT_STAT	TCPIP_MAC_RX_PKT_STAT
MACEventAck	TCPIP_MAC_EventAck
MACEventClearNotifyEvents	TCPIP_MAC_EventNotifyClear
MACEventGetPending	TCPIP_MAC_EventPendingEventsGet
MACEventSetNotifyEvents	TCPIP_MAC_EventNotifyEventsSet
MACEventSetNotifyHandler	TCPIP_MAC_EventNotifyHandlerSet
MACFCEnable	TCPIP_MAC_FlowControlEnable
MACFCSetPauseValue	TCPIP_MAC_FlowControlPauseValueSet
MACFCSetRxWMark	TCPIP_MAC_FlowControlRxWMarkSet
TCPIP_MAC_FC_TYPE	TCPIP_MAC_FLOWCONTROLTYPE
MACCheckLink	TCPIP_MAC_LinkCheck
MACCalcIPBufferChecksum	TCPIP_MAC_ChecksumIPBufferCalc
MACCalcRxChecksum	TCPIP_MAC_ChecksumRxCalc
MACConnect	TCPIP_MAC_Connect
MACPowerDown	TCPIP_MAC_PowerDown
MACPowerUp	TCPIP_MAC_PowerUp
MACGetHeader	TCPIP_MAC_HeaderGet
MACDiscardRx	TCPIP_MAC_RxDiscard
MACGet	TCPIP_MAC_Get
MACGetArray	TCPIP_MAC_ArrayGet
MACGetReadPtrInRx	TCPIP_MAC_RxReadPtrGet
MACGetRxSize	TCPIP_MAC_RxSizeGet
MACSetBaseReadPtr	TCPIP_MAC_ReadPtrBaseSet
MACSetReadPtr	TCPIP_MAC_ReadPtrSet
MACSetReadPtrInRx	TCPIP_MAC_RxReadPtrSet
MACIsRxReady	TCPIP_MAC_RxIsReady

MACRxAcknowledge	TCPIP_MAC_RxAcknowledge
MACRxGetPacket	TCPIP_MAC_RxPacketGet
MACRxProfile	TCPIP_MAC_RxProfile
MACRxRegisterNotifyHandler	TCPIP_MAC_RxNotifyHandlerRegister
MACRxFilterOperation	TCPIP_MAC_RxFilterOperation
MACRxFilterPatternMode	TCPIP_MAC_RxFilterPatternMode
MACRxFilterSetHashTableEntry	TCPIP_MAC_RxFilterHashTableEntrySet
MACTxPacket	TCPIP_MAC_TxPacket
MACTxProfile	TCPIP_MAC_TxProfile
MACTxRegisterNotifyHandler	TCPIP_MAC_TxNotifyHandlerRegister
MACIsTxReady	TCPIP_MAC_TxIsReady
MACFlush	TCPIP_MAC_Flush
MACGetSslBaseAddr	TCPIP_MAC_SslBaseAddrGet
MACGetTxBaseAddr	TCPIP_MAC_TxBaseAddrGet
MACGetTxBuffSize	TCPIP_MAC_TxBuffSizeGet
MACPut	TCPIP_MAC_Put
MACPutArray	TCPIP_MAC_ArrayPut
MACPutHeader	TCPIP_MAC_HeaderPut
MACSetWritePtr	TCPIP_MAC_WritePtrSet

5.8.3 Announce TCP/IP Stack Library

5.8.3.1 Introduction

Announce TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the Announce TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

This module will facilitate device discovery on DHCP enabled networks by broadcasting a UDP message on port 30303 whenever the local IP address changes.

5.8.3.2 Release Notes

MPLAB Harmony Version: v0.70b **Announce TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.3.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.3.4 Using the Library

This topic describes the basic architecture of the Announce TCPIP Stack Library and provides information and examples on how to use it.

Interface Header File: [tcpip_announce_config.h](#)

The interface to the Announce TCPIP Stack library is defined in the "[tcpip_announce_config.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the Announce TCPIP Stack library should include "tcpip.h".

Library File:

The Announce TCPIP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCPIP Stack interacts with the framework.

5.8.3.4.1 Abstraction Model

This library provides the API of the Announce TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description**Announce Software Abstraction Block Diagram**

Note: This diagram was not available at time of release and will included in a future release of MPLAB Harmony.

5.8.3.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Announce module.

Library Interface Section	Description
Functions	Routines for Configuring DHCP
Data Types and Constants	This section provides various definitions describing This API

5.8.3.5 Configuring the Library**Macros**

	Name	Description
	ANNOUNCE_MAX_PAYLOAD	Maximum size of a payload sent once
	ANNOUNCE_PORT	The announce port

Description

The configuration of the Announce TCP/IP Stack Library is based on the file tcpip_config.h

This header file contains the configuration selection for the Announce TCP/IP Stack Library. Based on the selections made, the Announce TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the AnnounceTCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.3.5.1 ANNOUNCE_MAX_PAYLOAD Macro**C**

```
#define ANNOUNCE_MAX_PAYLOAD (512)
```

Description

Maximum size of a payload sent once

5.8.3.5.2 ANNOUNCE_PORT Macro**C**

```
#define ANNOUNCE_PORT 30303
```

Description

The announce port

5.8.3.6 Building the Library

This section list the files that are available in the \src of the Announce driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.3.7 Library Interface

Data Types and Constants

	Name	Description
	TCPIP_ANNOUNCE_MODULE_CONFIG	Announce layer configuration/initialization

Functions

	Name	Description
	TCPIP_ANNOUNCE_DeInit	This is function TCPIP_ANNOUNCE_DeInit.
	TCPIP_ANNOUNCE_Init	manager
	TCPIP_ANNOUNCE_Send	This is function TCPIP_ANNOUNCE_Send.
	TCPIP_ANNOUNCE_Task	This is function TCPIP_ANNOUNCE_Task.
	TCPIP_ANNOUNCE_TaskPending	This is function TCPIP_ANNOUNCE_TaskPending.

Description

This section describes the Application Programming Interface (API) functions of the Announce TCP/IP Stack.

Refer to each section for a detailed description.

5.8.3.7.1 Functions

5.8.3.7.1.1 TCPIP_ANNOUNCE_DeInit Function

C

```
void TCPIP_ANNOUNCE_DeInit(  
    const TCPIP_STACK_MODULE_CTRL* const stackData  
);
```

Description

This is function TCPIP_ANNOUNCE_DeInit.

5.8.3.7.1.2 TCPIP_ANNOUNCE_Init Function

C

```
bool TCPIP_ANNOUNCE_Init(  
    const TCPIP_STACK_MODULE_CTRL* const stackData,  
    const TCPIP_ANNOUNCE_MODULE_CONFIG* announceData  
);
```

Description

manager

5.8.3.7.1.3 TCPIP_ANNOUNCE_Send Function

C

```
void TCPIP_ANNOUNCE_Send();
```

Description

This is function TCPIP_ANNOUNCE_Send.

5.8.3.7.1.4 TCPIP_ANNOUNCE_Task Function

C

```
bool TCPIP_ANNOUNCE_Task(TCPIP_NET_IF *pNetIf);
```

Description

This is function TCPIP_ANNOUNCE_Task.

5.8.3.7.1.5 TCPIP_ANNOUNCE_TaskPending Function

C

```
bool TCPIP_ANNOUNCE_TaskPending();
```

Description

This is function TCPIP_ANNOUNCE_TaskPending.

5.8.3.7.2 Data Types and Constants

5.8.3.7.2.1 TCPIP_ANNOUNCE_MODULE_CONFIG Structure

C

```
typedef struct {
} TCPIP_ANNOUNCE_MODULE_CONFIG;
```

Description

Announce layer configuration/initialization

5.8.3.8 Files

Files

Name	Description
tcpip_announce_config.h	Announce configuration file
tcpip_announce_manager.h	

Description

5.8.3.8.1 tcpip_announce_config.h

Announce Configuration file

This file contains the Announce module configuration options

Macros

	Name	Description
	ANNOUNCE_MAX_PAYLOAD	Maximum size of a payload sent once
	ANNOUNCE_PORT	The announce port

5.8.3.8.2 tcpip_announce_manager.h

TCPIP Announce Manager Header

Functions

	Name	Description
	TCPIP_ANNOUNCE_DeInit	This is function TCPIP_ANNOUNCE_DeInit.
	TCPIP_ANNOUNCE_Init	manager
	TCPIP_ANNOUNCE_Send	This is function TCPIP_ANNOUNCE_Send.
	TCPIP_ANNOUNCE_Task	This is function TCPIP_ANNOUNCE_Task.
	TCPIP_ANNOUNCE_TaskPending	This is function TCPIP_ANNOUNCE_TaskPending.

Structures

	Name	Description
	TCPIP_ANNOUNCE_MODULE_CONFIG	Announce layer configuration/initialization

5.8.4 ARP TCP/IP Stack Library

5.8.4.1 Introduction

Address Resolution Protocol (ARP) TCP/IP Stack Library

for

Microchip Microcontrollers

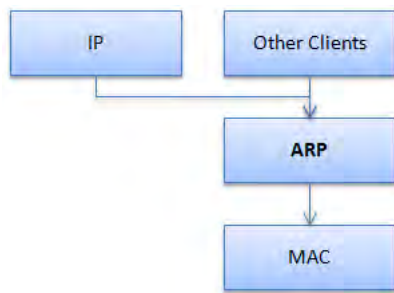
This library provides the API of the ARP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The Address Resolution Protocol, or ARP, is a foundation layer of TCP/IP. It translates IP addresses to physical MAC addresses, or locates a gateway through which a machine may be located.

TCP and UDP applications will not need to access ARP directly. The TCPOpen and UDPOpen functions will handle both ARP and DNS operations transparently.

Responses to incoming ARP requests are processed automatically. Resolution of ARP requests follows a simple state machine, as indicated in the following diagram.



5.8.4.2 Release Notes

MPLAB Harmony Version: v0.70b **ARP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.4.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.4.4 Using the Library

This topic describes the basic architecture of the ARP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [arp.h](#)

The interface to the arp TCP/IP Stack library is defined in the "[arp.h](#)" header file. This file is included by the "tcpip.h" file. Any C

language source (.c) file that uses the ARP TCP/IP Stack library should include "tcpip.h".

Library File:

The ARP TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

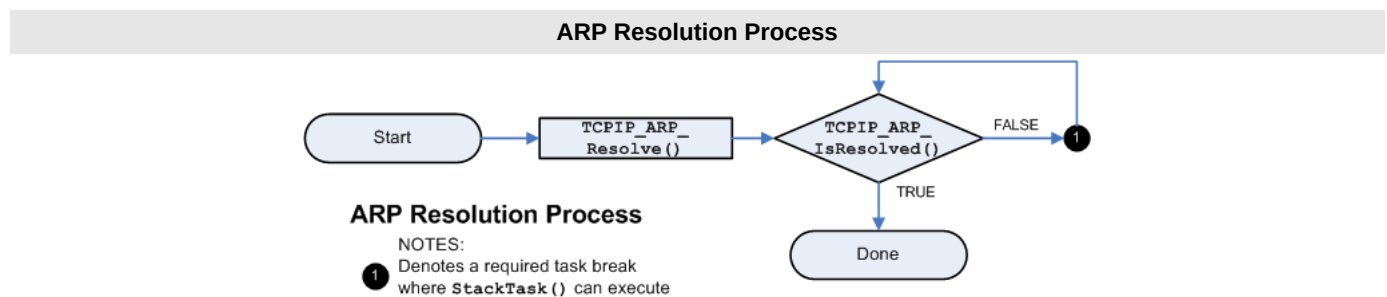
5.8.4.4.1 Abstraction Model

This library provides the API of the ARP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

ARP Software Abstraction Block Diagram

This module provides software abstraction of the ARP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.



5.8.4.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the ARP module.

Library Interface Section	Description
General Functions	Functions used by the ARP library interface.
Cache Manipulation Functions	This section provides an interface to the internal data storage (cache) that the ARP module implements.
Data Types and Constants	Derived types and enumerations used by the ARP library interface.

5.8.4.4.3 How the Library Works

The ARP module provides address resolution capabilities for the TCP/IP stack.

The ARP module is an independent module that maintains its state across calls and updates its state machine. A data storage (cache) is maintained and updated internally for each interface existent in the system. The number of entries in each cache is configurable at initialization time.

The ARP state machine can remove entries from the cache and can provide signals when a specific entry has been resolved or has timed out. The purging of the cache is done internally base on a time-out parameter that is dynamically configurable.

The module provides a notification mechanism which simplifies the design of the ARP clients. The most important client of the ARP module is the IPv4 layer.

5.8.4.5 Configuring the Library

Macros

	Name	Description
	<u>ARP_CACHE_ENTRIES_97J60</u>	This is macro ARP_CACHE_ENTRIES_97J60.
	<u>ARP_CACHE_ENTRIES_DEFAULT</u>	number of entries in the cache default number of entries per interface
	<u>ARP_CACHE_ENTRIES_ENCJ60</u>	This is macro ARP_CACHE_ENTRIES_ENCJ60.
	<u>ARP_CACHE_ENTRIES_ENCJ600</u>	This is macro ARP_CACHE_ENTRIES_ENCJ600.
	<u>ARP_CACHE_ENTRIES_MRF24W</u>	This is macro ARP_CACHE_ENTRIES_MRF24W.
	<u>ARP_CACHE_ENTRIES_PIC32INT</u>	specific interfaces numbers
	<u>ARP_CACHE_PENDING_ENTRY_TMO</u>	timeout for a pending to be solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been solved a solved entry moves to the solved entries timeout
	<u>ARP_CACHE_PENDING_RETRY_TMO</u>	timeout for resending an ARP request for a pending entry In order to prevent the ARP flooding the standard recommends it to be greater than 1 sec. Of course, it should be less than <u>ARP_CACHE_PENDING_ENTRY_TMO</u>
	<u>ARP_CACHE_PERMANENT_QUOTA</u>	max percentage of permanent entries in the cache note that since permanent entries cannot be removed they tend to degrade the efficiency of the cache look up
	<u>ARP_CACHE_PURGE_QUANTA</u>	how many entries to delete, once the threshold is reached
	<u>ARP_CACHE_PURGE_THRESHOLD</u>	default purge threshold, percentage once the number of resolved entries in the cache gets beyond the threshold some resolved entries will be purged
	<u>ARP_CACHE_SOLVED_ENTRY_TMO</u>	timeout for a solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been referenced again
	<u>ARP_TASK_PROCESS_RATE</u>	ARP task processing rate, in seconds the ARP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other ARP_CACHE_xxx_TMO are multiple of this

Description

The configuration of the TCP/IP Stack ARP is based on the file sys_config.h

This header file contains the configuration selection for the TCP/IP Stack ARP. Based on the selections made, the TCP/IP Stack ARP will support or not support selected features. These configuration settings will apply to the single instance of the TCP/IP Stack ARP.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.4.5.1 ARP_CACHE_ENTRIES_97J60 Macro

C

```
#define ARP_CACHE_ENTRIES_97J60 5
```

Description

This is macro ARP_CACHE_ENTRIES_97J60.

5.8.4.5.2 ARP_CACHE_ENTRIES_DEFAULT Macro

C

```
#define ARP_CACHE_ENTRIES_DEFAULT 5
```

Description

number of entries in the cache default number of entries per interface

5.8.4.5.3 ARP_CACHE_ENTRIES_ENCJ60 Macro

C

```
#define ARP_CACHE_ENTRIES_ENCJ60 5
```

Description

This is macro ARP_CACHE_ENTRIES_ENCJ60.

5.8.4.5.4 ARP_CACHE_ENTRIES_ENCJ600 Macro

C

```
#define ARP_CACHE_ENTRIES_ENCJ600 5
```

Description

This is macro ARP_CACHE_ENTRIES_ENCJ600.

5.8.4.5.5 ARP_CACHE_ENTRIES_MRF24W Macro

C

```
#define ARP_CACHE_ENTRIES_MRF24W 5
```

Description

This is macro ARP_CACHE_ENTRIES_MRF24W.

5.8.4.5.6 ARP_CACHE_ENTRIES_PIC32INT Macro

C

```
#define ARP_CACHE_ENTRIES_PIC32INT 5
```

Description

specific interaces numbers

5.8.4.5.7 ARP_CACHE_PENDING_ENTRY_TMO Macro

C

```
#define ARP_CACHE_PENDING_ENTRY_TMO (1 * 60)
```

Description

timeout for a pending to be solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been solved a solved entry moves to the solved entries timeout

5.8.4.5.8 ARP_CACHE_PENDING_RETRY_TMO Macro

C

```
#define ARP_CACHE_PENDING_RETRY_TMO (2)
```

Description

timeout for resending an ARP request for a pending entry In order to prevent the ARP flooding the standard recommends it to be greater than 1 sec. Of course, it should be less than [ARP_CACHE_PENDING_ENTRY_TMO](#)

5.8.4.5.9 ARP_CACHE_PERMANENT_QUOTA Macro

C

```
#define ARP_CACHE_PERMANENT_QUOTA 50
```

Description

max percentage of permanent entries in the cache note that since permanent entries cannot be removed they tend to degrade the efficiency of the cache look up

5.8.4.5.10 ARP_CACHE_PURGE_QUANTA Macro

C

```
#define ARP_CACHE_PURGE_QUANTA 3
```

Description

how many entries to delete, once the threshold is reached

5.8.4.5.11 ARP_CACHE_PURGE_THRESHOLD Macro

C

```
#define ARP_CACHE_PURGE_THRESHOLD 75
```

Description

default purge threshold, percentage once the number of resolved entries in the cache gets beyond the threshold some resolved entries will be purged

5.8.4.5.12 ARP_CACHE_SOLVED_ENTRY_TMO Macro

C

```
#define ARP_CACHE_SOLVED_ENTRY_TMO (20 * 60)
```

Description

timeout for a solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been referenced again

5.8.4.5.13 ARP_TASK_PROCESS_RATE Macro

C

```
#define ARP_TASK_PROCESS_RATE (2)
```

Description

ARP task processing rate, in seconds the ARP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other ARP_CACHE_XXX_TMO are multiple of this

5.8.4.6 Building the Library

This section list the files that are available in the \src of the ARP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- arp.c

5.8.4.7 Library Interface

Data Types and Constants

	Name	Description
	TCPIP_ARP_ENTRY_QUERY	ARP Entry Query
	TCPIP_ARP_ENTRY_TYPE	Type of ARP entry
	TCPIP_ARP_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved.
	TCPIP_ARP_EVENT_TYPE	Events reported by ARP
	TCPIP_ARP_HANDLE	ARP Handle
	TCPIP_ARP_OPERATION_TYPE	Type of ARP operation
	TCPIP_ARP_RESULT	ARP Results (success and failure codes)
	ARP_MODULE_CONFIG	This is type ARP_MODULE_CONFIG.
	arp_app_callbacks	TCPIP_STACK_USE_ZEROCONF_LINK_LOCAL API specific Definitions

Cache Manipulation Functions

	Name	Description
	TCPIP_ARP_CacheEntriesNoGet	Used to retrieve the number of entereies for a specific interface
	TCPIP_ARP_CacheThresholdSet	Sets the cache threshold for the specified interface in %
	TCPIP_ARP_EntryQuery	Querries an ARP cache entry using the index of the cache line.
	TCPIP_ARP_EntryRemoveAll	Removes all the mapping belonging to an interface.

General Functions

	Name	Description
	TCPIP_ARP_CallbacksDeregister	De-Registering callbacks with ARP module that are registered previously.
	TCPIP_ARP_CallbacksRegister	Registering callback with ARP module to get notified about certian events.
	TCPIP_ARP_Deinitialize	Deinitializes the ARP module.
	TCPIP_ARP_EntryGet	gets the current mapping for an IP address
	TCPIP_ARP_EntryRemove	removes the mapping of an address, even a permanent one
	TCPIP_ARP_EntryRemoveNet	Removes all the entries belonging to a network interface.
	TCPIP_ARP_HandlerDeRegister	deregister the event handler
	TCPIP_ARP_HandlerRegister	Register an ARP resolve handler
	TCPIP_ARP_Initialize	Initializes the ARP module.
	TCPIP_ARP_IsResolved	Determines if an ARP request has been resolved yet.
	TCPIP_ARP_Probe	Transmits an ARP probe to resolve an IP address.

	TCPIP_ARP_Resolve	Transmits an ARP request to resolve an IP address.
	TCPIP_ARP_Task	Performs ARP tasks from within TCPIP stack tasks.
	TCPIP_ARP_TaskIsPending	Returns true if an ARP task is pending, false otherwise.
	TCPIP_ARP_EntrySet	Adds an ARP cache entry for the specified interface.
	TCPIP_ARP_Process	Processes an incoming ARP packet.

Description

This section describes the Application Programming Interface (API) functions of the ARP TCP/IP Stack Library.

Refer to each section for a detailed description.

5.8.4.7.1 General Functions**5.8.4.7.1.1 TCPIP_ARP_CallbacksDeregister Function****C**

```
bool TCPIP_ARP_CallbacksDeregister(
    int8_t id
);
```

Description

This function allows end user-application to de-register with callbacks, which were registered previously. This is called by user-application, when its no longer interested in notifications from ARP-Module. This allows the other application to get registered with ARP-module.

Preconditions

None

Parameters

Parameters	Description
reg_id	Registration-id returned in TCPIP_ARP_CallbacksRegister call

Returns

true - On success false - Failure to indicate invalid reg_id

5.8.4.7.1.2 TCPIP_ARP_CallbacksRegister Function**C**

```
int8_t TCPIP_ARP_CallbacksRegister(
    struct arp_app_callbacks * app
);
```

Description

This function allows end user application to register with callbacks, which will be called by ARP module to give notification to user-application about events occurred at ARP layer. For ex: when a ARP-packet is received, which is conflicting with our own pair of addresses (MAC-Address and IP-address). This is an extension for zeroconf protocol implementation (ZeroconfLL.c)

Preconditions

None

Parameters

Parameters	Description
app	ARP-Application callbacks structure supplied by user-application

Returns

id > 0 - Returns non-negative value that represents the id of registration The same id needs to be used in de-registration -1 - When registered applications exceed MAX_REG_APPS and there is no free slot for registration

5.8.4.7.1.3 TCPIP_ARP_Deinitialize Function**C**

```
void TCPIP_ARP_Deinitialize(  
    const TCPIP_STACK_MODULE_CTRL* const stackData  
);
```

Description

Deinitializes the ARP module.

Preconditions

None.

Parameters

Parameters	Description
stackCtrl	stack initialization parameters, used by TCPIP_ARP_Initialize to initialize the ARP module.

Returns

None.

Remarks

None.

5.8.4.7.1.4 TCPIP_ARP_EntryGet Function**C**

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryGet(  
    TCPIP_NET_HANDLE hNet,  
    IPV4_ADDR* ipAdd,  
    TCPIP_MAC_ADDR* pHwAdd,  
    bool probe  
);
```

Description

If probe == false The function behaves identical to [TCPIP_ARP_IsResolved\(\)](#): If the corresponding MAC address exists in the cache it is copied to the user supplied pHwAdd

If probe == true The function behaves identical to [TCPIP_ARP_Resolve\(\)](#):

- If the corresponding MAC address does not exist in the cache

this function transmits an ARP request. Upon the address resolution it calls the registered handler (if available) with the supplied notification parameter (if != 0)

- If the hardware address exists, the result is written to pHwAdd

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	Interface to use
ipAdd	The ip address to get entries from
pHwAdd	pointer to store the hardware address
probe	boolean to specify if ARP probing is initiated or not

Returns

ARP_RES_ENTRY_SOLVED - if the required entry is already solved
ARP_RES_ENTRY_QUEUED - if the required entry was already queued
ARP_RES_ENTRY_NEW - if the operation succeeded and a new entry was added (and queued for resolving)
ARP_RES_CACHE_FULL - if new entry could not be inserted, the cache was full
ARP_RES_BAD_ADDRESS - bad address specified
ARP_RES_NO_INTERFACE - no such interface

Remarks

similar to [TCPIP_ARP_Resolve\(\)](#) + [TCPIP_ARP_IsResolved\(\)](#) It avoids a double hash search when the mapping exists.

Example

None

5.8.4.7.1.5 TCPIP_ARP_EntryRemove Function**C**

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryRemove(  
    TCPIP_NET_HANDLE hNet,  
    IPV4_ADDR* ipAdd  
);
```

Description

None

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	Interface to use
ipAdd	ip Address to remove entries for

Returns

[TCPIP_ARP_RESULT](#) On Success - On Failure - ARP_RES_NO_ENTRY (if no such mapping exists)

Remarks

None

Example

None

5.8.4.7.1.6 TCPIP_ARP_EntryRemoveNet Function

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveNet(  
    TCPIP_NET_HANDLE hNet,  
    IPV4_ADDR* ipAdd,  
    IPV4_ADDR* mask,  
    TCPIP_ARP_ENTRY_TYPE type  
);
```

Description

if(entry->type == type and entry->ipAdd & mask == ipAdd & mask) then remove entry

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	Interface handle to use
ipAdd	ip address
mask	ip address of mask
type	type of entries to remove:
valid types	ARP_ENTRY_TYPE_PERMANENT ARP_ENTRY_TYPE_COMPLETE ARP_ENTRY_TYPE_INCOMPLETE ARP_ENTRY_TYPE_ANY

Returns

ARP_RES_OK

Remarks

None

Example

None

5.8.4.7.1.7 TCPIP_ARP_HandlerDeRegister Function

C

```
bool TCPIP_ARP_HandlerDeRegister(  
    TCPIP_ARP_HANDLE hArp  
);
```

Description

None

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hArp	ARP Handle

Returns

bool On Success - true On Failure - false (If no such handler registered)

Remarks

None

Example

None

5.8.4.7.1.8 TCPIP_ARP_HandlerRegister Function

C

```
TCPIP_ARP_HANDLE TCPIP_ARP_HandlerRegister(  
    TCPIP_NET_HANDLE hNet,  
    TCPIP_ARP_EVENT_HANDLER handler,  
    const void* hParam  
);
```

Description

NONE

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	Specifies interface to register on. Use hNet == 0 to register on all interfaces available.
handler	Handler to be called for event.
hParam	The hParam is passed by the client and will be used by the ARP when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

Returns

TCPIP_ARP_HANDLE On Success - Returns a valid handle On Failure - null handle

Remarks

NONE

Example

None

5.8.4.7.1.9 TCPIP_ARP_Initialize Function

C

```
bool TCPIP_ARP_Initialize(  
    const TCPIP_STACK_MODULE_CTRL* const stackData,  
    const ARP_MODULE_CONFIG* arpData  
);
```

Description

Initializes the ARP module. Calls can be done with the request of not tearing down the ARP cache This helps for ifup/ifdown sequences. Of course, if this is the case the memory allocated for the ARP cache has to be from a persistent heap.

Preconditions

None

Parameters

Parameters	Description
stackCtrl	stack initialization parameters
arpData	ARP specific initialization parameters

Returns

true if initialization succeeded, false otherwise

Remarks

The request to maintain old ARP cache info (deleteOld field from the [ARP_MODULE_CONFIG](#) initialization data) is not implemented for stack nit/deinit sequences. To maintain the data after the stack is completely de-initialized would need a persistent heap that's not yet implemented. The selection cannot be changed by ifup since this operation does not carry ARP configuration parameters (arpData == 0).

5.8.4.7.1.10 TCPIP_ARP_IsResolved Function**C**

```
bool TCPIP_ARP_IsResolved(  
    TCPIP_NET_HANDLE hNet,  
    IPV4_ADDR* IPAddr,  
    TCPIP_MAC_ADDR* MACAddr  
);
```

Description

This function checks if an ARP request has been resolved yet, and if so, stores the resolved MAC address in the pointer provided.

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	interface to use
IPAddr	The IP address to be resolved. This must match the IP address provided to the TCPIP_ARP_Resolve() function call.
MACAddr	A buffer to store the corresponding MAC address retrieved from the ARP query.

Return Values

Return Values	Description
true	The IP address has been resolved and MACAddr MAC address field indicates the response.
false	The IP address is not yet resolved. Try calling TCPIP_ARP_IsResolved() again at a later time. If you don't get a response after a application specific timeout period, you may want to call TCPIP_ARP_Resolve() again to transmit another ARP query (in case if the original query or response was lost on the network). If you never receive an ARP response, this may indicate that the IP address isn't in use.

5.8.4.7.1.11 TCPIP_ARP_Probe Function**C**

```
TCPIP_ARP_RESULT TCPIP_ARP_Probe(  
    TCPIP_NET_HANDLE hNet,  
    IPV4_ADDR* IPAddr,
```

```
    IPV4_ADDR* srcAddr,  
    TCPIP_ARP_OPERATION_TYPE opType  
);
```

Description

This function transmits an ARP probe to determine the hardware address of a given IP address. The packet will use the type of operation and the source address specified as parameters.

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	interface to use
IPAddr	The IP address to be resolved. The address must be specified in network byte order (big endian).
srcAddr	The source address to be used in the ARP packet
opType	Operation code to be set in the outgoing ARP packet

Returns

ARP_RES_ENTRY_SOLVED - if the required entry is already solved
ARP_RES_ENTRY_QUEUED - if the required entry was already queued
ARP_RES_ENTRY_NEW - if the operation succeeded and a new entry was added (and queued for resolving)
ARP_RES_CACHE_FULL - if new entry could not be inserted, the cache was full
ARP_RES_BAD_ADDRESS - bad address specified
ARP_RES_NO_INTERFACE - no such interface

Remarks

This function is a more advanced version of [TCPIP_ARP_Resolve](#). It allows the caller to specify the operation type and the source address of the outgoing ARP packet.

No check is done for IPAddr to be valid.

To retrieve the ARP query result, call the [TCPIP_ARP_IsResolved\(\)](#) function.

5.8.4.7.1.12 TCPIP_ARP_Resolve Function

C

```
TCPIP_ARP_RESULT TCPIP_ARP_Resolve(  
    TCPIP_NET_HANDLE hNet,  
    IPV4_ADDR* IPAddr  
);
```

Description

This function transmits an ARP request to determine the hardware address of a given IP address. Upon the address resolution it calls the registered handler (if available) with the supplied notification parameter (if != 0)

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	interface to use
IPAddr	The IP address to be resolved. The address must be specified in network byte order (big endian).

Returns

An element from the [TCPIP_ARP_RESULT](#) enumeration. ARP_RES_ENTRY_SOLVED - if the required entry is already solved
ARP_RES_ENTRY_QUEUED - if the required entry was already queued
ARP_RES_ENTRY_NEW - if the operation succeeded and a new entry was added (and queued for resolving)
ARP_RES_CACHE_FULL - if new entry could not be inserted, the cache was full
ARP_RES_BAD_ADDRESS - bad address specified
ARP_RES_NO_INTERFACE - no such interface

Remarks

To retrieve the ARP query result, call the [TCPIP_ARP_IsResolved\(\)](#) function.

5.8.4.7.1.13 TCPIP_ARP_Task Function**C**

```
void TCPIP_ARP_Task();
```

Description

Performs ARP tasks from within TCPIP stack tasks.

Preconditions

None.

Returns

None.

Remarks

None.

5.8.4.7.1.14 TCPIP_ARP_TaskIsPending Function**C**

```
bool TCPIP_ARP_TaskIsPending();
```

Description

Returns true if an ARP task is pending, false otherwise.

Preconditions

None.

Returns

true - ARP task is pending
false - ARP task is not pending

5.8.4.7.1.15 TCPIP_ARP_EntrySet Function**C**

```
TCPIP_ARP_RESULT TCPIP_ARP_EntrySet(  
    TCPIP_NET_HANDLE hNet,  
    IPV4_ADDR* ipAdd,  
    TCPIP_MAC_ADDR* hwAdd,  
    bool perm  
);
```

Description

Can be added as permanent. Not subject to timeouts. If cache is full, an entry will be deleted to make room.

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	Interface to use
ipAdd	The ip address
hwAdd	
perm	

Returns

On Success - ARP_RES_OK/ARP_RES_ENTRY_EXIST ON Failure - An Error (for example, cache is full with permanent entries that cannot be purged or the permanent quota exceeded)

Remarks

None

Example

None

5.8.4.7.1.16 TCPIP_ARP_Process Function**C**

```
TCPIP_ARP_RESULT TCPIP_ARP_Process(  
    TCPIP_NET_IF* pIf,  
    TCPIP_MAC_PACKET* pPkt  
);
```

Description

Retrieves an ARP packet from the MAC buffer and determines if it is a response to our request (in which case the ARP is resolved) or if it is a request requiring our response (in which case we transmit one.)

Preconditions

ARP packet is ready in the MAC buffer.

Return Values

Return Values	Description
ARP_RES_OK	processing OK.
ARP_RES_error	some error occurred

5.8.4.7.2 Cache Manipulation Functions**5.8.4.7.2.1 TCPIP_ARP_CacheEntriesNoGet Function****C**

```
size_t TCPIP_ARP_CacheEntriesNoGet(  
    TCPIP_NET_HANDLE hNet,  
    TCPIP_ARP_ENTRY_TYPE type  
);
```

Description

None

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	Interface to use
type	Type of ARP entry

Returns

The number of entries of the specified type per interface

Remarks

None

Example

None

5.8.4.7.2.2 TCPIP_ARP_CacheThresholdSet Function**C**

```
TCPIP_ARP_RESULT TCPIP_ARP_CacheThresholdSet(  
    TCPIP_NET_HANDLE hNet,  
    int purgeThres,  
    int purgeEntries  
);
```

Description

once the number of entries in the cache is > than the threshold a number of purgeEntries (usually one) will be discarded

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	Interface handle to use
purgeThres	Threshold to start cache purging
purgeEntries	Number of entries to purge

Returns

ARP_RES_OK

Remarks

None

Example

None

5.8.4.7.2.3 TCPIP_ARP_EntryQuery Function**C**

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryQuery(  
    TCPIP_NET_HANDLE hNet,  
    size_t index,
```

```
    TCPIP_ARP_ENTRY_QUERY* pArpQuery  
);
```

Description

None

Preconditions

ARP module should have been initialized The index has to be a valid one i.e. < then [TCPIP_ARP_CacheEntriesNoGet\(\)](#) populates the supplied query routine if not NULL

Parameters

Parameters	Description
hNet	Interface handle to use
index	Index to cache
pArpQuery	entry type, ip address, hardware address

Returns

On Success - ARP_RES_OK On Failure - ARP_RES_BAD_INDEX (if index is out of range) use it for displaying the cache contents

Remarks

None

Example

None

5.8.4.7.2.4 TCPIP_ARP_EntryRemoveAll Function

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveAll(  
    TCPIP_NET_HANDLE hNet  
);
```

Description

None

Preconditions

ARP module should have been initialized

Parameters

Parameters	Description
hNet	network interface handle

Returns

ARP_RES_OK

Remarks

None

Example

None

5.8.4.7.3 Data Types and Constants

5.8.4.7.3.1 TCPIP_ARP_ENTRY_QUERY Structure

C

```
typedef struct {
    TCPIP_ARP_ENTRY_TYPE entryType;
    IPV4_ADDR entryIpAdd;
    TCPIP_MAC_ADDR entryHwAdd;
} TCPIP_ARP_ENTRY_QUERY;
```

Description

Structure: TCPIP_ARP_ENTRY_QUERY

None

Members

Members	Description
TCPIP_ARP_ENTRY_TYPE entryType;	what entry type
IPV4_ADDR entryIpAdd;	the entry IP address
TCPIP_MAC_ADDR entryHwAdd;	the entry hardware address

5.8.4.7.3.2 TCPIP_ARP_ENTRY_TYPE Enumeration

C

```
typedef enum {
    ARP_ENTRY_TYPE_INVALID,
    ARP_ENTRY_TYPE_PERMANENT,
    ARP_ENTRY_TYPE_COMPLETE,
    ARP_ENTRY_TYPE_INCOMPLETE,
    ARP_ENTRY_TYPE_ANY,
    ARP_ENTRY_TYPE_TOTAL
} TCPIP_ARP_ENTRY_TYPE;
```

Description

Enumeration: TCPIP_ARP_ENTRY_TYPE

None

Members

Members	Description
ARP_ENTRY_TYPE_INVALID	empty entry
ARP_ENTRY_TYPE_PERMANENT	entry valid and permanent
ARP_ENTRY_TYPE_COMPLETE	entry valid
ARP_ENTRY_TYPE_INCOMPLETE	entry not resolved yet
ARP_ENTRY_TYPE_ANY	any busy entry (PERMANENT COMPLETE INCOMPLETE)
ARP_ENTRY_TYPE_TOTAL	total entries - the number of entries the cache can store

Remarks

None

5.8.4.7.3.3 TCPIP_ARP_EVENT_HANDLER Type

C

```
typedef void (* TCPIP_ARP_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, const IPV4_ADDR* ipAdd, const
TCPIP_MAC_ADDR* MACAddr, TCPIP_ARP_EVENT_TYPE evType, const void* param);
```


Description

Type: TCPIP_ARP_EVENT_HANDLER

None

Remarks

The param member significance is module dependent. It can be an IP address, pointer to some other structure, etc. The handler is called when an event of some sort occurs for a particular IP address entry. If pNetIf == 0 then the notification is called for events on any interface

5.8.4.7.3.4 TCPIP_ARP_EVENT_TYPE Enumeration

C

```
typedef enum {
    ARP_EVENT_SOLVED = 0x01,
    ARP_EVENT_UPDATED = 0x02,
    ARP_EVENT_PERM_UPDATE = 0x04,
    ARP_EVENT_TMO = -1
} TCPIP_ARP_EVENT_TYPE;
```

Description

Enumeration: TCPIP_ARP_EVENT_TYPE

None

Members

Members	Description
ARP_EVENT_SOLVED = 0x01	a queued cache entry was solved;
ARP_EVENT_UPDATED = 0x02	an existent cache entry was updated
ARP_EVENT_PERM_UPDATE = 0x04	an update for an permanent entry was received however the permanent entry was not updated
ARP_EVENT_TMO = -1	an entry could not be solved and a tmo occurred

Remarks

Possibly multiple events can be set, where it makes sense.

5.8.4.7.3.5 TCPIP_ARP_HANDLE Type

C

```
typedef const void* TCPIP_ARP_HANDLE;
```

Description

Type: TCPIP_ARP_HANDLE

A handle that a client can use.

Remarks

This handle can be used by the client after the event handler has been registered.

5.8.4.7.3.6 TCPIP_ARP_OPERATION_TYPE Enumeration

C

```
typedef enum {
    ARP_OPERATION_REQ = 1,
```

```

    ARP_OPERATION_RESP = 2,
    ARP_OPERATION_MASK = 0x000f,
    ARP_OPERATION_CONFIGURE = 0x1000
} TCPIP_ARP_OPERATION_TYPE;

```

Description

Enumeration: TCPIP_ARP_OPERATION_TYPE

Operation to be performed by an ARP probe

Members

Members	Description
ARP_OPERATION_REQ = 1	ARP request
ARP_OPERATION_RESP = 2	ARP response
ARP_OPERATION_MASK = 0x000f	extract ARP operation
ARP_OPERATION_CONFIGURE = 0x1000	stack configuration ARP packet

Remarks

Used for low level functionality, [TCPIP_ARP_Probe](#)

5.8.4.7.3.7 TCPIP_ARP_RESULT Enumeration**C**

```

typedef enum {
    ARP_RES_OK = 0,
    ARP_RES_ENTRY_NEW,
    ARP_RES_ENTRY_SOLVED,
    ARP_RES_ENTRY_QUEUED,
    ARP_RES_ENTRY_EXIST,
    ARP_RES_PERM_QUOTA_EXCEED,
    ARP_RES_NO_ENTRY = -1,
    ARP_RES_CACHE_FULL = -2,
    ARP_RES_TX_FAILED = -3,
    ARP_RES_BAD_INDEX = -4,
    ARP_RES_BAD_ADDRESS = -5,
    ARP_RES_NO_INTERFACE = -6,
    ARP_RES_BAD_TYPE = -7,
    ARP_RES_CONFIGURE_ERR = -8
} TCPIP_ARP_RESULT;

```

Description

Enumeration: TCPIP_ARP_RESULT

Various Definitions for Success and Failure Codes

Members

Members	Description
ARP_RES_OK = 0	operation succeeded
ARP_RES_ENTRY_NEW	operation succeeded and a new entry was added
ARP_RES_ENTRY_SOLVED	the required entry is already solved
ARP_RES_ENTRY_QUEUED	the required entry was already queued
ARP_RES_ENTRY_EXIST	the required entry was already cached
ARP_RES_PERM_QUOTA_EXCEED	info: the quota of permanent entries was exceeded
ARP_RES_NO_ENTRY = -1	no such entry exists
ARP_RES_CACHE_FULL = -2	the cache is full and no entry could be removed to make room
ARP_RES_TX_FAILED = -3	failed to transmit an ARP message

ARP_RES_BAD_INDEX = -4	bad query index
ARP_RES_BAD_ADDRESS = -5	bad IP address specified
ARP_RES_NO_INTERFACE = -6	no such interface exists
ARP_RES_BAD_TYPE = -7	no such type is valid/exists
ARP_RES_CONFIGURE_ERR = -8	interface is configuring now, no ARP probes

Remarks

None

5.8.4.7.3.8 ARP_MODULE_CONFIG Structure**C**

```
typedef struct {
    size_t cacheEntries;
    bool deleteOld;
    int entrySolvedTmo;
    int entryPendingTmo;
    int entryRetryTmo;
    int permQuota;
    int purgeThres;
    int purgeQuanta;
} ARP_MODULE_CONFIG;
```

Description

This is type ARP_MODULE_CONFIG.

Members

Members	Description
size_t cacheEntries;	cache entries for this interface
bool deleteOld;	delete old cache if still in place, else don't re-initialize it
int entrySolvedTmo;	solved entry removed after this tmo if not referenced - seconds
int entryPendingTmo;	timeout for a pending to be solved entry in the cache, in seconds
int entryRetryTmo;	timeout for resending an ARP request for a pending entry - seconds 1 sec < tmo < entryPendingTmo
int permQuota;	max percentage of permanent entries allowed in the cache -
int purgeThres;	purge threshold -
int purgeQuanta;	no of entries to delete once the threshold is reached

5.8.4.7.3.9 arp_app_callbacks Structure**C**

```
struct arp_app_callbacks {
    bool used;
    void (* TCPIP_ARP_PacketNotify)(TCPIP_NET_IF* pNetIf, uint32_t SenderIPAddr, uint32_t TargetIPAddr, TCPIP_MAC_ADDR* SenderMACAddr, TCPIP_MAC_ADDR* TargetMACAddr, uint8_t op_req);
};
```

Description

TCPIP_STACK_USE_ZEROCONF_LINK_LOCAL API specific Definitions

5.8.4.8 Files

Files

Name	Description
arp.h	This is file arp.h.
arp_config.h	ARP configuration file
arp_manager.h	Stack internal definitions for ARP module

Description

5.8.4.8.1 arp.h

This is file arp.h.

Enumerations

	Name	Description
	TCPIP_ARP_ENTRY_TYPE	Type of ARP entry
	TCPIP_ARP_EVENT_TYPE	Events reported by ARP
	TCPIP_ARP_OPERATION_TYPE	Type of ARP operation
	TCPIP_ARP_RESULT	ARP Results (success and failure codes)

Functions

	Name	Description
	TCPIP_ARP_CacheEntriesNoGet	Used to retrieve the number of entereies for a specific interface
	TCPIP_ARP_CacheThresholdSet	Sets the cache threshold for the specified interface in %
	TCPIP_ARP_EntryGet	gets the current mapping for an IP address
	TCPIP_ARP_EntryQuery	Queries an ARP cache entry using the index of the cache line.
	TCPIP_ARP_EntryRemove	removes the mapping of an address, even a permanent one
	TCPIP_ARP_EntryRemoveAll	Removes all the mapping belonging to an interface.
	TCPIP_ARP_EntryRemoveNet	Removes all the entries belonging to a network interface.
	TCPIP_ARP_EntrySet	Adds an ARP cache entry for the specified interface.
	TCPIP_ARP_HandlerDeRegister	deregister the event handler
	TCPIP_ARP_HandlerRegister	Register an ARP resolve handler
	TCPIP_ARP_IsResolved	Determines if an ARP request has been resolved yet.
	TCPIP_ARP_Probe	Transmits an ARP probe to resolve an IP address.
	TCPIP_ARP_Resolve	Transmits an ARP request to resolve an IP address.

Structures

	Name	Description
	ARP_MODULE_CONFIG	This is type ARP_MODULE_CONFIG.
	TCPIP_ARP_ENTRY_QUERY	ARP Entry Query

Types

	Name	Description
	TCPIP_ARP_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved.

	TCPIP_ARP_HANDLE	ARP Handle

5.8.4.8.2 arp_config.h

Address Resolution Protocol (ARP) Configuration file

This file contains the ARP module configuration options

Macros

	Name	Description
	ARP_CACHE_ENTRIES_97J60	This is macro ARP_CACHE_ENTRIES_97J60.
	ARP_CACHE_ENTRIES_DEFAULT	number of entries in the cache default number of entries per interface
	ARP_CACHE_ENTRIES_ENCJ60	This is macro ARP_CACHE_ENTRIES_ENCJ60.
	ARP_CACHE_ENTRIES_ENCJ600	This is macro ARP_CACHE_ENTRIES_ENCJ600.
	ARP_CACHE_ENTRIES_MRF24W	This is macro ARP_CACHE_ENTRIES_MRF24W.
	ARP_CACHE_ENTRIES_PIC32INT	specific interaces numbers
	ARP_CACHE_PENDING_ENTRY_TMO	timeout for a pending to be solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been solved a solved entry moves to the solved entries timeout
	ARP_CACHE_PENDING_RETRY_TMO	timeout for resending an ARP request for a pending entry In order to prevent the ARP flooding the standard recommends it to be greater than 1 sec. Of course, it should be less than ARP_CACHE_PENDING_ENTRY_TMO
	ARP_CACHE_PERMANENT_QUOTA	max percentage of permanent entries in the cache note that since permanent entries cannot be removed they tend to degrade the efficiency of the cache look up
	ARP_CACHE_PURGE_QUANTA	how many entries to delete, once the threshold is reached
	ARP_CACHE_PURGE_THRESHOLD	default purge threshold, percentage once the number of resolved entries in the cache gets beyond the threshold some resolved entries will be purged
	ARP_CACHE_SOLVED_ENTRY_TMO	timeout for a solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been referenced again
	ARP_TASK_PROCESS_RATE	ARP task processing rate, in seconds the ARP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other ARP_CACHE_xxx_TMO are multiple of this

5.8.4.8.3 arp_manager.h

ARP module manager header

This file contains the stack internal API for the ARP module

Functions

	Name	Description
	TCPIP_ARP_CallbacksDeregister	De-Registering callbacks with ARP module that are registered previously.
	TCPIP_ARP_CallbacksRegister	Registering callback with ARP module to get notified about certian events.
	TCPIP_ARP_Deinitialize	Deinitializes the ARP module.
	TCPIP_ARP_Initialize	Initializes the ARP module.
	TCPIP_ARP_Process	Processes an incoming ARP packet.
	TCPIP_ARP_Task	Performs ARP tasks from within TCPIP stack tasks.

	TCPIP_ARP_TaskIsPending	Returns true if an ARP task is pending, false otherwise.

Structures

	Name	Description
	arp_app_callbacks	TCPIP_STACK_USE_ZEROCONF_LINK_LOCAL API specific Definitions

5.8.5 Berkeley TCP/IP Stack Library

5.8.5.1 Introduction

Berkeley TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the Berkeley TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs. Using this interface, programmers familiar with BSD sockets can quickly develop applications using Microchip's TCP/IP Stack.

The illustration below shows a typical interaction for a TCP server or client using the Berkeley [socket](#) APIs.

5.8.5.2 Release Notes

MPLAB Harmony Version: v0.70b **Berkeley TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.5.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO

YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.5.4 Using the Library

This topic describes the basic architecture of the Berkeley TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [berkeley_api.h](#)

The interface to the Berkeley TCP/IP Stack library is defined in the "[berkeley_api.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the berkeley TCP/IP Stack library should include "tcpip.h".

Library File:

The Berkeley TCP/IP Stack library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

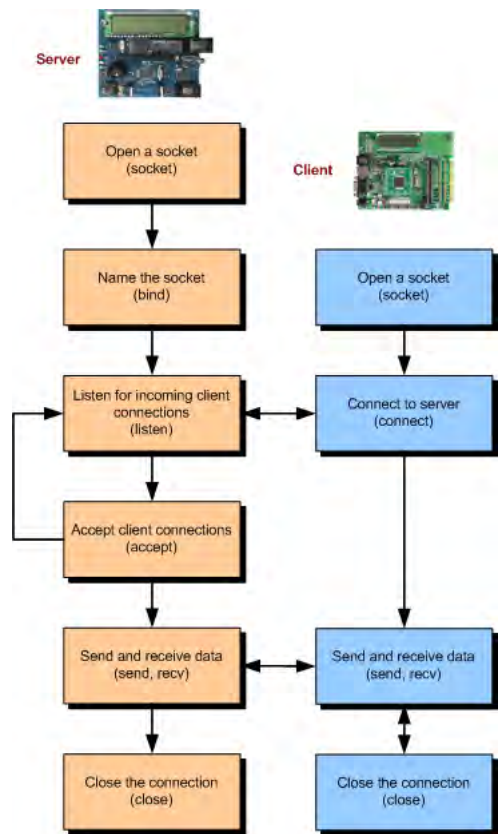
5.8.5.4.1 Abstraction Model

This library provides the API of the Berkeley TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Berkeley Software Abstraction Block Diagram

The illustration below shows a typical interaction for a TCP server or client using the Berkeley [socket](#) APIs.



5.8.5.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Berkeley module.

Library Interface Section	Description
Functions	Routines for Configuring DHCP
Data Types and Constants	This section provides various definitions describing This API

5.8.5.5 Configuring the Library

Macros

	Name	Description
	MAX_BSD_CLIENT_CONNECTIONS	Berkeley API client Sockets simultaneous supported connections.
	MAX_BSD_SERVER_CONNECTIONS	Berkeley API server Sockets simultaneous supported connections.

Description

The configuration of the Berkeley TCP/IP Stack is based on the file tcpip_config.h

This header file contains the configuration selection for the Berkeley TCP/IP Stack. Based on the selections made, the Berkeley TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the Berkeley TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.5.5.1 MAX_BSD_CLIENT_CONNECTIONS Macro

C

```
#define MAX_BSD_CLIENT_CONNECTIONS (2)
```

Description

Berkeley API client Sockets simultaneous supported connections.

5.8.5.5.2 MAX_BSD_SERVER_CONNECTIONS Macro

C

```
#define MAX_BSD_SERVER_CONNECTIONS (2)
```

Description

Berkeley API server Sockets simultaneous supported connections.

5.8.5.6 Building the Library

This section list the files that are available in the \src of the Berkeley driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.5.7 Library Interface

Data Types and Constants

	Name	Description
	AF_INET	Internet Address Family - UDP, TCP, etc.
	INADDR_ANY	IP address for server binding.
	INVALID_TCP_PORT	Invalid TCP port
	IP_ADDR_ANY	IP Address for server binding
	IPPROTO_IP	Indicates IP pseudo-protocol.
	IPPROTO_TCP	Indicates TCP for the internet address family.
	IPPROTO_UDP	Indicates UDP for the internet address family.
	SOCK_DGRAM	Connectionless datagram socket . Use UDP for the internet address family.
	SOCK_STREAM	Connection based byte streams. Use TCP for the internet address family.
	SOCKET_CNXN_IN_PROGRESS	Socket connection state.
	SOCKET_DISCONNECTED	Socket disconnected
	SOCKET_ERROR	Socket error
	BERKELEY_MODULE_CONFIG	Berkeley API module configuration structure
	SOCKADDR	generic address structure for all address families
	SOCKADDR_IN	In the Internet address family
	SOCKET	Socket descriptor
	in_addr	in_addr structure

	sockaddr	generic address structure for all address families
	sockaddr_in	In the Internet address family

Functions

	Name	Description
	accept	This function accepts connection requests queued for a listening socket .
	bind	This function assigns a name to the socket descriptor.
	closesocket	The closesocket function closes an existing socket .
	connect	This function connects to the peer communications end point.
	gethostname	Returns the standard host name for the system.
	listen	The listen function sets the specified socket in a listen mode
	recv	The recv() function is used to receive incoming data that has been queued for a socket .
	recvfrom	The recvfrom() function is used to receive incoming data that has been queued for a socket .
	send	The send function is used to send outgoing data on an already connected socket .
	sendto	This function used to send the data for both connection oriented and connection-less sockets.
	socket	This function creates a new Berkeley socket.

Description

This section describes the Application Programming Interface (API) functions of the Berkeley TCP/IP Stack.

Refer to each section for a detailed description.

5.8.5.7.1 Functions

5.8.5.7.1.1 accept Function

C

```
SOCKET accept(
    SOCKET s,
    struct sockaddr* addr,
    int* addrlen
);
```

Description

The accept function is used to accept connection requests queued for a listening [socket](#). If a connection request is pending, accept removes the request from the queue, and a new [socket](#) is created for the connection. The original listening [socket](#) remains open and continues to queue new connection requests. The [socket](#) must be a [SOCK_STREAM](#) type [socket](#).

Preconditions

[listen](#) function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket . must be bound to a local name and in listening mode.
addr	Optional pointer to a buffer that receives the address of the connecting entity.
addrlen	Optional pointer to an integer that contains the length of the address addr

Returns

If the `accept` function succeeds, it returns a non-negative integer that is a descriptor for the accepted [socket](#). Otherwise, the value [SOCKET_ERROR](#) is returned. (and `errno` set accordingly).

Remarks

None.

5.8.5.7.1.2 bind Function**C**

```
int bind(  
    SOCKET s,  
    const struct sockaddr* name,  
    int namelen  
);
```

Description

The `bind` function assigns a name to an unnamed [socket](#). The name represents the local address of the communication endpoint. For sockets of type [SOCK_STREAM](#), the name of the remote endpoint is assigned when a [connect](#) or [accept](#) function is executed.

Preconditions

[socket](#) function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket .
name	pointer to the sockaddr structure containing the local address of the socket .
namelen	length of the sockaddr structure.

Returns

If `bind` is successful, a value of 0 is returned. A return value of [SOCKET_ERROR](#) indicates an error. (and `errno` set accordingly).

Remarks

None.

5.8.5.7.1.3 closesocket Function**C**

```
int closesocket(  
    SOCKET s  
);
```

Description

The `closesocket` function closes an existing [socket](#). This function releases the [socket](#) descriptor s. Any data buffered at the [socket](#) is discarded. If the [socket](#) s is no longer needed, `closesocket()` must be called in order to release all resources associated with s.

Preconditions

None.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket

Returns

If `closesocket` is successful, a value of 0 is returned. A return value of [SOCKET_ERROR](#) (-1) indicates an error. (and `errno` set accordingly).

Remarks

None.

5.8.5.7.1.4 connect Function**C**

```
int connect(  
    SOCKET s,  
    struct sockaddr* name,  
    int namelen  
);
```

Description

The `connect` function assigns the address of the peer communications endpoint. For stream sockets, connection is established between the endpoints. For datagram sockets, an address filter is established between the endpoints until changed with another `connect()` function.

Preconditions

[socket](#) function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket .
name	pointer to the sockaddr structure containing the peer address and port number.
namelen	length of the sockaddr structure.

Returns

If the `connect()` function succeeds, it returns 0. Otherwise, the value [SOCKET_ERROR](#) is returned to indicate an error condition (and `errno` set accordingly). For stream based [socket](#), if the connection is not established yet, `connect` returns [SOCKET_ERROR](#) and `errno` = `EINPROGRESS`.

Remarks

None.

5.8.5.7.1.5 gethostname Function**C**

```
int gethostname(  
    char* name,  
    int namelen  
);
```

Description

This function returns the standard host name of the system which is calling this function. The returned name is null-terminated.

Preconditions

None.

Parameters

Parameters	Description
name	Pointer to a buffer that receives the local host name.
namelen	size of the name array.

Returns

Success will return a value of 0. If name is too short to hold the host name or any other error occurs, [SOCKET_ERROR](#) (-1) will be returned (and errno set accordingly). On error, *name will be unmodified and no null terminator will be generated.

Remarks

The function returns the host name as set on the default network interface.

5.8.5.7.1.6 listen Function**C**

```
int listen(  
    SOCKET s,  
    int backlog  
);
```

Description

This function sets the specified [socket](#) in a listen mode. Calling the listen function indicates that the application is ready to [accept](#) connection requests arriving at a [socket](#) of type [SOCK_STREAM](#). The connection request is queued (if possible) until accepted with an [accept](#) function. The backlog parameter defines the maximum number of pending connections that may be queued.

Preconditions

[bind\(\)](#) must have been called on the s [socket](#) first.

Parameters

Parameters	Description
s	Socket identifier returned from a prior socket() call.
backlog	Maximum number of connection requests that can be queued. Note that each backlog requires a TCP socket to be allocated.

Returns

Returns 0 on success, else return [SOCKET_ERROR](#). (and errno set accordingly).

Remarks

None

5.8.5.7.1.7 recv Function**C**

```
int recv(  
    SOCKET s,  
    char* buf,  
    int len,  
    int flags  
);
```

Description

The `recv()` function is used to receive incoming data that has been queued for a `socket`. This function can be used with both datagram and stream `socket`. If the available data is too large to fit in the supplied application buffer `buf`, excess bytes are discarded in case of `SOCK_DGRAM` type sockets. For `SOCK_STREAM` types, the data is buffered internally so the application can retrieve all data by multiple calls of `recvfrom`.

Preconditions

`connect` function should be called for TCP and UDP sockets. Server side, `accept` function should be called.

Parameters

Parameters	Description
<code>s</code>	Socket descriptor returned from a previous call to <code>socket</code> .
<code>buf</code>	application data receive buffer.
<code>len</code>	buffer length in bytes.
<code>flags</code>	no significance in this implementation

Returns

If `recv` is successful, the number of bytes copied to application buffer `buf` is returned. A return value of `SOCKET_ERROR` (-1) indicates an error condition (and `errno` set accordingly). A value of zero indicates `socket` has been shutdown by the peer.

Remarks

None.

5.8.5.7.1.8 `recvfrom` Function

C

```
int recvfrom(  
    SOCKET s,  
    char* buf,  
    int len,  
    int flags,  
    struct sockaddr* from,  
    int* fromlen  
);
```

Description

The `recvfrom()` function is used to receive incoming data that has been queued for a `socket`. This function can be used with both datagram and stream type sockets. If the available data is too large to fit in the supplied application buffer `buf`, excess bytes are discarded in case of `SOCK_DGRAM` type sockets. For `SOCK_STREAM` types, the data is buffered internally so the application can retrieve all data by multiple calls of `recvfrom`.

Preconditions

`socket` function should be called.

Parameters

Parameters	Description
<code>s</code>	Socket descriptor returned from a previous call to <code>socket</code> .
<code>buf</code>	application data receive buffer.
<code>len</code>	buffer length in bytes.
<code>flags</code>	message flags. Currently this is not supported.
<code>from</code>	pointer to the <code>sockaddr</code> structure that will be filled in with the destination address.
<code>fromlen</code>	size of buffer pointed by <code>from</code> .

Returns

If `recvfrom` is successful, the number of bytes copied to application buffer `buf` is returned. A return value of `SOCKET_ERROR` (-1) indicates an error condition (and `errno` set accordingly). A value of zero indicates `socket` has been shutdown by the peer.

Remarks

None.

5.8.5.7.1.9 send Function**C**

```
int send(  
    SOCKET s,  
    const char* buf,  
    int len,  
    int flags  
);
```

Description

The `send` function is used to send outgoing data on an already connected `socket`. This function is used to send a reliable, ordered stream of data bytes on a `socket` of type `SOCK_STREAM` but can also be used to send datagrams on a `socket` of type `SOCK_DGRAM`.

Preconditions

`connect` function should be called for TCP and UDP sockets. Server side, `accept` function should be called.

Parameters

Parameters	Description
<code>s</code>	Socket descriptor returned from a previous call to <code>socket</code> .
<code>buf</code>	application data buffer containing data to transmit.
<code>len</code>	length of data in bytes.
<code>flags</code>	message flags. Currently this field is not supported.

Returns

On success, `send` returns number of bytes sent. Zero indicates no data send. In case of error it returns `SOCKET_ERROR` (and `errno` set accordingly).

Remarks

None.

5.8.5.7.1.10 sendto Function**C**

```
int sendto(  
    SOCKET s,  
    const char* buf,  
    int len,  
    int flags,  
    const struct sockaddr* to,  
    int tolen  
);
```

Description

The `sendto` function is used to `send` outgoing data on a `socket`. The destination address is given by `to` and `tolen`. Both Datagram and stream sockets are supported.

Preconditions

[socket](#) function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket .
buf	application data buffer containing data to transmit.
len	length of data in bytes.
flags	message flags. Currently this field is not supported.
to	Optional pointer to the the sockaddr structure containing the destination address. If NULL, the currently bound remote port and IP address are used as the destination.
tolen	length of the sockaddr structure.

Returns

On success, sendto returns number of bytes sent. In case of error returns [SOCKET_ERROR](#) (and errno set accordingly).

Remarks

None.

5.8.5.7.1.11 socket Function**C**

```
SOCKET socket(  
    int af,  
    int type,  
    int protocol  
);
```

Description

This function creates a new BSD socket for the microchip TCPIP stack. The return socket descriptor is used for the subsequent BSD operations.

Preconditions

BerkeleySocketInit function should be called.

Parameters

Parameters	Description
af	address family - AF_INET .
type	socket type SOCK_DGRAM or SOCK_STREAM .
protocol	IP protocol IPPROTO_UDP or IPPROTO_TCP .

Returns

New socket descriptor. [SOCKET_ERROR](#) in case of error. (and errno set accordingly).

Remarks

None.

5.8.5.7.2 Data Types and Constants

5.8.5.7.2.1 AF_INET Macro

C

```
#define AF_INET 2           // Internet Address Family - UDP, TCP, etc.
```

Description

Internet Address Family - UDP, TCP, etc.

5.8.5.7.2.2 INADDR_ANY Macro

C

```
#define INADDR_ANY 0x00000000u // IP address for server binding.
```

Description

IP address for server binding.

5.8.5.7.2.3 INVALID_TCP_PORT Macro

C

```
#define INVALID_TCP_PORT (0L) //Invalid TCP port
```

Description

Invalid TCP port

5.8.5.7.2.4 IP_ADDR_ANY Macro

C

```
#define IP_ADDR_ANY 0u // IP Address for server binding
```

Description

IP Address for server binding

5.8.5.7.2.5 IPPROTO_IP Macro

C

```
#define IPPROTO_IP 0 // Indicates IP pseudo-protocol.
```

Description

Indicates IP pseudo-protocol.

5.8.5.7.2.6 IPPROTO_TCP Macro

C

```
#define IPPROTO_TCP 6 // Indicates TCP for the internet address family.
```

Description

Indicates TCP for the internet address family.

5.8.5.7.2.7 IPPROTO_UDP Macro

C

```
#define IPPROTO_UDP 17 // Indicates UDP for the internet address family.
```

Description

Indicates UDP for the internet address family.

5.8.5.7.2.8 SOCK_DGRAM Macro**C**

```
#define SOCK_DGRAM 110 //Connectionless datagram socket. Use UDP for the internet address family.
```

Description

Connectionless datagram [socket](#). Use UDP for the internet address family.

5.8.5.7.2.9 SOCK_STREAM Macro**C**

```
#define SOCK_STREAM 100 //Connection based byte streams. Use TCP for the internet address family.
```

Description

Connection based byte streams. Use TCP for the internet address family.

5.8.5.7.2.10 SOCKET_CNXN_IN_PROGRESS Macro**C**

```
#define SOCKET_CNXN_IN_PROGRESS (-2) //Socket connection state.
```

Description

Socket connection state.

5.8.5.7.2.11 SOCKET_DISCONNECTED Macro**C**

```
#define SOCKET_DISCONNECTED (-3) //Socket disconnected
```

Description

Socket disconnected

5.8.5.7.2.12 SOCKET_ERROR Macro**C**

```
#define SOCKET_ERROR (-1) //Socket error
```

Description

Socket error

5.8.5.7.2.13 BERKELEY_MODULE_CONFIG Structure**C**

```
typedef struct {  
} BERKELEY_MODULE_CONFIG;
```

Description

Berkeley API module configuration structure

5.8.5.7.2.14 SOCKADDR Type

C

```
typedef struct sockaddr SOCKADDR;
```

Description

generic address structure for all address families

5.8.5.7.2.15 SOCKADDR_IN Type

C

```
typedef struct sockaddr_in SOCKADDR_IN;
```

Description

In the Internet address family

5.8.5.7.2.16 SOCKET Type

C

```
typedef int16_t SOCKET;
```

Description

Socket descriptor

5.8.5.7.2.17 in_addr Structure

C

```
struct in_addr {  
    union {  
        struct {  
            uint8_t s_b1, s_b2, s_b3, s_b4;  
        } S_un_b;  
        struct {  
            uint16_t s_w1, s_w2;  
        } S_un_w;  
        uint32_t S_addr;  
    } S_un;  
};
```

Description

in_addr structure

Members

Members	Description
union { struct { uint8_t s_b1, s_b2, s_b3, s_b4; } S_un_b; struct { uint16_t s_w1, s_w2; } S_un_w; uint32_t S_addr; } S_un;	union of IP address
struct { uint8_t s_b1, s_b2, s_b3, s_b4; } S_un_b;	IP address in Byte
struct { uint16_t s_w1, s_w2; } S_un_w;	IP address in Word
uint32_t S_addr;	IP address

5.8.5.7.2.18 sockaddr Structure**C**

```

struct sockaddr {
    unsigned short sa_family;
    char sa_data[14];
};

```

Description

generic address structure for all address families

Members

Members	Description
unsigned short sa_family;	address family
char sa_data[14];	up to 14 bytes of direct address

5.8.5.7.2.19 sockaddr_in Structure**C**

```

struct sockaddr_in {
    short sin_family;
    uint16_t sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};

```

Description

In the Internet address family

Members

Members	Description
short sin_family;	Address family; must be AF_INET .
uint16_t sin_port;	Internet Protocol (IP) port.

struct in_addr sin_addr;	IP address in network byte order.
char sin_zero[8];	Padding to make structure the same size as SOCKADDR .

5.8.5.8 Files

Files

Name	Description
berekely_api_config.h	Berekely Sockets API Client and Server
berkeley_api.h	

Description

5.8.5.8.1 berekely_api_config.h

Berekely Sockets API Configuration file

This file contains the BerekelyAPI module configuration options

Macros

	Name	Description
	MAX_BSD_CLIENT_CONNECTIONS	Berkeley API client Sockets simultaneous supported connections.
	MAX_BSD_SERVER_CONNECTIONS	Berkeley API server Sockets simultaneous supported connections.

5.8.5.8.2 berkeley_api.h

Berekely Socket Distribution API Header File

Functions

	Name	Description
	accept	This function accepts connection requests queued for a listening socket .
	bind	This function assigns a name to the socket descriptor.
	closesocket	The closesocket function closes an existing socket .
	connect	This function connects to the peer communications end point.
	gethostname	Returns the standard host name for the system.
	listen	The listen function sets the specified socket in a listen mode
	recv	The recv() function is used to receive incoming data that has been queued for a socket .
	recvfrom	The recvfrom() function is used to receive incoming data that has been queued for a socket .
	send	The send function is used to send outgoing data on an already connected socket .
	sendto	This function used to send the data for both connection oriented and connection-less sockets.
	socket	This function creates a new Berkeley socket.

Macros

	Name	Description
	AF_INET	Internet Address Family - UDP, TCP, etc.

	INADDR_ANY	IP address for server binding.
	INVALID_TCP_PORT	Invalid TCP port
	IP_ADDR_ANY	IP Address for server binding
	IPPROTO_IP	Indicates IP pseudo-protocol.
	IPPROTO_TCP	Indicates TCP for the internet address family.
	IPPROTO_UDP	Indicates UDP for the internet address family.
	SOCK_DGRAM	Connectionless datagram socket . Use UDP for the internet address family.
	SOCK_STREAM	Connection based byte streams. Use TCP for the internet address family.
	SOCKET_CNXN_IN_PROGRESS	Socket connection state.
	SOCKET_DISCONNECTED	Socket disconnected
	SOCKET_ERROR	Socket error

Structures

	Name	Description
	in_addr	in_addr structure
	sockaddr	generic address structure for all address families
	sockaddr_in	In the Internet address family
	BERKELEY_MODULE_CONFIG	Berkeley API module configuration structure

Types

	Name	Description
	SOCKADDR	generic address structure for all address families
	SOCKADDR_IN	In the Internet address family
	SOCKET	Socket descriptor

5.8.6 DHCP TCP/IP Stack Library

5.8.6.1 Introduction

Dynamic Host Configuration Protocol (DHCP) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the DHCP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The DHCP client module will allow your application to dynamically obtain an IP address from a DHCP server on the same network. Doing this will reset the IP address, subnet mask, gateway address, and some other configuration parameters in your AppConfig structure.

5.8.6.2 Release Notes

MPLAB Harmony Version: v0.70b **DHCP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.6.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.6.4 Using the Library

This topic describes the basic architecture of the DHCP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [dhcp.h](#)

The interface to the DHCP TCP/IP Stack library is defined in the "[dhcp.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the DHCP TCP/IP Stack library should include "tcpip.h".

Library File:

The DHCP TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

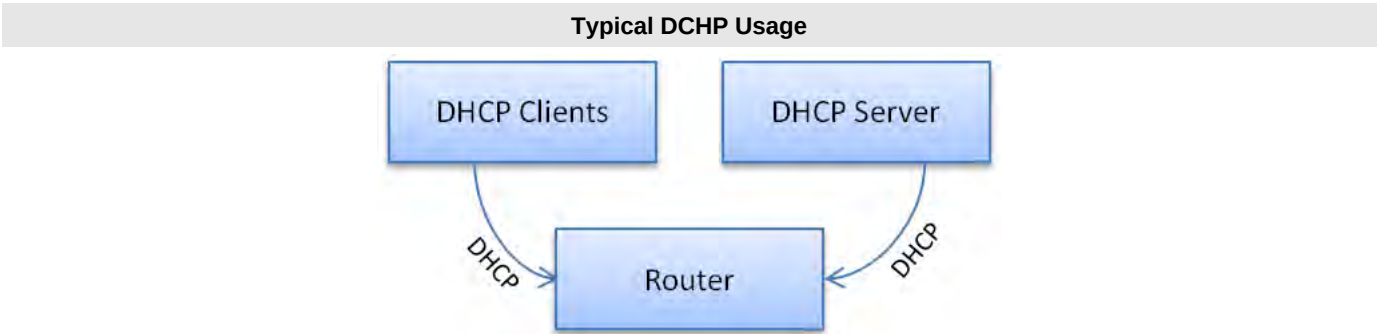
5.8.6.4.1 Abstraction Model

This library provides the API of the DHCP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

DHCP Software Abstraction Block Diagram

This module provides software abstraction of the DHCP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.



5.8.6.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the DHCP module.

Library Interface Section	Description
Configuration Functions	Routines for Configuring DHCP
Status Functions	Routines for Obtaining DHCP Status Information
Data Types and Constants	This section provides various definitions describing This API

5.8.6.4.3 How the Library Works

To use DHCP, include the files dhcp.c, dhcps.c, and [dhcp.h](#) in your project, and add or uncomment the definition "#define TCPIP_STACK_USE_DHCP_CLIENT" to tcpip_config.h. The TCP/IP stack also includes a simple DHCP server that can supply an IP address to one DHCP client. To enable this functionality, add the macro "#define TCPIP_STACK_USE_DHCP_SERVER" to tcpip_config.h.

5.8.6.5 Configuring the Library

Macros

	Name	Description
	DHCP_CLIENT_ENABLED	enable/disable the DHCP client at stack start up
	DHCP_CLIENT_PORT	UDP client local port for DHCP Client transactions
	DHCP_SERVER_PORT	UDP remote port for DHCP Server
	DHCP_TASK_TICK_RATE	5 ms default rate
	DHCP_TIMEOUT	Defines how long to wait before a DHCP request times out, seconds

Description

The configuration of the DHCP TCP/IP Stack is based on the file `sys_config.h`

This header file contains the configuration selection for the DHCP TCP/IP Stack. Based on the selections made, the DHCP TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the DHCP TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.6.5.1 DHCP_CLIENT_ENABLED Macro**C**

```
#define DHCP_CLIENT_ENABLED 1
```

Description

enable/disable the DHCP client at stack start up

Remarks

the interface initialization setting in `TCPIP_NETWORK_CONFIG` takes precedence!

5.8.6.5.2 DHCP_CLIENT_PORT Macro**C**

```
#define DHCP_CLIENT_PORT (68u)
```

Description

UDP client local port for DHCP Client transactions

5.8.6.5.3 DHCP_SERVER_PORT Macro**C**

```
#define DHCP_SERVER_PORT (67u)
```

Description

UDP remote port for DHCP Server

5.8.6.5.4 DHCP_TASK_TICK_RATE Macro**C**

```
#define DHCP_TASK_TICK_RATE (5)    // 5 ms default rate
```

Description

5 ms default rate

5.8.6.5.5 DHCP_TIMEOUT Macro**C**

```
#define DHCP_TIMEOUT (2u1)
```

Description

Defines how long to wait before a DHCP request times out, seconds

5.8.6.6 Building the Library

This section list the files that are available in the \src of the DHCP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.6.7 Library Interface

Data Types and Constants

	Name	Description
	DHCP_MODULE_CONFIG	DHCP Module Configuration Typedef
	TCPIP_DHCP_EVENT_HANDLER	DHCP Event Handler
	TCPIP_DHCP_EVENT_TYPE	DHCP Event Type
	TCPIP_DHCP_HANDLE	DHCP Handle

Configuration Functions

	Name	Description
	TCPIP_DHCP_Disable	Disables the DHCP Client for the specified interface.
	TCPIP_DHCP_Enable	Enables the DHCP client for the specified interface.
	TCPIP_DHCP_HandlerDeRegister	DHCP De-Register Handler
	TCPIP_DHCP_HandlerRegister	DHCP Register Handler
	TCPIP_DHCP_Renew	Renews the DHCP lease for the specified interface.
	TCPIP_DHCP_Request	Requests the supplied IPv4 address from a DHCP server.
	TCPIP_DHCP_RequestTimeoutSet	DHCP Set Request Timeout

Status Functions

	Name	Description
	TCPIP_DHCP_IsBound	Determins if the DHCP client has an IP address lease on the specified interface.
	TCPIP_DHCP_IsEnabled	Determins if the DHCP client is enabled on the specified interface.
	TCPIP_DHCP_IsServerDetected	Determins if the DHCP client on the specified interface has seen a DHCP server.

Description

This section describes the Application Programming Interface (API) functions of the DHCP TCP/IP Stack.

Refer to each section for a detailed description.

5.8.6.7.1 Configuration Functions

5.8.6.7.1.1 TCPIP_DHCP_Disable Function

C

```
bool TCPIP_DHCP_Disable(
    TCPIP_NET_HANDLE hNet
);
```

Description

Disables the DHCP client for the specified interface.

Preconditions

None

Parameters

Parameters	Description
pNetIf	Interface to disable the DHCP client on.

Returns

true if success false otherwise

Remarks

When the interface continues using its old configuration, it is possible that the lease may expire and the DHCP server provide the IP to another client. The application should not request the keeping of the old lease unless there is no danger of conflict.

5.8.6.7.1.2 TCPIP_DHCP_Enable Function**C**

```
bool TCPIP_DHCP_Enable(  
    TCPIP_NET_HANDLE hNet  
);
```

Description

Enables the DHCP client for the specified interface, if it is disabled. If it is already enabled, nothing is done.

Preconditions

None

Parameters

Parameters	Description
hNet	Interface to enable the DHCP client on.

Returns

true if success false otherwise

5.8.6.7.1.3 TCPIP_DHCP_HandlerDeRegister Function**C**

```
bool TCPIP_DHCP_HandlerDeRegister(  
    TCPIP_DHCP_HANDLE hDhcp  
);
```

Description

deregister the event handler returns true or false if no such handler registered

5.8.6.7.1.4 TCPIP_DHCP_HandlerRegister Function**C**

```
TCPIP_DHCP_HANDLE TCPIP_DHCP_HandlerRegister(  
    TCPIP_NET_HANDLE hNet,  
    TCPIP_DHCP_EVENT_HANDLER handler,
```

```
    const void* hParam  
);
```

Description

Register an DHCP event handler. Use hNet == 0 to register on all interfaces available. Returns a valid handle if the call succeeds, or a null handle if the call failed. Function has to be called after the DHCP is initialized. The hParam is passed by the client and will be used by the DHCP when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

5.8.6.7.1.5 TCPIP_DHCP_Renew Function**C**

```
bool TCPIP_DHCP_Renew(  
    TCPIP_NET_HANDLE hNet  
);
```

Description

Tries to contact the server and renew the DHCP lease for the specified interface. The interface should have the DHCP enabled and in bound state for this call to succeed.

Preconditions

DHCP enabled, a valid lease

Parameters

Parameters	Description
hNet	Interface to renew the DHCP lease on.

Returns

true if success false otherwise

5.8.6.7.1.6 TCPIP_DHCP_Request Function**C**

```
bool TCPIP_DHCP_Request(  
    TCPIP_NET_HANDLE hNet,  
    IPV4_ADDR reqAddress  
);
```

Description

If the DHCP client is not enabled on that interface, this call will first try to enable it. If this succeeds or the DHCP client was already enabled, the following steps are taken: The DHCP client probes the DHCP server and requests the supplied IPv4 address as a valid lease for the specified interface. If the server acknowledges the request, then this is the new IPv4 address of the interface. If the DHCP server rejects the request, then the whole DHCP process is resumed starting with the DHCP Discovery phase.

Preconditions

DHCP enabled, a valid lease

Parameters

Parameters	Description
hNet	Interface to renew the DHCP lease on.

Returns

true if success false otherwise

Remarks

The requested IPv4 address should be a previous lease that was granted to the host. This call should be used when the host is restarting.

5.8.6.7.1.7 TCPIP_DHCP_RequestTimeoutSet Function

C

```
bool TCPIP_DHCP_RequestTimeoutSet(
    TCPIP_NET_HANDLE hNet,
    int tmo
);
```

Description

Adjust the DHCP timeout. How long to wait before a DHCP request times out, seconds.

5.8.6.7.2 Status Functions

5.8.6.7.2.1 TCPIP_DHCP_IsBound Function

C

```
bool TCPIP_DHCP_IsBound(
    TCPIP_NET_HANDLE hNet
);
```

Description

Determines if the DHCP client has an IP address lease on the specified interface.

Preconditions

None

Parameters

Parameters	Description
hNet	Interface to query

Returns

true - DHCP client has obtained an IP address lease (and likely other parameters) and these values are currently being used.
false - No IP address is currently leased

5.8.6.7.2.2 TCPIP_DHCP_IsEnabled Function

C

```
bool TCPIP_DHCP_IsEnabled(
    TCPIP_NET_HANDLE hNet
);
```

Description

Determines if the DHCP client is enabled on the specified interface.

Preconditions

None

Parameters

Parameters	Description
hNet	Interface to query.

Returns

None

5.8.6.7.2.3 TCPIP_DHCP_IsServerDetected Function**C**

```
bool TCPIP_DHCP_IsServerDetected(  
    TCPIP_NET_HANDLE hNet  
);
```

Description

Determines if the DHCP client on the specified interface has seen a DHCP server.

Preconditions

None

Parameters

Parameters	Description
hNet	Interface to query.

Returns

true - At least one DHCP server is attached to the specified network interface. false - No DHCP servers are currently detected on the specified network interface.

5.8.6.7.3 Data Types and Constants**5.8.6.7.3.1 DHCP_MODULE_CONFIG Structure****C**

```
typedef struct {  
    bool dhcpEnable;  
    int dhcpTmo;  
    int dhcpCliPort;  
    int dhcpSrvPort;  
} DHCP_MODULE_CONFIG;
```

Description

DHCP Module Configuration

Used in [dhcp_config.h](#)

Members

Members	Description
int dhcpTmo;	timeout to wait for a DHCP request, seconds
int dhcpCliPort;	client port for DHCP client transactions
int dhcpSrvPort;	remote server port for DHCP server messages

5.8.6.7.3.2 TCPIP_DHCP_EVENT_HANDLER Type

C

```
typedef void (* TCPIP_DHCP_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_EVENT_TYPE evType,
const void* param);
```

Description

Type: TCPIP_DHCP_EVENT_HANDLER

Prototype of a DHCP event handler clients can register a handler with the DHCP service. Once an DHCP event occurs the DHCP service will called the registered handler. The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

5.8.6.7.3.3 TCPIP_DHCP_EVENT_TYPE Enumeration

C

```
typedef enum {
    DHCP_EVENT_DISCOVER = 1,
    DHCP_EVENT_BOUND,
    DHCP_EVENT_LEASE_EXPIRED,
    DHCP_EVENT_CONN_LOST,
    DHCP_EVENT_SERVICE_DISABLED
} TCPIP_DHCP_EVENT_TYPE;
```

Description

Enumeration: TCPIP_DHCP_EVENT_TYPE

None

Members

Members	Description
DHCP_EVENT_DISCOVER = 1	DHCP cycle started
DHCP_EVENT_BOUND	DHCP lease obtained
DHCP_EVENT_LEASE_EXPIRED	lease expired
DHCP_EVENT_CONN_LOST	connection to the DHCP server lost, reverted to the defaultIP address
DHCP_EVENT_SERVICE_DISABLED	DHCP service disabled, reverted to the defaultIP address

5.8.6.7.3.4 TCPIP_DHCP_HANDLE Type

C

```
typedef const void* TCPIP_DHCP_HANDLE;
```

Description

Type: TCPIP_DHCP_HANDLE

A handle that a client can use after the event handler has been registered.

5.8.6.8 Files

Files

Name	Description
dhcp.h	Dynamic Host Configuration Protocol (DHCP) Client API

dhcp_config.h	DHCP configuration file

Description**5.8.6.8.1 dhcp.h**

Provides automatic IP address, subnet mask, gateway address, DNS server address, and other configuration parameters on DHCP enabled networks.

- Reference: RFC 2131, 2132

Enumerations

	Name	Description
	TCPIP_DHCP_EVENT_TYPE	DHCP Event Type

Functions

	Name	Description
	TCPIP_DHCP_Disable	Disables the DHCP Client for the specified interface.
	TCPIP_DHCP_Enable	Enables the DHCP client for the specified interface.
	TCPIP_DHCP_HandlerDeRegister	DHCP De-Register Handler
	TCPIP_DHCP_HandlerRegister	DHCP Register Handler
	TCPIP_DHCP_IsBound	Determines if the DHCP client has an IP address lease on the specified interface.
	TCPIP_DHCP_IsEnabled	Determines if the DHCP client is enabled on the specified interface.
	TCPIP_DHCP_IsServerDetected	Determines if the DHCP client on the specified interface has seen a DHCP server.
	TCPIP_DHCP_Renew	Renews the DHCP lease for the specified interface.
	TCPIP_DHCP_Request	Requests the supplied IPv4 address from a DHCP server.
	TCPIP_DHCP_RequestTimeoutSet	DHCP Set Request Timeout

Structures

	Name	Description
	DHCP_MODULE_CONFIG	DHCP Module Configuration Typedef

Types

	Name	Description
	TCPIP_DHCP_EVENT_HANDLER	DHCP Event Handler
	TCPIP_DHCP_HANDLE	DHCP Handle

File Name

dhcp.h

Company

Microchip Technology Incorporated

5.8.6.8.2 dhcp_config.h

Dynamic Host Configuration Protocol (DHCP) Configuration file

This file contains the DHCP module configuration options

Macros

	Name	Description
	<u>DHCP_CLIENT_ENABLED</u>	enable/disable the DHCP client at stack start up
	<u>DHCP_CLIENT_PORT</u>	UDP client local port for DHCP Client transactions
	<u>DHCP_SERVER_PORT</u>	UDP remote port for DHCP Server
	<u>DHCP_TASK_TICK_RATE</u>	5 ms default rate
	<u>DHCP_TIMEOUT</u>	Defines how long to wait before a DHCP request times out, seconds

5.8.7 DHCP Server TCP/IP Stack Library

5.8.7.1 Introduction

DHCP Server TCP/IP Stack Library
for
Microchip Microcontrollers

This library provides the API of the DHCP server TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The DHCP Server module is used to assign the IP address to the DHCP client from the configured IP address database. When the server receives a request from a client, the DHCP server determines the network to which the DHCP client is connected, and then allocates an IP address that is appropriate for the client, and sends configuration information appropriate for that client. DHCP servers typically grant IP addresses to clients only for a limited interval. DHCP clients are responsible for renewing their IP address before that interval has expired, and must stop using the address once the interval has expired, if they have not been able to renew it.

5.8.7.2 Release Notes

MPLAB Harmony Version: v0.70b **DHCP Server TCP/IP Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.7.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.7.4 Using the Library

This topic describes the basic architecture of the DHCP Server TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [dhcps.h](#)

The interface to the DHCP Server TCP/IP Stack library is defined in the "[dhcps.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the DHCP Server TCP/IP Stack library should include "tcpip.h".

Library File:

The DHCP Server TCP/IP Stack library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.7.4.1 Abstraction Model

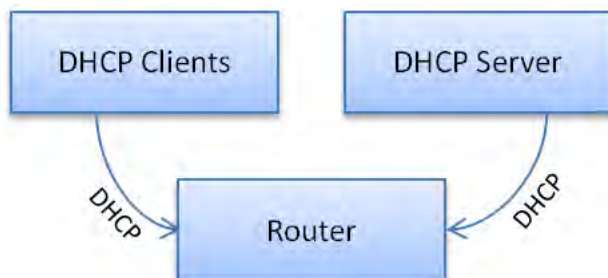
This library provides the API of the DHCP Server TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

DHCP Server Software Abstraction Block Diagram

This module provides software abstraction of the DHCP Server module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

Typical DCHP Usage



5.8.7.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the DHCP Server module.

Library Interface Section	Description
Configuration Functions	Routines for Configuring DHCP server
Status Functions	Routines for Obtaining DHCP Status Information
Data Types and Constants	This section provides various definitions describing This API

5.8.7.4.3 How the Library Works

To use DHCP Server, include the files `dhcps.c`, `dhcp.c`, and [dhcp.h](#) in your project, and add or uncomment the definition `"#define TCPIP_STACK_USE_DHCP_SERVER"` to `tcip_config.h`.

5.8.7.5 Configuring the Library

Macros

	Name	Description
	DHCP_SERVER_GATEWAY_ADDRESS	as we are in same network, Gateway address is same as server IP address.
	DHCP_SERVER_IP_ADDRESS	First three octet should match to the TP address range for Server address and the begining address should be reserved for server address
	DHCP_SERVER_NETMASK_ADDRESS	Net mask value
	DHCP_SERVER_PRIMARY_DNS_ADDRESS	DNS Primary address
	DHCP_SERVER_SECONDARY_DNS_ADDRESS	DNS Secondary address
	DHCP_IP_ADDRESS_RANGE_START	Address range is starting from 100, because the from 1 to 100 is reserved. Start address(192.168.100.1) should be used as server address
	DHCP_LEASE_DURATION	timeout for a solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been referenced again
	DHCP_LEASE_ENTRIES_DEFAULT	number of entries in the cache default number of entries per interface
	DHCP_LEASE_REMOVED_BEFORE_ACK	The entry should be removed from the entry if there is no REQUEST after OFFER
	DHCP_LEASE_SOLVED_ENTRY_TMO	timeout for a solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been referenced again
	DHCP_TASK_PROCESS_RATE	DHCP task processing rate, in seconds the DHCP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc.

Description

The configuration of the DHCP Server TCP/IP Stack is based on the file [dhcps_config.h](#)

This header file contains the configuration selection for the DHCP Server TCP/IP Stack. Based on the selections made, the DHCP Server TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the DHCP Server TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.7.5.1 DHCP_SERVER_GATEWAY_ADDRESS Macro

C

```
#define DHCP_SERVER_GATEWAY_ADDRESS "192.168.100.1" // as we are in same network, Gateway  
address is same as server IP address.
```

Description

as we are in same network, Gateway address is same as server IP address.

5.8.7.5.2 DHCP_SERVER_IP_ADDRESS Macro

C

```
#define DHCP_SERVER_IP_ADDRESS "192.168.100.1" // First three octet should match to the TP  
address range for Server address
```

Description

First three octet should match to the TP address range for Server address and the beginning address should be reserved for server address

5.8.7.5.3 DHCP_SERVER_NETMASK_ADDRESS Macro

C

```
#define DHCP_SERVER_NETMASK_ADDRESS "255.255.255.0" // Net mask value
```

Description

Net mask value

5.8.7.5.4 DHCP_SERVER_PRIMARY_DNS_ADDRESS Macro

C

```
#define DHCP_SERVER_PRIMARY_DNS_ADDRESS "192.168.100.1" // DNS Primary address
```

Description

DNS Primary address

5.8.7.5.5 DHCP_SERVER_SECONDARY_DNS_ADDRESS Macro

C

```
#define DHCP_SERVER_SECONDARY_DNS_ADDRESS "192.168.100.1" // DNS Secondary address
```

Description

DNS Secondary address

5.8.7.5.6 DHCP_IP_ADDRESS_RANGE_START Macro

C

```
#define DHCP_IP_ADDRESS_RANGE_START "192.168.100.100" // Address range is starting from 100,  
because the from 1 to 100 is reserved. Start address(192.168.100.1)
```

Description

Address range is starting from 100, because the from 1 to 100 is reserved. Start address(192.168.100.1) should be used as server address

5.8.7.5.7 DHCP_LEASE_DURATION Macro

C

```
#define DHCP_LEASE_DURATION DHCP_LEASE_SOLVED_ENTRY_TMO
```

Description

timeout for a solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been referenced again

5.8.7.5.8 DHCP_LEASE_ENTRIES_DEFAULT Macro

C

```
#define DHCP_LEASE_ENTRIES_DEFAULT 15
```

Description

number of entries in the cache default number of entries per interface

5.8.7.5.9 DHCP_LEASE_REMOVED_BEFORE_ACK Macro

C

```
#define DHCP_LEASE_REMOVED_BEFORE_ACK (5)
```

Description

The entry should be removed from the entry if there is no REQUEST after OFFER

5.8.7.5.10 DHCP_LEASE_SOLVED_ENTRY_TMO Macro

C

```
#define DHCP_LEASE_SOLVED_ENTRY_TMO (20 * 60)
```

Description

timeout for a solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been referenced again

5.8.7.5.11 DHCP_TASK_PROCESS_RATE Macro

C

```
#define DHCP_TASK_PROCESS_RATE (1)
```

Description

DHCPS task processing rate, in seconds the DHCPS module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc.

5.8.7.6 Building the Library

This section list the files that are available in the \src of the DHCP server. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.7.7 Library Interface

Data Types and Constants

	Name	Description
	DHCP_SERVER_POOL_ENTRY_TYPE	DHCP server pool types are used to get the leased IP address details.
	DHCP_LEASE_ENTRY	listing of the leases
	DHCP_LEASE_HANDLE	Lease details which is used by the CLI command prompt.
	DHCP_MODULE_CONFIG	DHCP server configuration details

Configuration Functions

	Name	Description
	TCPIP_DHCP_Disable	Disables the DHCP Server for the specified interface.
	TCPIP_DHCP_Enable	Enables the DHCP Server for the specified interface.
	TCPIP_DHCP_GetPoolEntries	Get all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCP_LeaseEntryGet	returns a lease entry and allows iteration through the whole list of leases if leaseHandle it starts from the begining returns a non-zero DHCP_LEASE_HANDLE to be used in the subsequent calls or 0 if end of list or wrong interface, or DHCP server not running on that interface
	TCPIP_DHCP_RemovePoolEntries	Removes all the entries or only used entries of a certain type belonging to a network interface.

Status Functions

	Name	Description
	TCPIP_DHCP_IsEnabled	Determines if the DHCP Server is enabled on the specified interface.

Description

This section describes the Application Programming Interface (API) functions of the DHCP Server TCP/IP Stack.

Refer to each section for a detailed description.

5.8.7.7.1 Configuration Functions

5.8.7.7.1.1 TCPIP_DHCP_Disable Function

C

```
bool TCPIP_DHCP_Disable(
    TCPIP_NET_HANDLE hNet
```

```
);
```

Description

Disables the DHCP Server for the specified interface.

Preconditions

None

Parameters

Parameters	Description
pNetIf	Interface to disable the DHCP Server on.

Returns

true if success false otherwise

Remarks

When the interface continues using its old configuration, it is possible that the lease may take sometime to expire. And The communication will be there till it is not expired. Lease time is configured in [dhcps_config.h](#).

5.8.7.7.1.2 TCPIP_DHCPS_Enable Function

C

```
bool TCPIP_DHCPS_Enable(  
    TCPIP_NET_HANDLE hNet  
);
```

Description

Enables the DHCP Server for the specified interface, if it is disabled. If it is already enabled, nothing is done.

Preconditions

None

Parameters

Parameters	Description
hNet	Interface to enable the DHCP Server on.

Returns

true if success false otherwise

5.8.7.7.1.3 TCPIP_DHCPS_GetPoolEntries Function

C

```
int TCPIP_DHCPS_GetPoolEntries(  
    TCPIP_NET_HANDLE netH,  
    DHCP_SERVER_POOL_ENTRY_TYPE type  
);
```

Description

This API is used to get the DHCP server entries from the pool as per [DHCP_SERVER_POOL_ENTRY_TYPE](#).

Preconditions

DHCP server module should have been initialized

Parameters

Parameters	Description
hNet	Interface handle to use
type	type of entries to remove:
valid types	DHCP_SERVER_POOL_ENTRY_ALL, DHCP_SERVER_POOL_ENTRY_IN_USE

Returns

true or false

Remarks

None

Example

None

5.8.7.7.1.4 TCPIP_DHCPs_LeaseEntryGet Function

C

```
DHCPS_LEASE_HANDLE TCPIP_DHCPs_LeaseEntryGet(  
    TCPIP_NET_HANDLE netH,  
    DHCPS_LEASE_ENTRY* pLeaseEntry,  
    DHCPS_LEASE_HANDLE leaseHandle  
);
```

Description

returns a lease entry and allows iteration through the whole list of leases if leaseHandle it starts from the beginning returns a non-zero [DHCPS_LEASE_HANDLE](#) to be used in the subsequent calls or 0 if end of list or wrong interface, or DHCP server not running on that interface

5.8.7.7.1.5 TCPIP_DHCPs_RemovePoolEntries Function

C

```
bool TCPIP_DHCPs_RemovePoolEntries(  
    TCPIP_NET_HANDLE netH,  
    DHCP_SERVER_POOL_ENTRY_TYPE type  
);
```

Description

This API is used to remove the DHCP server entries from the pool as per [DHCP_SERVER_POOL_ENTRY_TYPE](#).

Preconditions

DHCP server module should have been initialized

Parameters

Parameters	Description
hNet	Interface handle to use
type	type of entries to remove:
valid types	DHCP_SERVER_POOL_ENTRY_ALL, DHCP_SERVER_POOL_ENTRY_IN_USE

Returns

true or false

Remarks

None

Example

None

5.8.7.7.2 Status Functions

5.8.7.7.2.1 TCPIP_DHCPs_IsEnabled Function

C

```
bool TCPIP_DHCPs_IsEnabled(  
    TCPIP_NET_HANDLE hNet  
);
```

Description

Determines if the DHCP Server is enabled on the specified interface.

Preconditions

None

Parameters

Parameters	Description
hNet	Interface to query.

Returns

None

5.8.7.7.3 Data Types and Constants

5.8.7.7.3.1 DHCP_SERVER_POOL_ENTRY_TYPE Enumeration

C

```
typedef enum {  
    DHCP_SERVER_POOL_ENTRY_ALL,  
    DHCP_SERVER_POOL_ENTRY_IN_USE  
} DHCP_SERVER_POOL_ENTRY_TYPE;
```

Description

DHCP server pool types are used to get the leased IP address details.

Members

Members	Description
DHCP_SERVER_POOL_ENTRY_ALL	Get or Remove all the Leased address . There might be a address which may not be used.
DHCP_SERVER_POOL_ENTRY_IN_USE	Get or remove only Leased IP address.

5.8.7.7.3.2 DHCPs_LEASE_ENTRY Structure

C

```
typedef struct {
```

```
TCPIP_MAC_ADDR hwAddr;  
IPV4_ADDR ipAddress;  
uint32_t leaseTime;  
} DHCP_LEASE_ENTRY;
```

Description

listing of the leases

Members

Members	Description
TCPIP_MAC_ADDR hwAddr;	Client MAC address
IPV4_ADDR ipAddress;	Leased IP address
uint32_t leaseTime;	Lease period

5.8.7.7.3.3 DHCP_LEASE_HANDLE Type

C

```
typedef const void* DHCP_LEASE_HANDLE;
```

Description

Lease details which is used by the CLI command prompt.

5.8.7.7.3.4 DHCP_MODULE_CONFIG Structure

C

```
typedef struct {  
    bool enabled;  
    bool deleteOldLease;  
    size_t leaseEntries;  
    int entrySolvedTmo;  
    char * startIpAddressRange;  
} DHCP_MODULE_CONFIG;
```

Description

DHCP server configuration details

Members

Members	Description
bool enabled;	enable DHCP server
bool deleteOldLease;	delete old cache if still in place, specific DHCP params
size_t leaseEntries;	max number of lease entries
int entrySolvedTmo;	solved entry removed after this tmo if not referenced - seconds
char * startIpAddressRange;	start value of the IP address for DHCP clients.

5.8.7.8 Files

Files

Name	Description
dhcps.h	Dynamic Host Configuration Protocol(DHCP) Server APIs.
dhcps_config.h	DHCPS configuration file

Description**5.8.7.8.1 dhcps.h**

The DHCP server permanently assigns a free IP address to a requesting client from the range defined in the [dhcps_config.h](#) file.

- Reference: RFC 2131, 2132

Enumerations

	Name	Description
	DHCP_SERVER_POOL_ENTRY_TYPE	DHCP server pool types are used to get the leased IP address details.

Functions

	Name	Description
	TCPIP_DHCP_DISABLE	Disables the DHCP Server for the specified interface.
	TCPIP_DHCP_ENABLE	Enables the DHCP Server for the specified interface.
	TCPIP_DHCP_GET_POOL_ENTRIES	Get all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCP_IS_ENABLED	Determines if the DHCP Server is enabled on the specified interface.
	TCPIP_DHCP_LEASE_ENTRY_GET	returns a lease entry and allows iteration through the whole list of leases if leaseHandle it starts from the beginning returns a non-zero DHCP_LEASE_HANDLE to be used in the subsequent calls or 0 if end of list or wrong interface, or DHCP server not running on that interface
	TCPIP_DHCP_REMOVE_POOL_ENTRIES	Removes all the entries or only used entries of a certain type belonging to a network interface.

Structures

	Name	Description
	DHCP_LEASE_ENTRY	listing of the leases
	DHCP_MODULE_CONFIG	DHCP server configuration details

Types

	Name	Description
	DHCP_LEASE_HANDLE	Lease details which is used by the CLI command prompt.

File Name

dhcps.h

Company

Microchip Technology Incorporated

5.8.7.8.2 dhcps_config.h

Dynamic Host Configuration Protocol (DHCP) Configuration file

This file contains the DHCP module configuration options

Macros

	Name	Description
	<u>DHCP_SERVER_GATEWAY_ADDRESS</u>	as we are in same network, Gateway address is same as server IP address.
	<u>DHCP_SERVER_IP_ADDRESS</u>	First three octet should match to the TP address range for Server address and the begining address should be reserved for server address
	<u>DHCP_SERVER_NETMASK_ADDRESS</u>	Net mask value
	<u>DHCP_SERVER_PRIMARY_DNS_ADDRESS</u>	DNS Primary address
	<u>DHCP_SERVER_SECONDARY_DNS_ADDRESS</u>	DNS Secondary address
	<u>DHCPS_IP_ADDRESS_RANGE_START</u>	Address range is starting from 100, because the from 1 to 100 is reserved. Start address(192.168.100.1) should be used as server address
	<u>DHCPS_LEASE_DURATION</u>	timeout for a solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been referenced again
	<u>DHCPS_LEASE_ENTRIES_DEFAULT</u>	number of entries in the cache default number of entries per interface
	<u>DHCPS_LEASE_REMOVED_BEFORE_ACK</u>	The entry should be removed from the entry if there is no REQUEST after OFFER
	<u>DHCPS_LEASE_SOLVED_ENTRY_TMO</u>	timeout for a solved entry in the cache, in seconds the entry will be removed if the tmo elapsed and the entry has not been referenced again
	<u>DHCPS_TASK_PROCESS_RATE</u>	DHCPS task processing rate, in seconds the DHCPS module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc.

5.8.8 DNS TCP/IP Stack Library

5.8.8.1 Introduction

Domain Name System (DNS) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the DNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The Domain Name Service associates host names (such as www.microchip.com) with IP addresses (such as 10.0.54.2). The DNS Client module provides DNS resolution capabilities to the stack.

TCP applications do not need to use the DNS module. Any necessary DNS operations can be handled by the TCPOpen function. Applications built using UDP may need to use DNS when the IP address of the remote server is unknown.

5.8.8.2 Release Notes

MPLAB Harmony Version: v0.70b **DNS TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.8.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.8.4 Using the Library

This topic describes the basic architecture of the DNS TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [dns.h](#)

The interface to the DNS TCP/IP Stack library is defined in the "[dns.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the DNS TCP/IP Stack library should include "tcpip.h".

Library File:

The DNS TCP/IP Stack library archive (.a) file is installed with MPLAB Harmony.

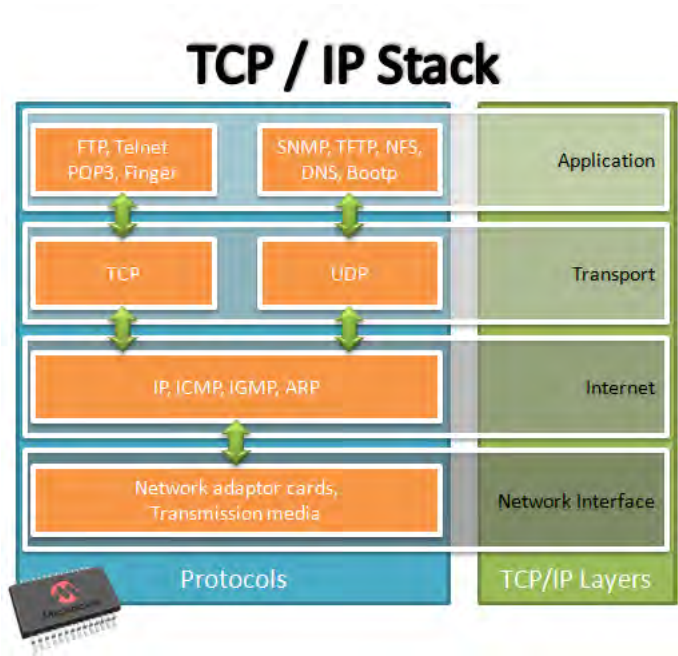
Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.8.4.1 Abstraction Model

This library provides the API of the DNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

DNS is part of the Application Layer:



DNS resolution operations follow a simple state machine, as indicated in the diagram below.

5.8.8.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the DNS module.

Library Interface Section	Description
General Functions	This section provides general interface routines to DNS
Data Types and Constants	This section provides various definitions describing this API

5.8.8.5 Configuring the Library

Macros

	Name	Description
	DNS_CLIENT_ARP_TMO	When the DNS Client performs an ARP request for the DNS Server this is the elapsed time after which a ARP resolution is considered to have timed out In seconds
	DNS_CLIENT_OPEN_TMO	The DNS Client performs an UDP socket open request to communicate with the DNS Server This is the elapsed time after which an open request is considered to have timed out In seconds
	DNS_CLIENT_PORT	Default port for DNS resolutions
	DNS_CLIENT_SERVER_TMO	When the DNS Client connected to the DNS Server this is the elapsed time after which an the communication is considered to have timed failed if there was no reply from the server In seconds

	DNS_CLIENT_TASK_PROCESS_RATE	DNS Client task processing rate, in milli-seconds the DNS Client module will process a timer event with this rate for processing its own state machine, etc.
	DNS_CLIENT_VERSION_NO	DNS Client version Depending on the version of the DNS implementation the DNS task process will be performed differently

Description

The configuration of the DNS TCP/IP Stack is based on the file `tcip_config.h`

This header file contains the configuration selection for the DNS TCP/IP Stack. Based on the selections made, the DNS TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the DNS TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.8.5.1 DNS_CLIENT_ARP_TMO Macro

C

```
#define DNS_CLIENT_ARP_TMO (1)
```

Description

When the DNS Client performs an ARP request for the DNS Server this is the elapsed time after which a ARP resolution is considered to have timed out In seconds

5.8.8.5.2 DNS_CLIENT_OPEN_TMO Macro

C

```
#define DNS_CLIENT_OPEN_TMO (1)
```

Description

The DNS Client performs an UDP [socket](#) open request to communicate with the DNS Server This is the elapsed time after which an open request is considered to have timed out In seconds

5.8.8.5.3 DNS_CLIENT_PORT Macro

C

```
#define DNS_CLIENT_PORT 53
```

Description

Default port for DNS resolutions

5.8.8.5.4 DNS_CLIENT_SERVER_TMO Macro

C

```
#define DNS_CLIENT_SERVER_TMO (1)
```

Description

When the DNS Client connected to the DNS Server this is the elapsed time after which an the communication is considered to have timed failed if there was no reply from the server In seconds

5.8.8.5.5 DNS_CLIENT_TASK_PROCESS_RATE Macro

C

```
#define DNS_CLIENT_TASK_PROCESS_RATE (500)
```

Description

DNS Client task processing rate, in milli-seconds the DNS Client module will process a timer event with this rate for processing its own state machine, etc.

5.8.8.5.6 DNS_CLIENT_VERSION_NO Macro

C

```
#define DNS_CLIENT_VERSION_NO 1
```

Description

DNS Client version Depending on the version of the DNS implementation the DNS task process will be performed differently

5.8.8.6 Building the Library

This section list the files that are available in the \src of the DNS driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.8.7 Library Interface

Data Types and Constants

	Name	Description
	DNS_CLIENT_MODULE_CONFIG	Provides a placeholder for DNS client configuration.
	DNS_RESOLVE_TYPE	This enumeration provides the RecordType argument for TCPIP_DNS_Resolve .
	DNS_RESULT	This enumeration provides result codes for various DNS operations.
	DNS_SERVER_MODULE_CONFIG	Provides a placeholder for DNS server configuration.

General Functions

	Name	Description
	TCPIP_DNS_IsResolved	Determines if the DNS resolution is complete and provides the IP.
	TCPIP_DNS_Resolve	Begins resolution of an address.
	TCPIP_DNS_UsageBegin	Claims access to the DNS module.
	TCPIP_DNS_UsageEnd	Releases control of the DNS module.

Description

This section describes the Application Programming Interface (API) functions of the DNS TCP/IP Stack.

Refer to each section for a detailed description.

5.8.8.7.1 General Functions

5.8.8.7.1.1 TCPIP_DNS_IsResolved Function

C

```
DNS_RESULT TCPIP_DNS_IsResolved(  
    const char* HostName,  
    void* HostIP  
);
```

Description

Call this function to determine if the DNS resolution of an address has been completed. If so, the resolved address will be provided in HostIP.

Preconditions

[TCPIP_DNS_Resolve](#) has been called.

Parameters

Parameters	Description
Hostname	A pointer to the null terminated string specifying the host for which to resolve an IP.
HostIP	A pointer to an IPV4_ADDR structure in which to store the resolved IP address once resolution is complete.

Return Values

Return Values	Description
DNS_RES_OK	The DNS client has obtained an IP HostIP will contain the resolved address.
DNS_RES_PENDING	The resolution process is still in progress.
DNS_RES_SERVER_TMO	DNS server timed out
DNS_RES_NO_ENTRY	no such entry to be resolved exists

Remarks

None.

5.8.8.7.1.2 TCPIP_DNS_Resolve Function

C

```
DNS_RESULT TCPIP_DNS_Resolve(  
    const char* HostName,  
    DNS_RESOLVE_TYPE Type  
);
```

Description

This function attempts to resolve a host name to an IP address. When called, it starts the DNS state machine. Call [TCPIP_DNS_IsResolved](#) repeatedly to determine if the resolution is complete.

Only one DNS resolution may be executed at a time. The Hostname must not be modified in memory until the resolution is complete.

Preconditions

[TCPIP_DNS_UsageBegin](#) returned DNS_RES_OK on a previous call.

Parameters

Parameters	Description
Hostname	A pointer to the null terminated string specifying the host for which to resolve an IP.
RecordType	DNS_TYPE_A or DNS_TYPE_MX depending on what type of record resolution is desired.

Returns

DNS_RES_OK

Remarks

This function requires access to one UDP [socket](#). If none are available, [UDP_MAX_SOCKETS](#) may need to be increased.

5.8.8.7.1.3 TCPIP_DNS_UsageBegin Function**C**

```
DNS_RESULT TCPIP_DNS_UsageBegin(  
    TCPIP_NET_HANDLE neth  
);
```

Description

This function acts as a semaphore to obtain usage of the DNS module. Call this function and ensure that it returns DNS_RES_OK before calling any other DNS APIs. Call [TCPIP_DNS_UsageEnd](#) when this application no longer needs the DNS module so that other applications may make use of it.

Preconditions

Stack is initialized.

Parameters

Parameters	Description
neth	interface to use If 0, a default interface will be selected

Return Values

Return Values	Description
DNS_RES_OK	the calling application has successfully taken ownership of the DNS module
DNS_RES_BUSY	The DNS module is currently in use. Yield to the stack and attempt this call again later.

Remarks

Ensure that [TCPIP_DNS_UsageEnd](#) is always called once your application has obtained control of the DNS module. If this is not done, the stack will hang for all future applications requiring DNS access.

5.8.8.7.1.4 TCPIP_DNS_UsageEnd Function**C**

```
DNS_RESULT TCPIP_DNS_UsageEnd(  
    TCPIP_NET_HANDLE neth  
);
```

Description

This function acts as a semaphore to release control of the DNS module. Call this function when this application no longer needs the DNS module so that other applications may make use of it.

Preconditions

[TCPIP_DNS_UsageBegin](#) returned DNS_RES_OK on a previous call.

Parameters

Parameters	Description
neth	interface to release Not used.

Return Values

Return Values	Description
DNS_RES_OK	The DNS module successfully released.

Remarks

Ensure that TCPIP_DNS_UsageEnd is always called once your application has obtained control of the DNS module. If this is not done, the stack will hang for all future applications requiring DNS access.

5.8.8.7.2 Data Types and Constants

5.8.8.7.2.1 DNS_CLIENT_MODULE_CONFIG Structure

C

```
typedef struct {  
} DNS_CLIENT_MODULE_CONFIG;
```

Description

DNS_CLIENT_MODULE_CONFIG Structure Typedef
Provides a placeholder for DNS client configuration.

Remarks

None.

5.8.8.7.2.2 DNS_RESOLVE_TYPE Enumeration

C

```
typedef enum {  
    DNS_TYPE_A = 1,  
    DNS_TYPE_MX = 15,  
    DNS_TYPE_AAAA = 28u  
} DNS_RESOLVE_TYPE;
```

Description

DNS_RESOLVE_TYPE Enumeration
This enumeration provides the RecordType argument for [TCPIP_DNS_Resolve](#).

Members

Members	Description
DNS_TYPE_A = 1	Constant for record type in TCPIP_DNS_Resolve . Indicates an A (standard address) record.
DNS_TYPE_MX = 15	Constant for record type in TCPIP_DNS_Resolve . Indicates an MX (mail exchanger) record.

DNS_TYPE_AAAA = 28u	Constant for record type in TCPIP_DNS_Resolve . Indicates a quad-A (IPv6 address) address record.

Remarks

None.

5.8.8.7.2.3 DNS_RESULT Enumeration

C

```
typedef enum {
    DNS_RES_OK = 0,
    DNS_RES_PENDING,
    DNS_RES_NO_ENTRY = -1,
    DNS_RES_CACHE_FULL = -2,
    DNS_RES_OPEN_TMO = -3,
    DNS_RES_SERVER_TMO = -4,
    DNS_RES_NO_SERVICE = -5,
    DNS_RES_NO_INTERFACE = -10,
    DNS_RES_BUSY = -11,
    DNS_RES_ARP_TMO = -12
} DNS_RESULT;
```

Description

DNS_RESULT Enumeration

This enumeration provides result codes for various DNS operations.

Members

Members	Description
DNS_RES_OK = 0	operation succeeded
DNS_RES_PENDING	operation is ongoing
DNS_RES_NO_ENTRY = -1	no such entry exists
DNS_RES_CACHE_FULL = -2	the cache is full and no entry could be
DNS_RES_OPEN_TMO = -3	DNS client couldn't get a socket
DNS_RES_SERVER_TMO = -4	DNS server response tmo
DNS_RES_NO_SERVICE = -5	DNS service not implemented
DNS_RES_NO_INTERFACE = -10	an active/requested interface could not be found
DNS_RES_BUSY = -11	module is in use by other task; retry later
DNS_RES_ARP_TMO = -12	ARP tmo

Remarks

None.

5.8.8.7.2.4 DNS_SERVER_MODULE_CONFIG Structure

C

```
typedef struct {
} DNS_SERVER_MODULE_CONFIG;
```

Description

DNS_SERVER_MODULE_CONFIG Structure Typedef

Provides a placeholder for DNS server configuration.

Remarks

None.

5.8.8.8 Files

Files

Name	Description
dns.h	DNS definitions and interface file
dns_config.h	DNS configuration file

Description

5.8.8.8.1 dns.h

Domain Name System (DNS) client Header file

This source file contains the DNS client module API

Enumerations

	Name	Description
	DNS_RESOLVE_TYPE	This enumeration provides the RecordType argument for TCPIP_DNS_Resolve .
	DNS_RESULT	This enumeration provides result codes for various DNS operations.

Functions

	Name	Description
	TCPIP_DNS_IsResolved	Determines if the DNS resolution is complete and provides the IP.
	TCPIP_DNS_Resolve	Begins resolution of an address.
	TCPIP_DNS_UsageBegin	Claims access to the DNS module.
	TCPIP_DNS_UsageEnd	Releases control of the DNS module.

Structures

	Name	Description
	DNS_CLIENT_MODULE_CONFIG	Provides a placeholder for DNS client configuration.
	DNS_SERVER_MODULE_CONFIG	Provides a placeholder for DNS server configuration.

5.8.8.8.2 dns_config.h

Domain Name Service (DNS) Configuration file

This file contains the DNS module configuration options

Macros

	Name	Description
	DNS_CLIENT_ARP_TMO	When the DNS Client performs an ARP request for the DNS Server this is the elapsed time after which a ARP resolution is considered to have timed out In seconds

	<u>DNS_CLIENT_OPEN_TMO</u>	The DNS Client performs an UDP <u>socket</u> open request to communicate with the DNS Server This is the elapsed time after which an open request is considered to have timed out In seconds
	<u>DNS_CLIENT_PORT</u>	Default port for DNS resolutions
	<u>DNS_CLIENT_SERVER_TMO</u>	When the DNS Client connected to the DNS Server this is the elapsed time after which an the communication is considered to have timed failed if there was no reply from the server In seconds
	<u>DNS_CLIENT_TASK_PROCESS_RATE</u>	DNS Client task processing rate, in milli-seconds the DNS Client module will process a timer event with this rate for processing its own state machine, etc.
	<u>DNS_CLIENT_VERSION_NO</u>	DNS Client version Depending on the version of the DNS implementation the DNS task process will be performed differently

5.8.9 Dynamic DNS TCP/IP Stack Library

5.8.9.1 Introduction

Dynamic Domain Name System (DDNS) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the DDNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service. These services can be used to provide DNS hostname mapping to devices that behind routers, firewalls, and/or on networks that dynamically assign IP addresses.

Note that this only solves one of the two problems for communicating to devices on local subnets from the Internet. While Dynamic DNS can help to locate the device, the router or firewall it sits behind must still properly forward the incoming connection request. This generally requires port forwarding to be configured for the router behind which the device is located.

The Dynamic DNS client supports the popular interface used by DynDNS.org, No-IP.com, and DNS-O-Matic.com.

IMPORTANT: The dynamic DNS services stipulate that updates should be made no more frequently than 10 minutes, and only when the IP address has changed. Updates made more often than that are considered abusive, and may eventually cause your account to be disabled. Production devices that get rebooted frequently may need to store the last known IP in non-volatile memory. You also should not enable this module while testing the rest of your application.

5.8.9.2 Release Notes

MPLAB Harmony Version: v0.70b **DDNS TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.9.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.9.4 Using the Library

This topic describes the basic architecture of the DDNS TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: `dyn_dns.h`

The interface to the DDNS TCP/IP Stack library is defined in the "dyn_dns.h" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the DDNS TCP/IP Stack library should include "tcpip.h".

Library File:

The DDNS TCP/IP Stack library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.9.4.1 Abstraction Model

This library provides the API of the DDNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description**DDNS Software Abstraction Block Diagram**

Note: This diagram was not available at time of release and will be added in a future version of MPLAB Harmony.

5.8.9.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the DDNS module.

Library Interface Section	Description
Functions	Routines for Configuring DDNS
Data Types and Constants	This section provides various definitions describing this API

5.8.9.5 Configuring the Library

Macros

	Name	Description
	DDNS_CHECKIP_SERVER	Default CheckIP server for determining current IP address
	DDNS_DEFAULT_PORT	Default port for CheckIP server

Description

The configuration of the DDNS TCP/IP Stack Library is based on the file `tcpip_config.h`

This header file contains the configuration selection for the DDNS TCP/IP Stack Library. Based on the selections made, the DDNS TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the DDNS TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.9.5.1 DDNS_CHECKIP_SERVER Macro

C

```
#define DDNS_CHECKIP_SERVER (const uint8_t*)"checkip.dyndns.com"
```

Description

Default CheckIP server for determining current IP address

5.8.9.5.2 DDNS_DEFAULT_PORT Macro

C

```
#define DDNS_DEFAULT_PORT (80u)
```

Description

Default port for CheckIP server

5.8.9.6 Building the Library

This section list the files that are available in the `\src` of the DDNS driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.9.7 Library Interface

Data Types and Constants

	Name	Description
	DDNS_MODULE_CONFIG	This is type DDNS_MODULE_CONFIG.
	DDNS_POINTERS	Configuration parameters for the Dynamic DNS Client
	DDNS_SERVICES	Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.
	DDNS_STATUS	Status message for DynDNS client. GOOD and NOCHG are ok, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Functions

	Name	Description
	TCPIP_DDNS_LastIPGet	Returns the status of the most recent update.
	TCPIP_DDNS_LastStatusGet	Returns the last known external IP address of the device.
	TCPIP_DDNS_ServiceSet	Selects a pre-configured Dynamic DNS service
	TCPIP_DDNS_UpdateForce	Forces an immediate DDNS update

Description

This section describes the Application Programming Interface (API) functions of the DDNS TCP/IP Stack.

Refer to each section for a detailed description.

5.8.9.7.1 Functions

5.8.9.7.1.1 TCPIP_DDNS_LastIPGet Function

C

```
IPV4_ADDR TCPIP_DDNS_LastIPGet();
```

Description

This function returns the status of the most recent update. See the [DDNS_STATUS](#) enumeration for possible codes.

Preconditions

None

Returns

[DDNS_STATUS](#) indicating the status code for the most recent update.

5.8.9.7.1.2 TCPIP_DDNS_LastStatusGet Function

C

```
DDNS_STATUS TCPIP_DDNS_LastStatusGet();
```

Description

This function returns the last known external IP address of the device.

Preconditions

None

Returns

The last known external IP address of the device.

5.8.9.7.1.3 TCPIP_DDNS_ServiceSet Function**C**

```
void TCPIP_DDNS_ServiceSet(  
    DDNS_SERVICES svc  
);
```

Description

This function selects a Dynamic DNS service based on parameters configured in `ddnsServiceHosts` and `ddnsServicePorts`. These arrays must match the [DDNS_SERVICES](#) enumeration.

Preconditions

None

Parameters

Parameters	Description
svc	one of the DDNS_SERVICES elements to indicate the selected service

Returns

None

5.8.9.7.1.4 TCPIP_DDNS_UpdateForce Function**C**

```
void TCPIP_DDNS_UpdateForce();
```

Description

This function forces the DDNS Client to execute a full update immediately. Any error message is cleared, and the update will be executed whether the IP address has changed or not. Call this function every time the DDNSClient parameters have been modified.

Preconditions

TCPIP_DDNS_Initialize must have been called.

Returns

None

5.8.9.7.2 Data Types and Constants**5.8.9.7.2.1 DDNS_MODULE_CONFIG Structure****C**

```
typedef struct {  
} DDNS_MODULE_CONFIG;
```

Description

This is type DDNS_MODULE_CONFIG.

5.8.9.7.2.2 DDNS_POINTERS Structure

C

```
typedef struct {
    union {
        uint8_t * szRAM;
        const uint8_t * szROM;
    } CheckIPServer;
    uint16_t CheckIPPort;
    union {
        uint8_t * szRAM;
        const uint8_t * szROM;
    } UpdateServer;
    uint16_t UpdatePort;
    union {
        uint8_t * szRAM;
        const uint8_t * szROM;
    } Username;
    union {
        uint8_t * szRAM;
        const uint8_t * szROM;
    } Password;
    union {
        uint8_t * szRAM;
        const uint8_t * szROM;
    } Host;
    struct {
        unsigned char CheckIPServer : 1;
        unsigned char UpdateServer : 1;
        unsigned char Username : 1;
        unsigned char Password : 1;
        unsigned char Host : 1;
    } ROMPointers;
} DDNS_POINTERS;
```

Description

This structure of pointers configures the Dynamic DNS Client. Initially, all pointers will be null and the client will be disabled. Set `DDNSClient.[field name].szRAM` to use a string stored in RAM, or `DDNSClient.[field name].szROM` to use a string stored in const. (Where [field name] is one of the parameters below.)

If a const string is specified, `DDNSClient.ROMPointers.[field name]` must also be set to 1 to indicate that this field should be retrieved from const instead of RAM.

Parameters

Parameters	Description
CheckIPServer	The server used to determine the external IP address
CheckIPPort	Port on the above server to connect to
UpdateServer	The server where updates should be posted
UpdatePort	Port on the above server to connect to
Username	The user name for the dynamic DNS server
Password	The password to supply when making updates
Host	The host name you wish to update
ROMPointers	Indicates which parameters to read from const instead of RAM.

5.8.9.7.2.3 DDNS_SERVICES Enumeration

C

```
typedef enum {  
    DYNDNS_ORG = 0u,  
    NO_IP_COM,  
    DNSOMATIC_COM  
} DDNS_SERVICES;
```

Description

Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.

Members

Members	Description
DYNDNS_ORG = 0u	www.dyndns.org
NO_IP_COM	www.no-ip.com
DNSOMATIC_COM	www.dnsomatic.com

5.8.9.7.2.4 DDNS_STATUS Enumeration

C

```
typedef enum {  
    DDNS_STATUS_GOOD = 0u,  
    DDNS_STATUS_NOCHG,  
    DDNS_STATUS_ABUSE,  
    DDNS_STATUS_BADSYS,  
    DDNS_STATUS_BADAGENT,  
    DDNS_STATUS_BDAUTH,  
    DDNS_STATUS_NOT_DONATOR,  
    DDNS_STATUS_NOT_FQDN,  
    DDNS_STATUS_NOHOST,  
    DDNS_STATUS_NOT_YOURS,  
    DDNS_STATUS_NUMHOST,  
    DDNS_STATUS_DNSERR,  
    DDNS_STATUS_911,  
    DDNS_STATUS_UPDATE_ERROR,  
    DDNS_STATUS_UNCHANGED,  
    DDNS_STATUS_CHECKIP_ERROR,  
    DDNS_STATUS_DNS_ERROR,  
    DDNS_STATUS_SKT_ERROR,  
    DDNS_STATUS_INVALID,  
    DDNS_STATUS_UNKNOWN  
} DDNS_STATUS;
```

Description

Status message for DynDNS client. GOOD and NOCHG are ok, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Members

Members	Description
DDNS_STATUS_GOOD = 0u	Update successful, hostname is now updated
DDNS_STATUS_NOCHG	Update changed no setting and is considered abusive. Additional 'nochg' updates will cause hostname to be blocked.
DDNS_STATUS_ABUSE	The hostname specified is blocked for update abuse.
DDNS_STATUS_BADSYS	System parameter not valid. Should be dyndns, statdns or custom.

DDNS_STATUS_BADAGENT	The user agent was blocked or not sent.
DDNS_STATUS_BDAUTH	The username and password pair do not match a real user.
DDNS_STATUS_NOT_DONATOR	An option available only to credited users (such as offline URL) was specified, but the user is not a credited user. If multiple hosts were specified, only a single !donator will be returned.
DDNS_STATUS_NOT_FQDN	The hostname specified is not a fully-qualified domain name (not in the form hostname.dyndns.org or domain.com).
DDNS_STATUS_NOHOST	The hostname specified does not exist in this user account (or is not in the service specified in the system parameter).
DDNS_STATUS_NOT_YOURS	The hostname specified does not belong to this user account.
DDNS_STATUS_NUMHOST	Too many hosts specified in an update.
DDNS_STATUS_DNSERR	Unspecified DNS error encountered by the DDNS service.
DDNS_STATUS_911	There is a problem or scheduled maintenance with the DDNS service.
DDNS_STATUS_UPDATE_ERROR	Error communicating with Update service.
DDNS_STATUS_UNCHANGED	The IP Check indicated that no update was necessary.
DDNS_STATUS_CHECKIP_ERROR	Error communicating with CheckIP service.
DDNS_STATUS_DNS_ERROR	DNS error resolving the CheckIP service.
DDNS_STATUS_SKT_ERROR	TCP socket opening error
DDNS_STATUS_INVALID	DDNS Client data is not valid.
DDNS_STATUS_UNKNOWN	DDNS client has not yet been executed with this configuration.

5.8.9.8 Files

Files

Name	Description
ddns.h	
ddns_config.h	DynDNS configuration file

Description

5.8.9.8.1 ddns.h

DynDNS Headers for Microchip TCP/IP Stack

Enumerations

	Name	Description
	DDNS_SERVICES	Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.
	DDNS_STATUS	Status message for DynDNS client. GOOD and NOCHG are ok, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Functions

	Name	Description
	TCPIP_DDNS_LastIPGet	Returns the status of the most recent update.
	TCPIP_DDNS_LastStatusGet	Returns the last known external IP address of the device.
	TCPIP_DDNS_ServiceSet	Selects a pre-configured Dynamic DNS service

	TCPIP_DDNS_UpdateForce	Forces an immediate DDNS update

Structures

	Name	Description
	DDNS_MODULE_CONFIG	This is type DDNS_MODULE_CONFIG.
	DDNS_POINTERS	Configuration parameters for the Dynamic DNS Client

5.8.9.8.2 ddns_config.h

Dynamic Domain Name Service (DynDNS) Configuration file

This file contains the DynDNS module configuration options

Macros

	Name	Description
	DDNS_CHECKIP_SERVER	Default CheckIP server for determining current IP address
	DDNS_DEFAULT_PORT	Default port for CheckIP server

5.8.10 FTP TCP/IP Stack Library**5.8.10.1 Introduction****File Transfer Protocol (FTP) TCP/IP Stack Library**

for

Microchip Microcontrollers

This library provides the API of the FTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

An embedded FTP (File Transfer Protocol) server is an excellent addition to any network-enabled device. FTP server capability facilitates the uploading of files to, and downloading of files from, an embedded device. Almost all computers have, at the very least, a command line FTP client that will allow a user to [connect](#) to an embedded FTP server.

5.8.10.2 Release Notes

MPLAB Harmony Version: v0.70b **FTP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

The FTP server provided here with these following functionalities. You can easily add new functionality as required.

The FTP server presented here incorporates the following features:

- Server Works with both Passive and Active mode

- FTP Server APIs are compatible with PIC24
- FTP connection is authenticated by using username as "admin" and password "microchip". Anonymous login is provided
- GET, MGET, PUT, and MPUT commands are provided

Known Issues:

None to report at this time.

5.8.10.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.10.4 Using the Library

This topic describes the basic architecture of the FTP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [ftp.h](#)

The interface to the FTP TCP/IP Stack library is defined in the "[ftp.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the FTP TCP/IP Stack library should include "tcpip.h".

Library File:

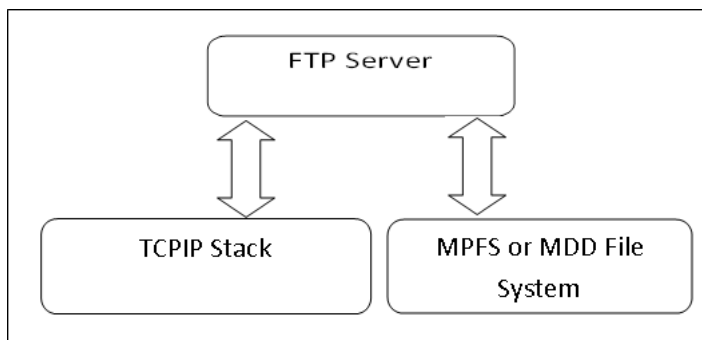
The FTP TCP/IP Stack library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.10.4.1 Abstraction Model

This library provides the API of the FTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description**FTP Server Dependencies**



Since FTP server requires the use of TCP/IP stack and a file system which can be a MPFS or MDD , it also inherits the program memory and RAM requirements of those as well.

5.8.10.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the FTP module.

Library Interface Section	Description
Data Types and Constants	This section provides various definitions describing This API

5.8.10.5 Configuring the Library

Macros

	Name	Description
	TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the FTP Data socket Use 0 for default
	TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the FTP Data socket Use 0 for default
	TCPIP_FTP_MAX_CONNECTIONS	Maximum number of FTP connections allowed per interface
	TCPIP_FTP_PASSWD_LEN	Specifies the max length of FTP login password
	TCPIP_FTP_PUT_ENABLED	Comment this line out to disable MPFS
	TCPIP_FTP_USER_NAME_LEN	Specifies the max length for user name

Description

The configuration of the FTP TCP/IP Stack is based on the file `tcPIP_config.h`

This header file contains the configuration selection for the FTP TCP/IP Stack. Based on the selections made, the FTP TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the FTP TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.10.5.1 TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE Macro

C

```
#define TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE 0
```


Description

Define the size of the RX buffer for the FTP Data [socket](#) Use 0 for default

5.8.10.5.2 TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE Macro**C**

```
#define TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE 0
```

Description

Define the size of the TX buffer for the FTP Data [socket](#) Use 0 for default

5.8.10.5.3 TCPIP_FTP_MAX_CONNECTIONS Macro**C**

```
#define TCPIP_FTP_MAX_CONNECTIONS (1)
```

Description

Maximum number of FTP connections allowed per interface

5.8.10.5.4 TCPIP_FTP_PASSWD_LEN Macro**C**

```
#define TCPIP_FTP_PASSWD_LEN (10)
```

Description

Specifies the max length of FTP login password

5.8.10.5.5 TCPIP_FTP_PUT_ENABLED Macro**C**

```
#define TCPIP_FTP_PUT_ENABLED
```

Description

Comment this line out to disable MPFS

5.8.10.5.6 TCPIP_FTP_USER_NAME_LEN Macro**C**

```
#define TCPIP_FTP_USER_NAME_LEN (10)
```

Description

Specifies the max length for user name

5.8.10.6 Building the Library

This section list the files that are available in the \src of the FTP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.10.7 Library Interface

Data Types and Constants

	Name	Description
	FTP_MODULE_CONFIG	FTP module dynamic configuration data

Description

This section describes the Application Programming Interface (API) functions of the FTP TCP/IP Stack.

Refer to each section for a detailed description.

5.8.10.7.1 Data Types and Constants

5.8.10.7.1.1 FTP_MODULE_CONFIG Structure

C

```
typedef struct {
    uint16_t dataSktTxBuffSize;
    uint16_t dataSktRxBuffSize;
    uint8_t  userName[10+1];
    uint8_t  password[10+1];
} FTP_MODULE_CONFIG;
```

Description

FTP module dynamic configuration data

Members

Members	Description
uint16_t dataSktTxBuffSize;	size of Data socket TX buffer for the associatted socket ; leave 0 for default
uint16_t dataSktRxBuffSize;	size of Data Socket RX buffer for the associatted socket ; leave 0 for default
uint8_t userName[10+1];	FTP login User name TCPIP_FTP_USER_NAME_LEN
uint8_t password[10+1];	FTP login password TCPIP_FTP_PASSWD_LEN

5.8.10.8 Files

Files

Name	Description
ftp.h	
ftp_config.h	FTP configuration file

Description

5.8.10.8.1 ftp.h

FTP Server Defs for Microchip TCP/IP Stack

Structures

	Name	Description
	FTP_MODULE_CONFIG	FTP module dynamic configuration data

5.8.10.8.2 ftp_config.h

File Transfer Protocol (FTP) Configuration file

This file contains the FTP module configuration options

Macros

	Name	Description
	TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the FTP Data socket Use 0 for default
	TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the FTP Data socket Use 0 for default
	TCPIP_FTP_MAX_CONNECTIONS	Maximum number of FTP connections allowed per interface
	TCPIP_FTP_PASSWD_LEN	Specifies the max length of FTP login password
	TCPIP_FTP_PUT_ENABLED	Comment this line out to disable MPFS
	TCPIP_FTP_USER_NAME_LEN	Specifies the max length for user name

5.8.11 HTTP Server TCP/IP Stack Library**5.8.11.1 Introduction**

Hypertext Transfer Protocol (HTTP) TCP/IP Stack Library
for
Microchip Microcontrollers

This library provides the API of the HTTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The HTTP web server module and its associated MPFS2 file system module allow the board to act as a web server. This facilitates an easy method to view status information and control applications using any standard web browser.

5.8.11.2 Release Notes

MPLAB Harmony Version: v0.70b **HTTP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.11.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.11.4 Using the Library

This topic describes the basic architecture of the HTTP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [http.h](#)

The interface to the HTTP TCP/IP Stack library is defined in the "[http.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the HTTP TCP/IP Stack library should include "tcpip.h".

Library File:

The HTTP TCPIP Stack library archive (.a) file is installed with MPLAB Harmony.

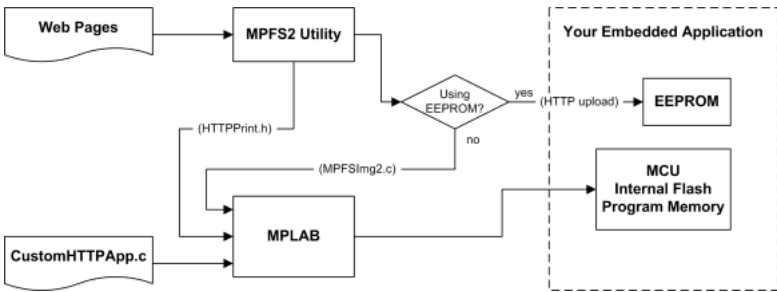
Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.11.4.1 Abstraction Model

This library provides the API of the HTTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description**HTTP Software Abstraction Block Diagram**

Three main components are necessary to understand how the HTTP web server works: the web pages, the MPFS2 Utility, and the source files CustomHTTPApp.c and HTTPPrint.h. An overview of the entire process is shown below.



Web Pages

This includes all the HTML and associated images, CSS stylesheets, and JavaScript files necessary to display the website. A sample application including all these components is located in the WebPages2 folder.

MPFS2 Utility

This program, supplied by Microchip, packages the web pages into a format that can be efficiently stored in either external non-volatile storage, or internal flash program memory. This program also indexes dynamic variables found in the web pages and updates HTTPPrint.h with these indices.

If external storage is being used, the MPFS2 Utility outputs a BIN file and can upload that file directly to the board. If the data

5.8.11.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the HTTP module.

Library Interface Section	Description
Functions	Routines for Configuring DHCP
Data Types and Constants	This section provides various definitions describing this API

5.8.11.5 Configuring the Library

Macros

	Name	Description
	HTTP_CACHE_LEN	Max lifetime (sec) of static responses as string
	HTTP_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default
	HTTP_DEFAULT_FILE	Indicate what file to serve when no specific one is requested
	HTTP_DEFAULT_LEN	For buffer overrun protection. Set to longest length of above two strings.
	HTTP_FILE_UPLOAD	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	HTTP_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	HTTP_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	HTTP_MAX_HEADER_LEN	Set to length of longest string above
	HTTP_MIN_CALLBACK_FREE	Define the minimum number of bytes free in the TX FIFO before executing callbacks
	HTTP_PORT	Define the listening port for the HTTP server

	HTTP_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the http socket Use 0 for default
	HTTP_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the http socket Use 0 for default
	HTTP_SSL_ONLY_CHAR	Files beginning with this character will only be served over HTTPS Set to 0x00 to require for all files Set to 0xff to require for no files
	HTTP_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	HTTP_USE_AUTHENTICATION	Enable basic authentication support
	HTTP_USE_COOKIES	Enable cookie support
	HTTP_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support
	HTTPS_DEFAULT_FILE	Indicate what file to serve when no specific one is requested
	HTTPS_PORT	Listening port for HTTPS server (when SSL enabled)
	TCPIP_STACK_USE_BASE64_DECODE	TCPIP_STACK_USE_BASE64_DECODE HTTP_USE_AUTHENTICATION

Description

The configuration of the HTTP TCP/IP Stack is based on the file `tcPIP_config.h`

This header file contains the configuration selection for the HTTP TCP/IP Stack. Based on the selections made, the HTTP TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the HTTP TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.11.5.1 HTTP_CACHE_LEN Macro**C**

```
#define HTTP_CACHE_LEN ("600")
```

Description

Max lifetime (sec) of static responses as string

5.8.11.5.2 HTTP_CONFIG_FLAGS Macro**C**

```
#define HTTP_CONFIG_FLAGS 0
```

Description

Define the HTTP module configuration flags Use 0 for default

5.8.11.5.3 HTTP_DEFAULT_FILE Macro**C**

```
#define HTTP_DEFAULT_FILE "index.htm"
```

Description

Indicate what file to serve when no specific one is requested

5.8.11.5.4 HTTP_DEFAULT_LEN Macro

C

```
#define HTTP_DEFAULT_LEN (10u)
```

Description

For buffer overrun protection. Set to longest length of above two strings.

5.8.11.5.5 HTTP_FILE_UPLOAD Macro

C

```
#define HTTP_FILE_UPLOAD "mpfsupload"
```

Description

Configure MPFS over HTTP updating Comment this line to disable updating via HTTP

5.8.11.5.6 HTTP_MAX_CONNECTIONS Macro

C

```
#define HTTP_MAX_CONNECTIONS (4)
```

Description

Maximum numbers of simultaneous supported HTTP connections.

5.8.11.5.7 HTTP_MAX_DATA_LEN Macro

C

```
#define HTTP_MAX_DATA_LEN (100u)
```

Description

Define the maximum data length for reading cookie and GET/POST arguments (bytes)

5.8.11.5.8 HTTP_MAX_HEADER_LEN Macro

C

```
#define HTTP_MAX_HEADER_LEN (15u)
```

Description

Set to length of longest string above

5.8.11.5.9 HTTP_MIN_CALLBACK_FREE Macro

C

```
#define HTTP_MIN_CALLBACK_FREE (16u)
```

Description

Define the minimum number of bytes free in the TX FIFO before executing callbacks

5.8.11.5.10 HTTP_PORT Macro

C

```
#define HTTP_PORT (80u)
```

Description

Define the listening port for the HTTP server

5.8.11.5.11 HTTP_SKT_RX_BUFF_SIZE Macro

C

```
#define HTTP_SKT_RX_BUFF_SIZE 0
```

Description

Define the size of the RX buffer for the http [socket](#) Use 0 for default

5.8.11.5.12 HTTP_SKT_TX_BUFF_SIZE Macro

C

```
#define HTTP_SKT_TX_BUFF_SIZE 0
```

Description

Define the size of the TX buffer for the http [socket](#) Use 0 for default

5.8.11.5.13 HTTP_SSL_ONLY_CHAR Macro

C

```
#define HTTP_SSL_ONLY_CHAR (0xFF)
```

Description

Files beginning with this character will only be served over HTTPS Set to 0x00 to require for all files Set to 0xff to require for no files

5.8.11.5.14 HTTP_TIMEOUT Macro

C

```
#define HTTP_TIMEOUT (45u)
```

Description

Max time (sec) to await more data before timing out and disconnecting the [socket](#)

5.8.11.5.15 HTTP_USE_AUTHENTICATION Macro

C

```
#define HTTP_USE_AUTHENTICATION
```

Description

Enable basic authentication support

5.8.11.5.16 HTTP_USE_COOKIES Macro

C

```
#define HTTP_USE_COOKIES
```

Description

Enable cookie support

5.8.11.5.17 HTTP_USE_POST Macro

C

```
#define HTTP_USE_POST
```

Description

Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support

5.8.11.5.18 HTTPS_DEFAULT_FILE Macro

C

```
#define HTTPS_DEFAULT_FILE "index.htm"
```

Description

Indicate what file to serve when no specific one is requested

5.8.11.5.19 HTTPS_PORT Macro

C

```
#define HTTPS_PORT (443u)
```

Description

Listening port for HTTPS server (when SSL enabled)

5.8.11.5.20 TCPIP_STACK_USE_BASE64_DECODE Macro

C

```
#define TCPIP_STACK_USE_BASE64_DECODE
```

Description

TCPIP_STACK_USE_BASE64_DECODE [HTTP_USE_AUTHENTICATION](#)

5.8.11.6 Library Interface

Data Types and Constants

	Name	Description
	HTTP_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection
	HTTP_FILE_TYPE	File type definitions
	HTTP_IO_RESULT	Result states for execution callbacks

	HTTP_MODULE_CONFIG	HTTP module dynamic configuration data
	HTTP_MODULE_FLAGS	HTTP module configuration flags
	HTTP_READ_STATUS	Result states for HTTPPostReadName and HTTPPostReadValue
	HTTP_STATUS	Supported Commands and Server Response Codes
	HTTPReadPostPair	Reads a name and value pair from a URL encoded string in the TCP buffer.

Functions

	Name	Description
	TCPIP_HTTP_ActiveConnectionCountGet	Gets the number of active connections.
	TCPIP_HTTP_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_CurrentConnectionByteCountDec	Decrements byte count.
	TCPIP_HTTP_CurrentConnectionByteCountGet	Returns how many bytes have been read so far
	TCPIP_HTTP_CurrentConnectionByteCountSet	Sets how many bytes have been read so far
	TCPIP_HTTP_CurrentConnectionCallbackPosGet	Returns the callback position indicator.
	TCPIP_HTTP_CurrentConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_CurrentConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.
	TCPIP_HTTP_CurrentConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_CurrentConnectionHasArgsSet	Sets whether there are get or cookie arguments
	TCPIP_HTTP_CurrentConnectionIsAuthorizedGet	Get the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_CurrentConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_CurrentConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_CurrentConnectionStatusSet	Sets HTTP status.
	TCPIP_HTTP_CurrentConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_CurrentConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_FileInclude	Writes a file byte-for-byte to the currently loaded TCP socket .
	TCPIP_HTTP_PostNameRead	Reads a name from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_PostValueRead	Reads a value from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_URLDecode	Parses a string from URL encoding to plain-text.
	HTTPCheckAuth	Performs validation on a specific user name and password.
	HTTPExecuteGet	Processes GET form field variables and cookies.
	HTTPExecutePost	Processes POST form variables and data.
	HTTPNeedsAuth	Determines if a given file name requires authentication
	HTTPPrint_varname	Inserts dynamic content into a web page

Description

This section describes the Application Programming Interface (API) functions of the HTTP TCP/IP Stack.

Refer to each section for a detailed description.

5.8.11.6.1 Functions

5.8.11.6.1.1 TCPIP_HTTP_ActiveConnectionCountGet Function

C

```
int TCPIP_HTTP_ActiveConnectionCountGet();
```

Description

Returns the number of active connections at the current time.

Preconditions

None

Remarks

The value returned by this function is informational only. The number of active connections changes dynamically.

Example

```
int nConns;

nConns = TCPIP_HTTP_ActiveConnectionCountGet();
```

5.8.11.6.1.2 TCPIP_HTTP_ArgGet Function**C**

```
const uint8_t* TCPIP_HTTP_ArgGet(
    const uint8_t* cData,
    const uint8_t* cArg
);
```

Description

Searches through a data array to find the value associated with a given argument. It can be used to find form field values in data received over GET or POST.

The end of data is assumed to be reached when a null name parameter is encountered. This requires the string to have an even number of null-terminated strings, followed by an additional null terminator.

Preconditions

The data array has a valid series of null terminated name/value pairs.

Parameters

Parameters	Description
data	the buffer to search
arg	the name of the argument to find

Returns

A pointer to the argument value, or NULL if not found.

Remarks

None.

Example

```
void HTTPPrint_cookieName(HTTP_CONN_HANDLE connHandle)
{
    const uint8_t *ptr;
    TCP_SOCKET sktHTTP;

    ptr = TCPIP_HTTP_ArgGet(TCPIP_HTTP_CurrentConnectionDataBufferGet(connHandle), (const
uint8_t*)"name");
    sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
    if(ptr)
        TCPIP_TCP_StringPut(sktHTTP, ptr);
    else
        TCPIP_TCP_StringPut(sktHTTP, (const uint8_t*)"not set");
}
```

5.8.11.6.1.3 TCPIP_HTTP_CurrentConnectionByteCountDec Function

C

```
void TCPIP_HTTP_CurrentConnectionByteCountDec(  
    HTTP_CONN_HANDLE connHandle,  
    uint32_t byteCount  
);
```

Description

Decrements byte count.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
byteCount	byte count reduction

Remarks

None.

Example

5.8.11.6.1.4 TCPIP_HTTP_CurrentConnectionByteCountGet Function

C

```
uint32_t TCPIP_HTTP_CurrentConnectionByteCountGet(  
    HTTP_CONN_HANDLE connHandle  
);
```

Description

Returns how many bytes have been read so far

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None.

Example

```
switch(TCPIP_HTTP_CurrentConnectionPostSmGet(connHandle))  
{  
    case SM_CFG_SNMP_READ_NAME:  
        // If all parameters have been read, end  
        if(TCPIP_HTTP_CurrentConnectionByteCountGet(connHandle) == 0u)  
        {  
            return HTTP_IO_DONE;  
        }  
    .  
    .  
    .  
}
```

```
}
```

5.8.11.6.1.5 TCPIP_HTTP_CurrentConnectionByteCountSet Function

C

```
void TCPIP_HTTP_CurrentConnectionByteCountSet(  
    HTTP_CONN_HANDLE connHandle,  
    uint32_t byteCount  
);
```

Description

Sets how many bytes have been read so far

Parameters

Parameters	Description
connHandle	HTTP connection handle
byteCount	byte count to be set

Remarks

None.

Example

5.8.11.6.1.6 TCPIP_HTTP_CurrentConnectionCallbackPosGet Function

C

```
uint32_t TCPIP_HTTP_CurrentConnectionCallbackPosGet(  
    HTTP_CONN_HANDLE connHandle  
);
```

Description

Returns the callback position indicator for connection defined by connHandle.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None.

Example

```
uint32_t callbackPos;  
callbackPos = TCPIP_HTTP_CurrentConnectionCallbackPosGet(connHandle);  
if(callbackPos == 0x00u)  
    callbackPos = (uint32_t)DDNSClient.Host.szRAM;  
callbackPos =  
(uint32_t)TCPIP_TCP_StringPut(TCPIP_HTTP_CurrentConnectionSocketGet(connHandle),  
(uint8_t*)callbackPos);  
if(*(uint8_t*)callbackPos == '0')  
    callbackPos = 0x00;  
TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, callbackPos);
```

5.8.11.6.1.7 TCPIP_HTTP_CurrentConnectionCallbackPosSet Function

C

```
void TCPIP_HTTP_CurrentConnectionCallbackPosSet(  
    HTTP_CONN_HANDLE connHandle,  
    uint32_t  
);
```

Description

Sets the callback position indicator for connection defined by connHandle.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None.

Example

```
void HTTPPrint_builddate(HTTP_CONN_HANDLE connHandle)  
{  
    TCP_SOCKET sktHTTP;  
    sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);  
  
    TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x01);  
    if(TCPIP_TCP_PutIsReady(sktHTTP) < strlen((const char*)__DATE__ " " __TIME__))  
    { // Don't have room to output build date and time  
        return;  
    }  
    TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x00);  
    TCPIP_TCP_StringPut(sktHTTP, (const void*)__DATE__ " " __TIME__);  
}
```

5.8.11.6.1.8 TCPIP_HTTP_CurrentConnectionDataBufferGet Function

C

```
uint8_t* TCPIP_HTTP_CurrentConnectionDataBufferGet(  
    HTTP_CONN_HANDLE connHandle  
);
```

Description

Returns pointer to connection general purpose data buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None.

Example

```

void HTTPPrint_cookiesname(HTTP_CONN_HANDLE connHandle)
{
    const uint8_t *ptr;
    TCP_SOCKET sktHTTP;

    ptr = TCPIP_HTTP_ArgGet(TCPIP_HTTP_CurrentConnectionDataBufferGet(connHandle), (const
uint8_t*)"name");
    sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
    if(ptr)
        TCPIP_TCP_StringPut(sktHTTP, ptr);
    else
        TCPIP_TCP_StringPut(sktHTTP, (const uint8_t*)"not set");
}

```

5.8.11.6.1.9 TCPIP_HTTP_CurrentConnectionFileGet Function**C**

```

FILE_HANDLE TCPIP_HTTP_CurrentConnectionFileGet(
    HTTP_CONN_HANDLE connHandle
);

```

Description

Get handle to current connection's file.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None.

Example

```

uint8_t filename[20];

// Load the file name
// Make sure uint8_t filename[] above is large enough for your longest name
MPFSGetFilename(TCPIP_HTTP_CurrentConnectionFileGet(connHandle), filename, 20);

```

5.8.11.6.1.10 TCPIP_HTTP_CurrentConnectionHasArgsSet Function**C**

```

void TCPIP_HTTP_CurrentConnectionHasArgsSet(
    HTTP_CONN_HANDLE connHandle,
    bool args
);

```

Description

Sets whether there are get or cookie arguments

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
args	boolean if there arguments or not

Remarks

None.

Example

```
else if(!memcmp(filename, "cookies.htm", 11))
{ // If it's the LED updater file
    TCPIP_HTTP_CurrentConnectionHasArgsSet(connHandle, true);
}
```

5.8.11.6.1.11 TCPIP_HTTP_CurrentConnectionIsAuthorizedGet Function**C**

```
uint8_t TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(
    HTTP_CONN_HANDLE connHandle
);
```

Description

Get the authorization status for the current connection.

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None.

Example

```
uint8_t isAuth;

isAuth = TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(connHandle);
```

5.8.11.6.1.12 TCPIP_HTTP_CurrentConnectionIsAuthorizedSet Function**C**

```
void TCPIP_HTTP_CurrentConnectionIsAuthorizedSet(
    HTTP_CONN_HANDLE connHandle,
    uint8_t auth
);
```

Description

Sets the authorization status for the current connection.

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle
auth	new authorization state

Remarks

None.

Example

```
uint8_t auth = 0x80;

TCPIP_HTTP_CurrentConnectionIsAuthorizedSet(connHandle, auth);
```

5.8.11.6.1.13 TCPIP_HTTP_CurrentConnectionPostSmGet Function**C**

```
int TCPIP_HTTP_CurrentConnectionPostSmGet(
    HTTP_CONN_HANDLE connHandle
);
```

Description

Get the POST state machine state for connection defined by connHandle

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None

Example

```
switch(TCPIP_HTTP_CurrentConnectionPostSmGet(connHandle))
{
    // Find the name
    case SM_POST_LCD_READ_NAME:
        .
        .
        .
    // Found the value, so store the LCD and return
    case SM_POST_LCD_READ_VALUE:
        .
        .
        .
}
```

5.8.11.6.1.14 TCPIP_HTTP_CurrentConnectionPostSmSet Function**C**

```
void TCPIP_HTTP_CurrentConnectionPostSmSet(
    HTTP_CONN_HANDLE connHandle,
    int state
);
```

Description

Set the POST state machine state for connection defined by connHandle

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
state	Integer state for POST state machine

Remarks

None

Example

```
uint8_t* httpDataBuff;
switch(TCPIP_HTTP_CurrentConnectionPostSmGet(connHandle))
{
    // Find the name
    case SM_POST_LCD_READ_NAME:

        // Read a name
        if(TCPIP_HTTP_PostNameRead(connHandle, httpDataBuff, HTTP_MAX_DATA_LEN) ==
HTTP_READ_INCOMPLETE)
            return HTTP_IO_NEED_DATA;

        TCPIP_HTTP_CurrentConnectionPostSmSet(connHandle, SM_POST_LCD_READ_VALUE);
        // No break...continue reading value

        // Found the value, so store the LCD and return
    case SM_POST_LCD_READ_VALUE:
        .
        .
        .
}
```

5.8.11.6.1.15 TCPIP_HTTP_CurrentConnectionSocketGet Function**C**

```
TCP_SOCKET TCPIP_HTTP_CurrentConnectionSocketGet(
    HTTP_CONN_HANDLE connHandle
);
```

Description

Get the [socket](#) for the current connection.

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None.

Example

```
uint32_t byteCount;
TCP_SOCKET sktHTTP;

byteCount = TCPIP_HTTP_CurrentConnectionByteCountGet(connHandle);
sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
if(byteCount > TCPIP_TCP_GetIsReady(sktHTTP) + TCPIP_TCP_FifoRxFreeGet(sktHTTP))
{
    // Configuration Failure
    TCPIP_HTTP_CurrentConnectionStatusSet(connHandle, HTTP_REDIRECT);
    return HTTP_IO_DONE;
}
```

5.8.11.6.1.16 TCPIP_HTTP_CurrentConnectionStatusSet Function**C**

```
void TCPIP_HTTP_CurrentConnectionStatusSet(
    HTTP_CONN_HANDLE connHandle,
    HTTP_STATUS stat
);
```

Description

Sets HTTP status to value in enumeration [HTTP_STATUS](#).

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None.

Example

```
byteCount = TCPIP_HTTP_CurrentConnectionByteCountGet(connHandle);
sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
if(byteCount > TCPIP_TCP_GetIsReady(sktHTTP) + TCPIP_TCP_FifoRxFreeGet(sktHTTP))
{
    // Configuration Failure
    // 302 Redirect will be returned
    TCPIP_HTTP_CurrentConnectionStatusSet(connHandle, HTTP_REDIRECT);
}
```

5.8.11.6.1.17 TCPIP_HTTP_CurrentConnectionUserDataGet Function**C**

```
const void* TCPIP_HTTP_CurrentConnectionUserDataGet(
    HTTP_CONN_HANDLE connHandle
);
```

Description

Returns the value of the user parameter for the current connection.

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle

Remarks

None.

Example

```
uint32_t myConnData;  
  
myConnData = (uint32_t)TCPIP_HTTP_CurrentConnectionUserDataGet(connHandle);
```

5.8.11.6.1.18 TCPIP_HTTP_CurrentConnectionUserDataSet Function**C**

```
void TCPIP_HTTP_CurrentConnectionUserDataSet(  
    HTTP_CONN_HANDLE connHandle,  
    const void* uData  
);
```

Description

Sets the user data value for the current connection.

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle
uData	user supplied data

Remarks

None.

Example

```
uint32_t myConnData;  
  
TCPIP_HTTP_CurrentConnectionUserDataSet(connHandle, (const void*)myConnData);
```

5.8.11.6.1.19 TCPIP_HTTP_FileInclude Function**C**

```
void TCPIP_HTTP_FileInclude(  
    HTTP_CONN_HANDLE connHandle,  
    const uint8_t* cFile  
);
```

Description

Allows an entire file to be included as a dynamic variable, providing a basic templating system for HTML web pages. This reduces unneeded duplication of visual elements such as headers, menus, etc.

When pHttpCon->callbackPos is 0, the file is opened and as many bytes as possible are written. The current position is then saved to pHttpCon->callbackPos and the file is closed. On subsequent calls, reading begins at the saved location and continues. Once the end of the input file is reached, pHttpCon->callbackPos is set back to 0 to indicate completion.

Preconditions

None

Parameters

Parameters	Description
cFile	the name of the file to be sent

Returns

None

Remarks

Users should not call this function directly, but should instead add dynamic variables in the form of ~inc:filename.ext~ in their HTML code to include (for example) the file "filename.ext" at that specified location. The MPFS2 Generator utility will handle the rest.

Example**5.8.11.6.1.20 TCPIP_HTTP_PostNameRead Function****C**

```
HTTP_READ_STATUS TCPIP_HTTP_PostNameRead(  
    HTTP_CONN_HANDLE connHandle,  
    uint8_t* cData,  
    uint16_t wLen  
);
```

Description

Reads a name from a URL encoded string in the TCP buffer. This function is meant to be called from an [HTTPExecutePost](#) callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '=' character, which indicates the end of a name parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. It also removes the delimiting '=' from the buffer.

Preconditions

Front of TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name once it is read
wLen	how many bytes can be written to cData

Return Values

Return Values	Description
HTTP_READ_OK	name was successfully read
HTTP_READ_TRUNCATED	entire name could not fit in the buffer, so the value was truncated and data has been lost

HTTP_READ_INCOMPLETE	entire name was not yet in the buffer, so call this function again later to retrieve

Remarks

None

Example**5.8.11.6.1.21 TCPIP_HTTP_PostValueRead Function****C**

```

HTTP_READ_STATUS TCPIP_HTTP_PostValueRead(
    HTTP_CONN_HANDLE connHandle,
    uint8_t* cData,
    uint16_t wLen
);

```

Description

Reads a value from a URL encoded string in the TCP buffer. This function is meant to be called from an [HTTPExecutePost](#) callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData above the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the value parameter to be read. If pHttpCon->byteCount indicates that all expected bytes are in the buffer, it assumes that all remaining data is the value and acts accordingly.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. The terminating '&' character is also removed from the buffer.

Preconditions

Front of TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the value once it is read
wLen	how many bytes can be written to cData

Return Values

Return Values	Description
HTTP_READ_OK	value was successfully read
HTTP_READ_TRUNCATED	entire value could not fit in the buffer, so the value was truncated and data has been lost
HTTP_READ_INCOMPLETE	entire value was not yet in the buffer, so call this function again later to retrieve

Remarks

None

Example

5.8.11.6.1.22 TCPIP_HTTP_URLDecode Function

C

```
uint8_t* TCPIP_HTTP_URLDecode(  
    uint8_t* cData  
);
```

Description

Parses a string from URL encoding to plain-text. The following conversions are made: '=' to '0', '&' to '0', '+' to ' ', and "%xx" to a single hex byte.

After completion, the data has been decoded and a null terminator signifies the end of a name or value. A second null terminator (or a null name parameter) indicates the end of all the data.

Preconditions

The data parameter is null terminated and has at least one extra byte free.

Parameters

Parameters	Description
cData	The string which is to be decoded in place.

Returns

A pointer to the last null terminator in data, which is also the first free byte for new data.

Remarks

This function is called by the stack to parse GET arguments and cookie data. User applications can use this function to decode POST data, but first need to verify that the string is null-terminated.

Example

code>

5.8.11.6.1.23 HTTPCheckAuth Function

C

```
uint8_t HTTPCheckAuth(  
    HTTP_CONN_HANDLE connHandle,  
    uint8_t* cUser,  
    uint8_t* cPass  
);
```

Description

This function is implemented by the application developer in CustomHTTPApp.c. Its function is to determine if the user name and password supplied by the client are acceptable for this resource.

The value of curHTTP.isAuthorized will be set to the previous return value of HTTPRequiresAuthorization. This callback function can check this value to determine if only specific user names or passwords will be accepted for this resource.

Return values 0x80 - 0xff indicate that the credentials were accepted, while values from 0x00 to 0x79 indicate that authorization failed. While most applications will only use a single value to grant access, flexibility is provided to store multiple values in order to indicate which user (or user's group) logged in.

The return value of this function is saved as curHTTP.isAuthorized, and will be available to future callbacks, including any of the [HTTPExecuteGet](#), [HTTPExecutePost](#), or [HTTPPrint_varname](#) callbacks.

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle
cUser	the user name supplied by the client
cPass	the password supplied by the client

Return Values

Return Values	Description
<= 0x79	the credentials were rejected
>= 0x80	access is granted for this connection

Remarks

This function is only called when an Authorization header is encountered.

This function may NOT write to the TCP buffer.

5.8.11.6.1.24 HTTPExecuteGet Function**C**

```
HTTP_IO_RESULT HTTPExecuteGet(  
    HTTP_CONN_HANDLE connHandle  
);
```

Description

This function is implemented by the application developer in CustomHTTPApp.c. Its purpose is to parse the data received from URL parameters (GET method forms) and cookies and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated.

When this function is called, curHTTP.data contains sequential name/value pairs of strings representing the data received. In this format, [TCPIP_HTTP_ArgGet](#) can be used to search for specific variables in the input. If data buffer space associated with this connection is required, curHTTP.data may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including [HTTPExecutePost](#) and any [HTTPPrint_varname](#) dynamic substitutions.

This function may also issue redirections by setting curHTTP.data to the destination file name or URL, and curHTTP.httpStatus to HTTP_REDIRECT.

Finally, this function may set cookies. Set curHTTP.data to a series of name/value string pairs (in the same format in which parameters arrive) and then set curHTTP.hasArgs equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in curHTTP.data.

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle

Return Values

Return Values	Description
HTTP_IO_DONE	application is done processing
HTTP_IO_NEED_DATA	this value may not be returned because more data will not become available
HTTP_IO_WAITING	the application is waiting for an asynchronous process to complete, and this function should be called again later

Remarks

This function is only called if variables are received via URL parameters or Cookie arguments. This function may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use `curHTTP.callbackPos` or `curHTTP.data` for storage associated with individual requests.

5.8.11.6.1.25 HTTPExecutePost Function

C

```
HTTP_IO_RESULT HTTPExecutePost(  
    HTTP_CONN_HANDLE connHandle  
);
```

Description

This function is implemented by the application developer in `CustomHTTPApp.c`. Its purpose is to parse the data received from POST forms and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated before this function is called.

When this function is called, POST data will be waiting in the TCP buffer. `curHTTP.byteCount` will indicate the number of bytes remaining to be received before the browser request is complete.

Since data is still in the TCP buffer, the application must call [TCPIP_TCP_Get](#) or [TCPIP_TCP_ArrayGet](#) in order to retrieve bytes. When this is done, `curHTTP.byteCount` MUST be updated to reflect how many bytes now remain. The functions [TCPIP_TCP_Find](#), `TCPIFindString`, and [TCPIP_TCP_ArrayFind](#) may be helpful to locate data in the TCP buffer.

In general, data submitted from web forms via POST is URL encoded. The [TCPIP_HTTP_URLDecode](#) function can be used to decode this information back to a standard string if required. If data buffer space associated with this connection is required, `curHTTP.data` may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including `HTTPExecutePost` and any [HTTPPrint_varname](#) dynamic substitutions.

Whenever a POST form is processed it is recommended to issue a redirect back to the browser, either to a status page or to the same form page that was posted. This prevents accidental duplicate submissions (by clicking refresh or back/forward) and avoids browser warnings about "resubmitting form data". Redirects may be issued to the browser by setting `curHTTP.data` to the destination file or URL, and `curHTTP.httpStatus` to `HTTP_REDIRECT`.

Finally, this function may set cookies. Set `curHTTP.data` to a series of name/value string pairs (in the same format in which parameters arrive) and then set `curHTTP.hasArgs` equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in `curHTTP.data`.

Preconditions

None

Parameters

Parameters	Description
<code>connHandle</code>	HTTP connection handle

Return Values

Return Values	Description
HTTP_IO_DONE	application is done processing
HTTP_IO_NEED_DATA	more data is needed to continue, and this function should be called again later
HTTP_IO_WAITING	the application is waiting for an asynchronous process to complete, and this function should be called again later

Remarks

This function is only called when the request method is POST, and is only used when [HTTP_USE_POST](#) is defined. This method may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use curHTTP.callbackPos or curHTTP.data for storage associated with individual requests.

5.8.11.6.1.26 HTTPNeedsAuth Function**C**

```
uint8_t HTTPNeedsAuth(  
    HTTP_CONN_HANDLE connHandle,  
    uint8_t* cFile  
);
```

Description

This function is implemented by the application developer in CustomHTTPApp.c. Its function is to determine if a file being requested requires authentication to view. The user name and password, if supplied, will arrive later with the request headers, and will be processed at that time.

Return values 0x80 - 0xff indicate that authentication is not required, while values from 0x00 to 0x79 indicate that a user name and password are required before proceeding. While most applications will only use a single value to grant access and another to require authorization, the range allows multiple "realms" or sets of pages to be protected, with different credential requirements for each.

The return value of this function is saved as curHTTP.isAuthorized, and will be available to future callbacks, including [HTTPCheckAuth](#) and any of the [HTTPExecuteGet](#), [HTTPExecutePost](#), or [HTTPPrint_varname](#) callbacks.

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle
cFile	the name of the file being requested

Return Values

Return Values	Description
<= 0x79	valid authentication is required
>= 0x80	access is granted for this connection

Remarks

This function may NOT write to the TCP buffer.

5.8.11.6.1.27 HTTPPrint_varname Function

C

```
void HTTPPrint_varname(  
    HTTP_CONN_HANDLE connHandle,  
    uint16_t wParam1,  
    uint16_t wParam2,  
    ...  
);
```

Description

Functions in this style are implemented by the application developer in CustomHTTPApp.c. These functions generate dynamic content to be inserted into web pages and other files returned by the HTTP server.

Functions of this type are called when a dynamic variable is located in a web page. (ie, ~varname~) The name between the tilde '~' characters is appended to the base function name. In this example, the callback would be named HTTPPrint_varname.

The function prototype is located in your project's http_print.h, which is automatically generated by the MPFS2 Utility. The prototype will have uint16_t parameters included for each parameter passed in the dynamic variable. For example, the variable "~myArray(2,6)~" will generate the prototype "void HTTPPrint_varname(uint16_t, uint16_t);".

When called, this function should write its output directly to the TCP [socket](#) using any combination of [TCPIP_TCP_PutIsReady](#), [TCPIP_TCP_Put](#), [TCPIP_TCP_ArrayPut](#), [TCPIP_TCP_StringPut](#), [TCPIP_TCP_ArrayPut](#), and [TCPIP_TCP_StringPut](#).

Before calling, the HTTP server guarantees that at least [HTTP_MIN_CALLBACK_FREE](#) bytes (defaults to 16 bytes) are free in the output buffer. If the function is writing less than this amount, it should simply write the data to the [socket](#) and return.

In situations where a function needs to write more this amount, it must manage its output state using curHTTP.callbackPos. This value will be set to zero before the function is called. If the function is managing its output state, it must set this to a non-zero value before returning. Typically this is used to track how many bytes have been written, or how many remain to be written. If curHTTP.callbackPos is non-zero, the function will be called again when more buffer space is available. Once the callback completes, set this value back to zero to resume normal servicing of the request.

Preconditions

None

Parameters

Parameters	Description
connHandle	HTTP connection handle
wParam1	first parameter passed in the dynamic variable (if any)
wParam2	second parameter passed in the dynamic variable (if any)
...	additional parameters as necessary

Returns

None

Remarks

This function may service multiple HTTP requests simultaneously, especially when managing its output state. Exercise caution when using global or static variables inside this routine. Use curHTTP.callbackPos or curHTTP.data for storage associated with individual requests.

5.8.11.6.2 Data Types and Constants

5.8.11.6.2.1 HTTP_CONN_HANDLE Type

C

```
typedef const void* HTTP_CONN_HANDLE;
```

Description

HTTP connection identifier, handle of a HTTP connection

5.8.11.6.2.2 HTTP_FILE_TYPE Enumeration

C

```
typedef enum {  
    HTTP_TXT = 0u,  
    HTTP_HTM,  
    HTTP_HTML,  
    HTTP_CGI,  
    HTTP_XML,  
    HTTP_CSS,  
    HTTP_GIF,  
    HTTP_PNG,  
    HTTP_JPG,  
    HTTP_JAVA,  
    HTTP_WAV,  
    HTTP_UNKNOWN  
} HTTP_FILE_TYPE;
```

Description

File type definitions

Members

Members	Description
HTTP_TXT = 0u	File is a text document
HTTP_HTM	File is HTML (extension .htm)
HTTP_HTML	File is HTML (extension .html)
HTTP_CGI	File is HTML (extension .cgi)
HTTP_XML	File is XML (extension .xml)
HTTP_CSS	File is stylesheet (extension .css)
HTTP_GIF	File is GIF image (extension .gif)
HTTP_PNG	File is PNG image (extension .png)
HTTP_JPG	File is JPG image (extension .jpg)
HTTP_JAVA	File is java (extension .java)
HTTP_WAV	File is audio (extension .wav)
HTTP_UNKNOWN	File type is unknown

5.8.11.6.2.3 HTTP_IO_RESULT Enumeration

C

```
typedef enum {  
    HTTP_IO_DONE = 0u,  
    HTTP_IO_NEED_DATA,  
    HTTP_IO_WAITING  
} HTTP_IO_RESULT;
```

Description

Result states for execution callbacks

Members

Members	Description
HTTP_IO_DONE = 0u	Finished with procedure
HTTP_IO_NEED_DATA	More data needed to continue, call again later
HTTP_IO_WAITING	Waiting for asynchronous process to complete, call again later

5.8.11.6.2.4 HTTP_MODULE_CONFIG Structure**C**

```
typedef struct {
    uint16_t nConnections;
    uint16_t dataLen;
    uint16_t sktTxBuffSize;
    uint16_t sktRxBuffSize;
    uint16_t configFlags;
} HTTP_MODULE_CONFIG;
```

Description

HTTP module dynamic configuration data

Members

Members	Description
uint16_t nConnections;	number of http connections
uint16_t dataLen;	size of the data buffer for reading cookie and GET/POST arguments (bytes)
uint16_t sktTxBuffSize;	size of TX buffer for the associatted socket ; leave 0 for default
uint16_t sktRxBuffSize;	size of RX buffer for the associatted socket ; leave 0 for default
uint16_t configFlags;	a HTTP_MODULE_FLAGS value.

5.8.11.6.2.5 HTTP_MODULE_FLAGS Enumeration**C**

```
typedef enum {
    HTTP_MODULE_FLAG_ADJUST_SKT_FIFOS = 0x01,
    HTTP_MODULE_FLAG_NO_DELAY = 0x02
} HTTP_MODULE_FLAGS;
```

Description

HTTP module configuration flags

Members

Members	Description
HTTP_MODULE_FLAG_ADJUST_SKT_FIFOS = 0x01	adjust corresponding socket FIFO at run time improves throughput when the socket buffers are small
HTTP_MODULE_FLAG_NO_DELAY = 0x02	create the http sockets with NO-DELAY option it will flush data as soon as possible

5.8.11.6.2.6 HTTP_READ_STATUS Enumeration**C**

```
typedef enum {
```

```

    HTTP_READ_OK = 0u,
    HTTP_READ_TRUNCATED,
    HTTP_READ_INCOMPLETE
} HTTP_READ_STATUS;

```

Description

Result states for HTTPPostReadName and HTTPPostReadValue

Members

Members	Description
HTTP_READ_OK = 0u	Read was successful
HTTP_READ_TRUNCATED	Buffer overflow prevented by truncating value
HTTP_READ_INCOMPLETE	Entire object is not yet in the buffer. Try again later.

5.8.11.6.2.7 HTTP_STATUS Enumeration**C**

```

typedef enum {
    HTTP_GET = 0u,
    HTTP_POST,
    HTTP_BAD_REQUEST,
    HTTP_UNAUTHORIZED,
    HTTP_NOT_FOUND,
    HTTP_OVERFLOW,
    HTTP_INTERNAL_SERVER_ERROR,
    HTTP_NOT_IMPLEMENTED,
    HTTP_REDIRECT,
    HTTP_SSL_REQUIRED,
    HTTP_MPFS_FORM,
    HTTP_MPFS_UP,
    HTTP_MPFS_OK,
    HTTP_MPFS_ERROR
} HTTP_STATUS;

```

Description

Supported Commands and Server Response Codes

Members

Members	Description
HTTP_GET = 0u	GET command is being processed
HTTP_POST	POST command is being processed
HTTP_BAD_REQUEST	400 Bad Request will be returned
HTTP_UNAUTHORIZED	401 Unauthorized will be returned
HTTP_NOT_FOUND	404 Not Found will be returned
HTTP_OVERFLOW	414 Request-URI Too Long will be returned
HTTP_INTERNAL_SERVER_ERROR	500 Internal Server Error will be returned
HTTP_NOT_IMPLEMENTED	501 Not Implemented (not a GET or POST command)
HTTP_REDIRECT	302 Redirect will be returned
HTTP_SSL_REQUIRED	403 Forbidden is returned, indicating SSL is required
HTTP_MPFS_FORM	Show the MPFS Upload form
HTTP_MPFS_UP	An MPFS Upload is being processed
HTTP_MPFS_OK	An MPFS Upload was successful
HTTP_MPFS_ERROR	An MPFS Upload was not a valid image

5.8.11.6.2.8 HTTPReadPostPair Macro

C

```
#define HTTPReadPostPair(connHandle, cData, wLen) TCPIP_HTTP_PostValueRead(connHandle, cData, wLen)
```

Description

Reads a name and value pair from a URL encoded string in the TCP buffer. This function is meant to be called from an [HTTPExecutePost](#) callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates curHTTP.byteCount by decrementing it by the number of bytes read. It also removes the delimiting '&' from the buffer.

Once complete, two strings will exist in the cData buffer. The first is the parameter name that was read, while the second is the associated value.

Preconditions

Front of TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character and a value parameter terminated by a '&'.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name and value strings once they are read
wLen	how many bytes can be written to cData

Return Values

Return Values	Description
HTTP_READ_OK	name and value were successfully read
HTTP_READ_TRUNCATED	entire name and value could not fit in the buffer, so input was truncated and data has been lost
HTTP_READ_INCOMPLETE	entire name and value was not yet in the buffer, so call this function again later to retrieve

Remarks

This function is aliased to [TCPIP_HTTP_PostValueRead](#), since they effectively perform the same task. The name is provided only for completeness.

5.8.11.7 Files

Files

Name	Description
http.h	
http_config.h	HTTP configuration file

Description

5.8.11.7.1 http.h

HTTP Headers for Microchip TCP/IP Stack

Enumerations

	Name	Description
	HTTP_FILE_TYPE	File type definitions
	HTTP_IO_RESULT	Result states for execution callbacks
	HTTP_MODULE_FLAGS	HTTP module configuration flags
	HTTP_READ_STATUS	Result states for HTTPPostReadName and HTTPPostReadValue
	HTTP_STATUS	Supported Commands and Server Response Codes

Functions

	Name	Description
	HTTPCheckAuth	Performs validation on a specific user name and password.
	HTTPExecuteGet	Processes GET form field variables and cookies.
	HTTPExecutePost	Processes POST form variables and data.
	HTTPNeedsAuth	Determines if a given file name requires authentication
	HTTPPrint_varname	Inserts dynamic content into a web page
	TCPIP_HTTP_ActiveConnectionCountGet	Gets the number of active connections.
	TCPIP_HTTP_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_CurrentConnectionByteCountDec	Decrements byte count.
	TCPIP_HTTP_CurrentConnectionByteCountGet	Returns how many bytes have been read so far
	TCPIP_HTTP_CurrentConnectionByteCountSet	Sets how many bytes have been read so far
	TCPIP_HTTP_CurrentConnectionCallbackPosGet	Returns the callback position indicator.
	TCPIP_HTTP_CurrentConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_CurrentConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.
	TCPIP_HTTP_CurrentConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_CurrentConnectionHasArgsSet	Sets whether there are get or cookie arguments
	TCPIP_HTTP_CurrentConnectionIsAuthorizedGet	Get the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_CurrentConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_CurrentConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_CurrentConnectionStatusSet	Sets HTTP status.
	TCPIP_HTTP_CurrentConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_CurrentConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_FileInclude	Writes a file byte-for-byte to the currently loaded TCP socket .
	TCPIP_HTTP_PostNameRead	Reads a name from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_PostValueRead	Reads a value from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_URLDecode	Parses a string from URL encoding to plain-text.

Macros

	Name	Description
	HTTPReadPostPair	Reads a name and value pair from a URL encoded string in the TCP buffer.

Structures

	Name	Description
	HTTP_MODULE_CONFIG	HTTP module dynamic configuration data

Types

	Name	Description
	HTTP_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection

5.8.11.7.2 http_config.h

HyperText Transfer Protocol (HTTP) Configuration file

This file contains the HTTP module configuration options

Macros

	Name	Description
	HTTP_CACHE_LEN	Max lifetime (sec) of static responses as string
	HTTP_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default
	HTTP_DEFAULT_FILE	Indicate what file to serve when no specific one is requested
	HTTP_DEFAULT_LEN	For buffer overrun protection. Set to longest length of above two strings.
	HTTP_FILE_UPLOAD	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	HTTP_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	HTTP_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	HTTP_MAX_HEADER_LEN	Set to length of longest string above
	HTTP_MIN_CALLBACK_FREE	Define the minimum number of bytes free in the TX FIFO before executing callbacks
	HTTP_PORT	Define the listening port for the HTTP server
	HTTP_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the http socket Use 0 for default
	HTTP_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the http socket Use 0 for default
	HTTP_SSL_ONLY_CHAR	Files beginning with this character will only be served over HTTPS Set to 0x00 to require for all files Set to 0xff to require for no files
	HTTP_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	HTTP_USE_AUTHENTICATION	Enable basic authentication support
	HTTP_USE_COOKIES	Enable cookie support
	HTTP_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support
	HTTPS_DEFAULT_FILE	Indicate what file to serve when no specific one is requested
	HTTPS_PORT	Listening port for HTTPS server (when SSL enabled)
	TCPIP_STACK_USE_BASE64_DECODE	TCPIP_STACK_USE_BASE64_DECODE HTTP_USE_AUTHENTICATION

5.8.12 ICMP TCP/IP Stack Library

5.8.12.1 Introduction

Internet Control Message Protocol (ICMP) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the ICMP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The Internet Control Message Protocol is used to **send** error and status messages and requests. The ICMP module implements the Echo Reply message type (commonly referred to as a ping) which can be used to determine if a specified host is reachable across an IP network from a device running the TCP/IP stack. An ICMP server is also supported to respond to pings from other devices.

5.8.12.2 Release Notes

MPLAB Harmony Version: v0.70b **ICMP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.12.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.12.4 Using the Library

This topic describes the basic architecture of the ICMP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [icmp.h](#)

The interface to the ICMP TCP/IP Stack library is defined in the "[icmp.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the ICMP TCP/IP Stack library should include "tcpip.h".

Library File:

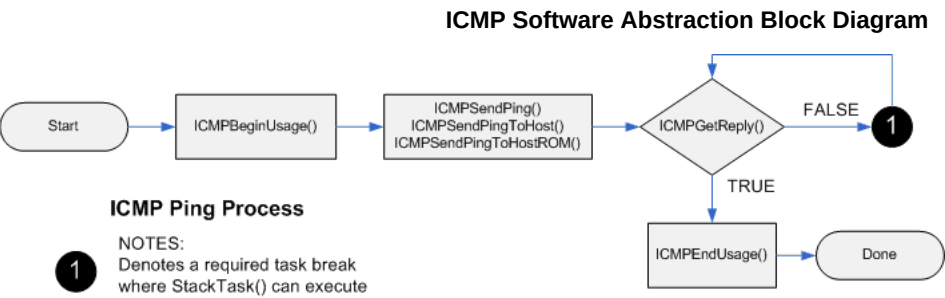
The ICMP TCP/IP Stack library archive (.a) file is installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.12.4.1 Abstraction Model

This library provides the API of the ICMP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description



5.8.12.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the ICMP module.

Library Interface Section	Description
Functions	This section provides general interface routines
Data Types and Constants	This section provides various definitions describing this API

5.8.12.5 Configuring the Library

Macros

	Name	Description
	ICMP_TIMEOUT	ICMP Timeout Value

Description

The configuration of the ICMP TCP/IP Stack is based on the file `tcip_config.h`.

This header file contains the configuration selection for the ICMP TCP/IP Stack. Based on the selections made, the ICMP TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the ICMP TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.12.5.1 ICMP_TIMEOUT Macro**C**

```
#define ICMP_TIMEOUT (4ul*SYS_TICK_TicksPerSecondGet())
```

Description

ICMP Timeout Value

5.8.12.6 Building the Library

This section list the files that are available in the `\src` of the ICMP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.12.7 Library Interface**Data Types and Constants**

	Name	Description
	ICMP_ECHO_RESULT	result of an <code>ICMPSendEchoRequest</code> call
	ICMP_HANDLE	a handle that a client can use after the event handler has been registered
	ICMP_MODULE_CONFIG	Provides a placeholder for ICMP module configuration.

Functions

	Name	Description
	TCPIP_ICMP_CallbackDeregister	De-registers the ICMP callback function.
	TCPIP_ICMP_CallbackRegister	Registers a callback to allow the application layer to process incoming ICMPv4 packets
	TCPIP_ICMP_EchoRequestSend	Sends an ICMP echo request to a remote node.

Description

This section describes the Application Programming Interface (API) functions of the ICMP TCP/IP Stack.

Refer to each section for a detailed description.

5.8.12.7.1 Functions

5.8.12.7.1.1 TCPIP_ICMP_CallbackDeregister Function

C

```
bool TCPIP_ICMP_CallbackDeregister(  
    ICMP_HANDLE hIcmp  
);
```

Description

This function a stack client to remove its former registered notification handler. After this operation the client will no longer be notified about the receiving of replies to the ICMP requests.

Preconditions

TCPIP Stack initialized, up and running. A previous successful call to [TCPIP_ICMP_CallbackRegister\(\)](#) has been done.

Parameters

Parameters	Description
hIcmp	an ICMP handle obtained by TCPIP_ICMP_CallbackRegister()

Returns

true - if the notification handler has been successfully removed false - if such notification handler couldn't be found

Remarks

None

Example

```
void MyICMPCallbackFunction(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data);  
ICMP_HANDLE hIcmp = TCPIP_ICMP_CallbackRegister(&MyICMPCallbackFunction);  
if(hIcmp != 0)  
{  
    // successfully registered my handler  
    // send requests and process the incoming results  
    // ...  
    // later on, once ew're done, remove the notification handler  
    TCPIP_ICMP_CallbackDeregister(hIcmp);  
}
```

5.8.12.7.1.2 TCPIP_ICMP_CallbackRegister Function

C

```
ICMP_HANDLE TCPIP_ICMP_CallbackRegister(  
    void (*callback)(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data)  
);
```

Description

Allows a stack client to be notified of the receiving of a response from an ICMP query. Once an Echo request reply is received, the notification handler callback will be called, letting the client know of the result of the query. The callback will contain as parameters:

- the network interface handle on which the query reply was received
- the remote host IP address
- a 32 bit value containing the sequence number in the low 16 bit part and the identifier value in the high 16 bit part.

Preconditions

TCPIP Stack initialized, up and running.

Parameters

Parameters	Description
callback	notification function to be registered This function will be called when an echo request reply is received.

Returns

- A non-null handle if the registration succeeded
- 0 if the registration operation failed (out of memory, for example).

Remarks

None

Example

```
// Callback function prototype
void MyICMPCallbackFunction(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data);

// *****
// Register callback function
ICMP_HANDLE hIcmp = TCPIP_ICMP_CallbackRegister(&MyICMPCallbackFunction);
if(hIcmp == 0)
{
    // process error; couldn't register a handler
}

// success; I can send an Echo request and receive notification

// *****
IPV4_ADDR remoteIP = 0xc0a00101;
uint16_t mySequenceNumber = 1;
uint16_t myId = 0x1234;
// send an ICMP query request
TCPIP_ICMP_EchoRequestSend(&remoteIP, mySequenceNumber, myId;

// *****
// process the ICMP reply in the callback function
void MyICMPCallbackFunction(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data)
{
    // process request from interface hNetIf and remoteIP address
    uint16_t* pReply = (uint16_t*)data;
    uint16_t myRecvId = *pReply;
    uint16_t myRecvSequenceNumber = *(pReply + 1);

    // check that the sequence number, ID and IP address match, etc.
}
```

5.8.12.7.1.3 TCPIP_ICMP_EchoRequestSend Function**C**

```
ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequestSend(
    IPV4_ADDR * targetAddr,
    uint16_t sequenceNumber,
    uint16_t identifier
);
```

Description

This function allows a stack client to [send](#) an ICMP query message to a remote host. The supplied sequence number and identifier will be used in the query message. The user will be notified by the result of the query using a notification handle registered by using the [TCPIP_ICMP_CallbackRegister](#) function.

Preconditions

TCPIP Stack initialized, up and running.

Parameters

Parameters	Description
targetAddr	remote host IP address.

Returns

true - if the query request was successfully sent false otherwise (interface not ready for transmission)

Remarks

None

Example

```
IPV4_ADDR remoteAddress = 0xc0a00101;
uint16_t mySequenceNumber = 1;
uint16_t myId = 0x1234;

if(TCPIP_ICMP_EchoRequestSend(&remoteAddress, mySequenceNumber, myId) )
{
    // successfully sent the ICMP request
    //
}
else
{
    // process the error
}
```

5.8.12.7.2 Data Types and Constants

5.8.12.7.2.1 ICMP_ECHO_RESULT Enumeration

C

```
typedef enum {
    ICMP_ECHO_OK = 0,
    ICMP_ECHO_ALLOC_ERROR = -1,
    ICMP_ECHO_ROUTE_ERROR = -2,
    ICMP_ECHO_TRANSMIT_ERROR = -3
} ICMP_ECHO_RESULT;
```

Description

result of an ICMPSendEchoRequest call

Members

Members	Description
ICMP_ECHO_OK = 0	operation successful error codes, < 0
ICMP_ECHO_ALLOC_ERROR = -1	could not allocate memory
ICMP_ECHO_ROUTE_ERROR = -2	could not find a route to destination
ICMP_ECHO_TRANSMIT_ERROR = -3	could not transmit (dead interface, etc.)

5.8.12.7.2.2 ICMP_HANDLE Type

C

```
typedef const void* ICMP_HANDLE;
```

Description

a handle that a client can use after the event handler has been registered

5.8.12.7.2.3 ICMP_MODULE_CONFIG Structure**C**

```
typedef struct {  
} ICMP_MODULE_CONFIG;
```

Description

ICMP Module Configuration Structure Typedef

Provides a placeholder for ICMP module configuration.

Remarks

None.

5.8.12.8 Files**Files**

Name	Description
icmp.h	
icmp_config.h	ICMP configuration file

Description**5.8.12.8.1 icmp.h**

ICMP Module Defs for Microchip TCP/IP Stack

Enumerations

	Name	Description
	ICMP_ECHO_RESULT	result of an ICMPSEndEchoRequest call

Functions

	Name	Description
	TCPIP_ICMP_CallbackDeregister	De-registers the ICMP callback function.
	TCPIP_ICMP_CallbackRegister	Registers a callback to allow the application layer to process incoming ICMPv4 packets
	TCPIP_ICMP_EchoRequestSend	Sends an ICMP echo request to a remote node.

Structures

	Name	Description
	ICMP_MODULE_CONFIG	Provides a placeholder for ICMP module configuration.

Types

	Name	Description
	ICMP_HANDLE	a handle that a client can use after the event handler has been registered

5.8.12.8.2 icmp_config.h

Internet Control Message Protocol (ICMP) Configuration file

This file contains the ICMP module configuration options

Macros

	Name	Description
	<code>ICMP_TIMEOUT</code>	ICMP Timeout Value

5.8.13 ICMPv6 TCP/IP Stack Library

5.8.13.1 Introduction

ICMPv6 TCP/IP Library

for

Microchip Microcontrollers

This library provides the API of the ICMPv6 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

This document specifies a set of APIs of Internet Control Message Protocol (ICMP) messages for use with version 6 of the Internet Protocol (IPv6).

The Internet Protocol, version 6 (IPv6) is a new version of IP, uses the Internet Control Message Protocol (ICMP) as defined for IPv4

[RFC-792], with a number of changes. It is called ICMPv6, and has an IPv6 Next Header value of 58.

ICMPv6 messages are used by IPv6 nodes to report *error messages* and *information messages*. ICMPv6 is also used for Ipv6 node diagnostic (i.e., IPv6 ping).

The ICMPv6 protocol also provides a framework for the following:

Neighbor Discovery

Neighbor Discovery is a series of five ICMPv6 messages that manage node-to-node communication on a link. Neighbor Discovery replaces Address Resolution Protocol (ARP), ICMP(IPv4) Router Discovery, and the ICMP(IPv4) Redirect message.

Multicast Listener Discovery (MLD)

Multicast Listener Discovery is a series of three ICMP messages that manage subnet multicast membership. Multicast Listener Discovery replaces version 2 of the Internet Group Management Protocol (IGMP) for IPv4.

Path MTU Discovery

The maximum transmission unit (MTU) for a path is the minimum link MTU of all links on a path between a source and a destination. IPv6 packets that are smaller than the path MTU do not require fragmentation by the host and are successfully forwarded by all routers on the path. To discover the path MTU, the sending host uses the receipt of **ICMPv6 Packet Too Big**

messages.

ICMPv6 Common Messages	Functions
Echo Request	Sent to check IPv6 connectivity to a particular host.
Echo Reply	Sent in response to an ICMPv6 Echo Request.
Destination Unreachable	Sent by a router or the destination host to inform the sending host that the packet or payload cannot be delivered.
Packet too big	Sent by a router to inform a sending host that a packet is too large to forward.
Time exceeded	Sent by a router to inform a sending host that the Hop Limit of an IPv6 packet has expired.
Parameter Problem	Sent by a router to inform a sending host that an error was encountered in processing the IPv6 header or an IPv6 extension header.

5.8.13.2 Release Notes

MPLAB Harmony Version: v0.70b **ICMPv6 TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.13.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.13.4 Using the Library

This topic describes the basic architecture of the ICMPv6 TCP/IP Library and provides information and examples on how to use it.

Interface Header File: [icmpv6.h](#)

The interface to the ICMPv6 TCP/IP library is defined in the "[icmpv6.h](#)" header file. Any C language source (.c) file that uses the ICMPv6 TCP/IP library should include "IPv6.h", [icmpv6.h](#) and [ndp.h](#)".

Library File:

The IP TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.13.4.1 Abstraction Model

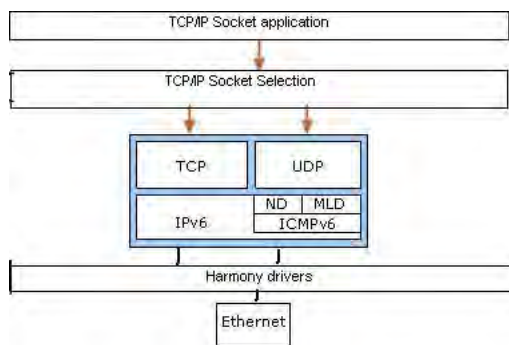
This library provides the API of the ICMPv6 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

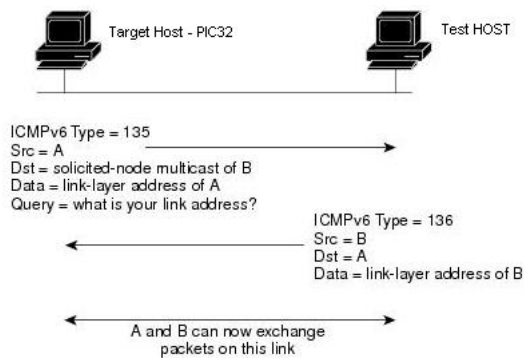
ICMPv6 Software Abstraction Block Diagram

This module provides software abstraction of the IPv6 module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

IPv6 Block Diagram



ICMPv6 message communication



5.8.13.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the IPv6 and ICMPv6 module.

Library Interface Section	Description
Functions	This section provides general interface routines
Data Types and Constants	This section provides various definitions describing this API

5.8.13.5 Configuring the Library

Macros

	Name	Description
	ICMPV6_TIMEOUT	ICMPv6 Timeout Value

Description

The configuration of the ICMPv6 TCP/IP Stack is based on the file [icmpv6_config.h](#)

This header file contains the configuration selection for the ICMPv6 TCP/IP Stack. Based on the selections made, the ICMPv6 TCP/IP Stack IP will support or not support selected features. These configuration settings will apply to all instances of the ICMPv6 TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build.

5.8.13.5.1 ICMPV6_TIMEOUT Macro

C

```
#define ICMPV6_TIMEOUT (4ul*SYS_TICK_ResolutionGet())
```

Description

ICMPv6 Timeout Value

5.8.13.6 Building the Library

This section list the files that are available in the \src of the IPv6 and ICMPv6 driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- ipv6.c
- icmpv6.c
- ndp.c

TCPIP_STACK_USE_IPV6 is enabled from tcpip_config.h .

5.8.13.7 Library Interface

Data Types and Constants

	Name	Description
	ICMPV6_ERR_PTB_CODE	Definition for ICMPv6 Packet Too Big error code
	ICMPV6_INFO_EREQ_CODE	Definition for ICMPv6 Packet Echo Request info code
	ICMPV6_INFO_ERPL_CODE	Definition for ICMPv6 Packet Echo Reply info code

	TCPIP_ICMPV6_PutHeaderEchoReply	This is macro TCPIP_ICMPV6_PutHeaderEchoReply.
	ICMPV6_ERR_DU_CODE	Definitions for ICMPv6 Destination Unreachable error code
	ICMPV6_ERR_PP_CODE	Definitions for ICMPv6 Parameter Problem error code
	ICMPV6_ERR_TE_CODE	Definitions for ICMPv6 Time Exceeded error code
	ICMPV6_HANDLE	a handle that a client can use after the event handler has been registered
	ICMPV6_HEADER_ECHO	Header for an ICMPv6 Echo Request/Reply packet
	ICMPV6_HEADER_ERROR	Header for an ICMPv6 Error packet
	ICMPV6_PACKET_TYPES	ICMPv6 packet types

Functions

	Name	Description
	TCPIP_ICMPV6_CallbackDeregister	Deregisters an upper-layer function from ICMPv6 callback.
	TCPIP_ICMPV6_CallbackRegister	Registers an upper-layer function for ICMPv6 callback.
	TCPIP_ICMPV6_Close	This is function TCPIP_ICMPV6_Close.
	TCPIP_ICMPV6_Flush	Flushes an ICMPv6 Packet
	TCPIP_ICMPV6_HeaderEchoRequestPut	Allocates a packet, IPv6 Header, and Upper-layer header for an ICMPv6 echo request.

Description

This section describes the Application Programming Interface (API) functions of the ICMPv6 TCP/IP Stack Library.

Refer to each section for a detailed description.

5.8.13.7.1 Functions

5.8.13.7.1.1 TCPIP_ICMPV6_CallbackDeregister Function

C

```
bool TCPIP_ICMPV6_CallbackDeregister(
    ICMPV6_HANDLE hIcmpv6
);
```

Description

deregisters an upper-layer function from the ICMPv6 register list.

Preconditions

None

Parameters

Parameters	Description
hIcmpv6	ICMPv6 handler

Returns

true or false

Remarks

None

5.8.13.7.1.2 TCPIP_ICMPV6_CallbackRegister Function

```
C
ICMPV6_HANDLE TCPIP_ICMPV6_CallbackRegister(
    void (*callback)(TCPIP_NET_HANDLE hNetIf, uint8_t type, IPV6_ADDR * localIP, IPV6_ADDR *
remoteIP, void * header)
);
```

Description

Registers an upper-layer function to handle ICMPv6 messages that may require action at the application layer (Echo Replies, Error messages)

Preconditions

None

Parameters

Parameters	Description
type	ICMPv6 header type
localIP	IPv6 destination address of the incoming message
remoteIP	IPv6 address of the node that originated the incoming message
header	Pointer to the ICMPv6 header

Returns

A **ICMPV6_HANDLE**

Remarks

None

5.8.13.7.1.3 TCPIP_ICMPV6_Close Function

```
C
void TCPIP_ICMPV6_Close(
    IPV6_PACKET * pkt
);
```

Description

This is function TCPIP_ICMPV6_Close.

5.8.13.7.1.4 TCPIP_ICMPV6_Flush Function

```
C
bool TCPIP_ICMPV6_Flush(
    IPV6_PACKET * pkt
);
```

Description

Flushes an ICMPv6 Packet

Preconditions

None

Parameters

Parameters	Description
pkt	The packet to flush

Returns

true if the packet was flushed, false if the packet was queued

Remarks

None

5.8.13.7.1.5 TCPIP_ICMPV6_HeaderEchoRequestPut Function**C**

```
IPV6_PACKET * TCPIP_ICMPV6_HeaderEchoRequestPut(  
    TCPIP_NET_HANDLE hNetIf,  
    IPV6_ADDR * localIP,  
    IPV6_ADDR * remoteIP,  
    uint8_t type,  
    uint16_t identifier,  
    uint16_t sequenceNumber  
);
```

Description

Allocates a packet, IPv6 Header, and Upper-layer header for an ICMPv6 echo request.

Preconditions

None

Parameters

Parameters	Description
pNetIf	The interface for the outgoing packet.
localIP	The local address that should be used for this packet.
remoteIP	The packet's destination address
type	Echo Request or Echo Reply
identifier	The Echo Request id.
sequenceNumber	The Echo request sequence number

Returns

IPV6_PACKET * - The constructed error packet or NULL

Remarks

None

5.8.13.7.2 Data Types and Constants**5.8.13.7.2.1 ICMPV6_ERR_PTB_CODE Macro****C**

```
#define ICMPV6_ERR_PTB_CODE 0u
```

Description

Definition for ICMPv6 Packet Too Big error code

5.8.13.7.2.2 ICMPV6_INFO_EREQ_CODE Macro

C

```
#define ICMPV6_INFO_EREQ_CODE 0u
```

Description

Definition for ICMPv6 Packet Echo Request info code

5.8.13.7.2.3 ICMPV6_INFO_ERPL_CODE Macro

C

```
#define ICMPV6_INFO_ERPL_CODE 0u
```

Description

Definition for ICMPv6 Packet Echo Reply info code

5.8.13.7.2.4 TCPIP_ICMPV6_PutHeaderEchoReply Macro

C

```
#define TCPIP_ICMPV6_PutHeaderEchoReply TCPIP_ICMPV6_HeaderEchoRequestPut
```

Description

This is macro TCPIP_ICMPV6_PutHeaderEchoReply.

5.8.13.7.2.5 ICMPV6_ERR_DU_CODE Enumeration

C

```
typedef enum {
    ICMPV6_ERR_DU_NO_ROUTE = 0u,
    ICMPV6_ERR_DU_PROHIBITED = 1u,
    ICMPV6_ERR_DU_OUTSIDE_SCOPE = 2u,
    ICMPV6_ERR_DU_ADDR_UNREACHABLE = 3u,
    ICMPV6_ERR_DU_PORT_UNREACHABLE = 4u,
    ICMPV6_ERR_DU_SRC_FAILED_INGRESS_POLICY = 5u,
    ICMPV6_ERR_DU_REJECT_ROUTE = 6u
} ICMPV6_ERR_DU_CODE;
```

Description

Definitions for ICMPv6 Destination Unreachable error code

Members

Members	Description
ICMPV6_ERR_DU_NO_ROUTE = 0u	No route to destination
ICMPV6_ERR_DU_PROHIBITED = 1u	Communication with destination administratively prohibited
ICMPV6_ERR_DU_OUTSIDE_SCOPE = 2u	Beyond scope of source address
ICMPV6_ERR_DU_ADDR_UNREACHABLE = 3u	Address unreachable
ICMPV6_ERR_DU_PORT_UNREACHABLE = 4u	Port unreachable

ICMPV6_ERR_DU_SRC_FAILED_INGRESS_POLICY = 5u	Source address failed ingress/egress policy
ICMPV6_ERR_DU_REJECT_ROUTE = 6u	Reject route to destination

5.8.13.7.2.6 ICMPV6_ERR_PP_CODE Enumeration

C

```
typedef enum {
    ICMPV6_ERR_PP_ERRONEOUS_HEADER = 0u,
    ICMPV6_ERR_PP_UNRECOGNIZED_NEXT_HEADER = 1u,
    ICMPV6_ERR_PP_UNRECOGNIZED_IPV6_OPTION = 2u
} ICMPV6_ERR_PP_CODE;
```

Description

Definitions for ICMPv6 Parameter Problem error code

Members

Members	Description
ICMPV6_ERR_PP_ERRONEOUS_HEADER = 0u	Erroneous header field encountered
ICMPV6_ERR_PP_UNRECOGNIZED_NEXT_HEADER = 1u	Unrecognized Next Header type encountered
ICMPV6_ERR_PP_UNRECOGNIZED_IPV6_OPTION = 2u	Unrecognized IPv6 option encountered

5.8.13.7.2.7 ICMPV6_ERR_TE_CODE Enumeration

C

```
typedef enum {
    ICMPV6_ERR_TE_HOP_LIMIT_EXCEEDED = 0u,
    ICMPV6_ERR_TE_FRAG_ASSEMBLY_TIME_EXCEEDED = 1u
} ICMPV6_ERR_TE_CODE;
```

Description

Definitions for ICMPv6 Time Exceeded error code

Members

Members	Description
ICMPV6_ERR_TE_HOP_LIMIT_EXCEEDED = 0u	Hop limit exceeded in transit
ICMPV6_ERR_TE_FRAG_ASSEMBLY_TIME_EXCEEDED = 1u	Fragment reassembly time exceeded

5.8.13.7.2.8 ICMPV6_HANDLE Type

C

```
typedef const void* ICMPV6_HANDLE;
```

Description

a handle that a client can use after the event handler has been registered

5.8.13.7.2.9 ICMPV6_HEADER_ECHO Structure

C

```
typedef struct {
    uint8_t vType;
    uint8_t vCode;
    uint16_t wChecksum;
    uint16_t identifier;
    uint16_t sequenceNumber;
} ICMPV6_HEADER_ECHO;
```

Description

Header for an ICMPv6 Echo Request/Reply packet

Members

Members	Description
uint8_t vType;	icmpV6 request or reply type
uint8_t vCode;	Error code
uint16_t wChecksum;	Packet TX and RX checksum
uint16_t identifier;	incoming and outgoing packet identifier
uint16_t sequenceNumber;	request and reply sequence number

5.8.13.7.2.10 ICMPV6_HEADER_ERROR Structure

C

```
typedef struct {
    uint8_t vType;
    uint8_t vCode;
    uint16_t wChecksum;
    uint32_t additionalData;
} ICMPV6_HEADER_ERROR;
```

Description

Header for an ICMPv6 Error packet

Members

Members	Description
uint8_t vType;	icmpV6 request or reply type
uint8_t vCode;	error code
uint16_t wChecksum;	Packet TX and RX checksum
uint32_t additionalData;	Unused for Dest. Unreachable and Time Exceeded. MTU for MTU. Pointer for Parameter Problem.

5.8.13.7.2.11 ICMPV6_PACKET_TYPES Enumeration

C

```
typedef enum {
    ICMPV6_ERROR_DEST_UNREACHABLE = 1u,
    ICMPV6_ERROR_PACKET_TOO_BIG = 2u,
    ICMPV6_ERROR_TIME_EXCEEDED = 3u,
    ICMPV6_ERROR_PARAMETER_PROBLEM = 4u,
    ICMPV6_INFO_ECHO_REQUEST = 128u,
    ICMPV6_INFO_ECHO_REPLY = 129u,
    ICMPV6_INFO_ROUTER_SOLICITATION = 133u,
    ICMPV6_INFO_ROUTER_ADVERTISEMENT = 134u,
```

```

ICMPV6_INFO_NEIGHBOR_SOLICITATION = 135u,
ICMPV6_INFO_NEIGHBOR_ADVERTISEMENT = 136u,
ICMPV6_INFO_REDIRECT = 137u
} ICMPV6_PACKET_TYPES;

```

Description

ICMPv6 packet types

Members

Members	Description
ICMPV6_ERROR_DEST_UNREACHABLE = 1u	Destination Unreachable error packet
ICMPV6_ERROR_PACKET_TOO_BIG = 2u	Packet Too Big error packet
ICMPV6_ERROR_TIME_EXCEEDED = 3u	Time Exceeded error packet
ICMPV6_ERROR_PARAMETER_PROBLEM = 4u	Parameter Problem error packet
ICMPV6_INFO_ECHO_REQUEST = 128u	Echo Request packet
ICMPV6_INFO_ECHO_REPLY = 129u	Echo Reply packet
ICMPV6_INFO_ROUTER_SOLICITATION = 133u	Router solicitation NDP packet
ICMPV6_INFO_ROUTER_ADVERTISEMENT = 134u	Router advertisement NDP packet
ICMPV6_INFO_NEIGHBOR_SOLICITATION = 135u	Neighbor Solicitation NDP packet
ICMPV6_INFO_NEIGHBOR_ADVERTISEMENT = 136u	Neighbor Advertisement NDP packet
ICMPV6_INFO_REDIRECT = 137u	Redirect NDP packet

5.8.13.8 Files

Files

Name	Description
icmpv6.h	IPv6 Internet Communication Message Protocol
icmpv6_config.h	ICMPv6 configuration file

Description

5.8.13.8.1 icmpv6.h

Internet Control Message Protocol (ICMP) in IPv6 functions the same as ICMP in IPv4. ICMPv6 is used to report error messages and information messages for IPv6 nodes. ICMP packets in IPv6 are used in the IPv6 neighbor discovery process, path MTU discovery, and the Multicast Listener Discovery (MLD) protocol for IPv6.

Enumerations

	Name	Description
	ICMPV6_ERR_DU_CODE	Definitions for ICMPv6 Destination Unreachable error code
	ICMPV6_ERR_PP_CODE	Definitions for ICMPv6 Parameter Problem error code
	ICMPV6_ERR_TE_CODE	Definitions for ICMPv6 Time Exceeded error code

**ICMPV6_PACKET_TYPES**

ICMPv6 packet types

Functions

	Name	Description
	TCPIP_ICMPV6_CallbackDeregister	Deregisters an upper-layer function from ICMPv6 callback.
	TCPIP_ICMPV6_CallbackRegister	Registers an upper-layer function for ICMPv6 callback.
	TCPIP_ICMPV6_Close	This is function TCPIP_ICMPV6_Close.
	TCPIP_ICMPV6_Flush	Flushes an ICMPv6 Packet
	TCPIP_ICMPV6_HeaderEchoRequestPut	Allocates a packet, IPv6 Header, and Upper-layer header for an ICMPv6 echo request.

Macros

	Name	Description
	ICMPV6_ERR_PTB_CODE	Definition for ICMPv6 Packet Too Big error code
	ICMPV6_INFO_EREQ_CODE	Definition for ICMPv6 Packet Echo Request info code
	ICMPV6_INFO_ERPL_CODE	Definition for ICMPv6 Packet Echo Reply info code
	TCPIP_ICMPV6_PutHeaderEchoReply	This is macro TCPIP_ICMPV6_PutHeaderEchoReply.

Structures

	Name	Description
	ICMPV6_HEADER_ECHO	Header for an ICMPv6 Echo Request/Reply packet
	ICMPV6_HEADER_ERROR	Header for an ICMPv6 Error packet

Types

	Name	Description
	ICMPV6_HANDLE	a handle that a client can use after the event handler has been registered

File Name

icmpv6.h

Company

Microchip Technology Incorporated

5.8.13.8.2 icmpv6_config.h

Internet Control Message Protocol for IPv6 (ICMPv6) Configuration file

This file contains the ICMPv6 module configuration options

Macros

	Name	Description
	ICMPV6_TIMEOUT	ICMPv6 Timeout Value

5.8.14 IPv4 TCP/IP Stack Library

5.8.14.1 Introduction

Internet Protocol (IP) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the IP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

IP is the workhorse protocol of the TCP/IP protocol suite. All TCP, UDP, ICMP, and IGMP data gets transmitted as IP datagrams (Figure 1.4). IP provides an unreliable, connectionless datagram delivery service.

IP provides a best effort service. When something goes wrong, such as a router temporarily running out of buffers, IP has a simple error handling algorithm: throw away the datagram and try to **send** an ICMP message back to the source. Any required reliability must be provided by the upper layers (e.g., TCP).

The term connectionless means that IP does not maintain any state information about successive datagrams. Each datagram is handled independently from all other datagrams. This also means that IP datagrams can get delivered out of order. If a source sends two consecutive datagrams (first A, then B) to the same destination, each is routed independently and can take different routes, with B arriving before A.

5.8.14.2 Release Notes

MPLAB Harmony Version: v0.70b **IPv4 TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.14.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.14.4 Using the Library

This topic describes the basic architecture of the IP TCPIP Stack Library and provides information and examples on how to use it.

Interface Header File: ip.h

The interface to the IPv4 TCP/IP Stack library is defined in the "ip.h" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the IPv4 TCP/IP Stack library should include "tcpip.h".

Library File:

The IPv4 TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.14.4.1 Abstraction Model

This library provides the API of the IPv4 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

IPv4 Software Abstraction Block Diagram

This module provides software abstraction of the IPv4 module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

Note: This diagram was not available at time of release and will be added in a future version of MPLAB Harmony.

5.8.14.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the IP module.

Library Interface Section	Description
Functions	Routines for configuring IPv4
Data Types and Constants	This section provides various definitions describing this API

5.8.14.5 Configuring the Library

Macros

	Name	Description
	IP_DEFAULT_ALLOCATION_BLOCK_SIZE	Used to configure payload size
	IPV4_QUEUED_PACKET_LIMIT	This option defines the maximum number of queued IPv4 TX packets. If an additional packet is queued, the oldest packet in the queue will be removed.

	IPV4_QUEUED_PACKET_TIMEOUT	This option defines the number of seconds an IPv4 TX packet will remain in the queue before being timed out
	IPV4_TASK_PROCESS_RATE	default 1 second

Description

The configuration of the IPv4 TCP/IP Stack is based on the file sys_config.h

This header file contains the configuration selection for the IPv4 TCP/IP Stack. Based on the selections made, the IPv4 TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the IPv4 TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.14.5.1 IP_DEFAULT_ALLOCATION_BLOCK_SIZE Macro**C**

```
#define IP_DEFAULT_ALLOCATION_BLOCK_SIZE (64u)
```

Description

Used to configure payload size

5.8.14.5.2 IPV4_QUEUED_PACKET_LIMIT Macro**C**

```
#define IPV4_QUEUED_PACKET_LIMIT 4u
```

Description

This option defines the maximum number of queued IPv4 TX packets. If an additional packet is queued, the oldest packet in the queue will be removed.

5.8.14.5.3 IPV4_QUEUED_PACKET_TIMEOUT Macro**C**

```
#define IPV4_QUEUED_PACKET_TIMEOUT 10u
```

Description

This option defines the number of seconds an IPv4 TX packet will remain in the queue before being timed out

5.8.14.5.4 IPV4_TASK_PROCESS_RATE Macro**C**

```
#define IPV4_TASK_PROCESS_RATE (1000) // default 1 second
```

Description

default 1 second

5.8.14.6 Building the Library

This section list the files that are available in the \src of the IP driver. It lists which files need to be included in the build based on

either a hardware feature present on the board or configuration option selected by the system.

- ip.c

5.8.14.7 Library Interface

Data Types and Constants

	Name	Description
	IPV4_HEADER	None
	IPV4_HEADER_TYPE	None
	IPV4_MODULE_CONFIG	
	IPV4_PACKET	Packet structure/state tracking for IPv4 packets

Functions

	Name	Description
	TCPIP_IPV4_PacketFormatTx	Formats an IPv4 packet and makes it ready for transmission.
	TCPIP_IPV4_PacketGetDestAddress	Returns the IPv4 destination address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketGetSourceAddress	Returns the IPv4 source address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketTransmit	Transmits an IPv4 packet over the network.
	TCPIP_IPV4_SelectSourceInterface	Selects a source address and an interface based on the IPv4 destination address

Description

This section describes the Application Programming Interface (API) functions of the IPv4 TCP/IP Stack Library.

Refer to each section for a detailed description.

5.8.14.7.1 Functions

5.8.14.7.1.1 TCPIP_IPV4_PacketFormatTx Function

C

```
void TCPIP_IPV4_PacketFormatTx(
    IPV4_PACKET* pPkt,
    uint8_t protocol,
    uint16_t ipLen
);
```

Description

The necessary fields are set into the IPv4 packet.

Preconditions

Properly allocated pPkt. The source and destination addresses should be updated in the packet.

Parameters

Parameters	Description
pPkt	the packet to be formatted
protocol	the protocol associated with the packet

ipLoadLen	the IPv4 packet payload length

Returns

None

Remarks

The segments should be properly updated with the right number of bytes (segLen). The ipLoadLen is added only to the 1st segment of the packet! Other segments (for packets having multiple packets) are not touched.

5.8.14.7.1.2 TCPIP_IPV4_PacketGetDestAddress Function

C

```
__inline__ const IPV4_ADDR* TCPIP_IPV4_PacketGetDestAddress(  
    TCPIP_MAC_PACKET* pPkt  
);
```

Description

The function will return a pointer to where the IPv4 destination address is located in the [TCPIP_MAC_PACKET](#). The [TCPIP_MAC_PACKET](#) is supposed to be a valid IPv4 packet that has properly destination address set.

Preconditions

pPkt - valid IPv4 packet, pNetLayer filed properly set

Parameters

Parameters	Description
pPkt	packet to query

Returns

a valid pointer to an IPV4_ADDR if it succeeds 0 - if call failed

Remarks

This function is mainly meant for RX packets.

5.8.14.7.1.3 TCPIP_IPV4_PacketGetSourceAddress Function

C

```
__inline__ const IPV4_ADDR* TCPIP_IPV4_PacketGetSourceAddress(  
    TCPIP_MAC_PACKET* pPkt  
);
```

Description

The function will return a pointer to where the IPv4 source address is located in the [TCPIP_MAC_PACKET](#). The [TCPIP_MAC_PACKET](#) is supposed to be a valid IPv4 packet that has properly source address set.

Preconditions

pPkt - valid IPv4 packet, pNetLayer filed properly set

Parameters

Parameters	Description
pPkt	packet to query

Returns

a valid pointer to an IPV4_ADDR if it succeeds 0 - if call failed

Remarks

This function is mainly meant for RX packets.

5.8.14.7.1.4 TCPIP_IPV4_PacketTransmit Function**C**

```
bool TCPIP_IPV4_PacketTransmit(  
    IPV4_PACKET* pPkt  
);
```

Description

The IPv4 packet is sent to the MAC for transmission.

Preconditions

pPkt should have been properly formatted with [TCPIP_IPV4_PacketFormatTx\(\)](#). The packet interface should be updated.

Parameters

Parameters	Description
pPkt	the packet to be transmitted

Returns

true - if the packet was handed to the MAC or is queued for transmission
false - the packet cannot be transmitted (wrong interface, etc.)

Remarks

Only single packets can be transmitted. Chained packets are not supported for now.

5.8.14.7.1.5 TCPIP_IPV4_SelectSourceInterface Function**C**

```
TCPIP_NET_HANDLE TCPIP_IPV4_SelectSourceInterface(  
    TCPIP_NET_HANDLE netH,  
    IPV4_ADDR* pDestAddress,  
    IPV4_ADDR* pSrcAddress,  
    bool srcSet  
);
```

Description

Updates the pSrcAddress and returns the needed interface, if successful if srcSet == 1 and netH != 0 the function won't change anything if srcSet == 1 and netH == 0 then the call will never fail it will use whatever value in pSrcAddress (even 0) and will try to come up with an appropriate interface. if srcSet == 0 and netH == 0 it will use the destination address if srcSet == 0 and netH != 0 it will use the address of that interface

Preconditions

netH has to be valid (if non 0)

Parameters

Parameters	Description
netH	network interface handle
pDestAddress	pointer to destination address
pSrcAddress	pointer to source address
srcSet	boolean; true if address pointed by pSrcAddress is valid

Returns

a valid interface - if it succeeds and a valid source interface selected 0 - interface selection failed

Remarks

None

5.8.14.7.2 Data Types and Constants

5.8.14.7.2.1 IPV4_HEADER Structure

C

```
typedef struct {  
    uint8_t VersionIHL;  
    uint8_t TypeOfService;  
    uint16_t TotalLength;  
    uint16_t Identification;  
    uint16_t FragmentInfo;  
    uint8_t TimeToLive;  
    uint8_t Protocol;  
    uint16_t HeaderChecksum;  
    IPV4_ADDR SourceAddress;  
    IPV4_ADDR DestAddress;  
} IPV4_HEADER;
```

Description

IPv4 packet header definition

None

Remarks

None

5.8.14.7.2.2 IPV4_HEADER_TYPE Enumeration

C

```
typedef enum {  
    IP_PROT_ICMP = (1u),  
    IP_PROT_TCP = (6u),  
    IP_PROT_UDP = (17u)  
} IPV4_HEADER_TYPE;
```

Description

IPv4 supported protocols

None

Remarks

None

5.8.14.7.2.3 IPV4_MODULE_CONFIG Structure

C

```
typedef struct {  
} IPV4_MODULE_CONFIG;
```

5.8.14.7.2.4 IPV4_PACKET Structure

C

```
typedef struct {
    TCPIP_MAC_PACKET macPkt;
    IPV4_ADDR srcAddress;
    IPV4_ADDR destAddress;
    TCPIP_NET_HANDLE netIfH;
    IPV4_ADDR arpTarget;
} IPV4_PACKET;
```

Description

Packet structure/state tracking for IPv4 packets

Members

Members	Description
TCPIP_MAC_PACKET macPkt;	standard MAC packet header safe cast to TCPIP_MAC_PACKET
IPV4_ADDR srcAddress;	packet source
IPV4_ADDR destAddress;	packet destination
TCPIP_NET_HANDLE netIfH;	packet interface
IPV4_ADDR arpTarget;	ARP resolution target

5.8.14.8 Files

Files

Name	Description
ip_config.h	IP configuration file
ipv4.h	

Description

5.8.14.8.1 ip_config.h

Internet Protocol (IP) Configuration file

This file contains the IP module configuration options

Macros

	Name	Description
	IP_DEFAULT_ALLOCATION_BLOCK_SIZE	Used to configure payload size
	IPV4_QUEUED_PACKET_LIMIT	This option defines the maximum number of queued IPv4 TX packets. If an additional packet is queued, the oldest packet in the queue will be removed.
	IPV4_QUEUED_PACKET_TIMEOUT	This option defines the number of seconds an IPv4 TX packet will remain in the queue before being timed out
	IPV4_TASK_PROCESS_RATE	default 1 second
	IPV6_DEFAULT_BASE_REACHABLE_TIME	30 seconds
	IPV6_DEFAULT_CUR_HOP_LIMIT	(IPv4 Time-to-Live parameter)
	IPV6_DEFAULT_LINK_MTU	Default Maximum Transmission Unit

	IPV6_DEFAULT_RETRANSMIT_TIME	1 second
	IPV6_INIT_TASK_PROCESS_RATE	default 32 ms
	IPV6_MINIMUM_LINK_MTU	Sets the lower bounds of the the Maximum Transmission Unit
	IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT	10 minutes
	IPV6_QUEUE_MCAST_PACKET_LIMIT	This option defines the maximum number of multicast queued IPv6. If an additional packet is queued, the oldest packet in the queue will be removed.
	IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT	This option defines the maximum number of queued packets per remote. If an additional packet needs to be queued, the oldest packet in the queue will be removed.
	IPV6_QUEUED_MCAST_PACKET_TIMEOUT	This option defines the number of seconds an IPv6 multicast packet will remain in the queue before being timed out
	IPV6_TASK_PROCESS_RATE	default 1 second
	IPV6_ULA_NTP_ACCESS_TMO	NTP access timeout for the IPv6 ULA address generation, ms
	IPV6_ULA_NTP_VALID_WINDOW	the NTP time stamp validity window, ms if a stamp was obtained outside this interval from the moment of the request a new request will be issued

5.8.14.8.2 ipv4.h

IPv4 Defs for Microchip TCP/IP Stack

Enumerations

	Name	Description
	IPV4_HEADER_TYPE	None

Functions

	Name	Description
	TCPIP_IPV4_PacketFormatTx	Formats an IPV4 packet and makes it ready for transmission.
	TCPIP_IPV4_PacketGetDestAddress	Returns the IPv4 destination address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketGetSourceAddress	Returns the IPv4 source address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketTransmit	Transmits an IPV4 packet over the network.
	TCPIP_IPV4_SelectSourceInterface	Selects a source address and an interface based on the IPv4 destination address

Structures

	Name	Description
	IPV4_HEADER	None
	IPV4_MODULE_CONFIG	
	IPV4_PACKET	Packet structure/state tracking for IPv4 packets

5.8.15 IPv6 TCP/IP Stack Library

5.8.15.1 Introduction

Internet Protocol (IP) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the IPv6 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

IPv6 is the workhorse protocol of the TCP/IP protocol suite. All TCP, UDP, ICMP, and IGMP data gets transmitted as IP datagrams (Figure 1.4). IP provides an unreliable, connectionless datagram delivery service.

IPv6 provides a best effort service. When something goes wrong, such as a router temporarily running out of buffers, IPv6 has a simple error handling algorithm: throw away the datagram and try to **send** an ICMP message back to the source. Any required reliability must be provided by the upper layers (e.g., TCP).

The term connectionless means that IPv6 does not maintain any state information about successive datagrams. Each datagram is handled independently from all other datagrams. This also means that IPv6 datagrams can get delivered out of order. If a source sends two consecutive datagrams (first A, then B) to the same destination, each is routed independently and can take different routes, with B arriving before A.

5.8.15.2 Release Notes

MPLAB Harmony Version: v0.70b **IPv6 TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.15.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO

YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.15.4 Using the Library

This topic describes the basic architecture of the IPv6 TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: ip.h

The interface to the IPv6 TCP/IP Stack library is defined in the "ip.h" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the IPv6 TCP/IP Stack library should include "tcpip.h".

Library File:

The IPv6 TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.15.4.1 Abstraction Model

This library provides the API of the IP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

IP Software Abstraction Block Diagram

This module provides software abstraction of the IP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

Note: This diagram was not available at time of release and will be added in a future version of MPLAB Harmony.

5.8.15.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the IP module.

Library Interface Section	Description
Functions	Routines for Configuring IPv6
Data Types and Constants	This section provides various definitions describing this API

5.8.15.5 Configuring the Library

Macros

	Name	Description
	<u>IPV6_INIT_TASK_PROCESS_RATE</u>	default 32 ms
	<u>IPV6_MINIMUM_LINK_MTU</u>	Sets the lower bounds of the the Maximum Transmission Unit
	<u>IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT</u>	10 minutes
	<u>IPV6_QUEUE_MCAST_PACKET_LIMIT</u>	This option defines the maximum number of multicast queued IPv6. If an additional packet is queued, the oldest packet in the queue will be removed.
	<u>IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT</u>	This option defines the maximum number of queued packets per remote. If an additional packet needs to be queued, the oldest packet in the queue will be removed.
	<u>IPV6_QUEUED_MCAST_PACKET_TIMEOUT</u>	This option defines the number of seconds an IPv6 multicast packet will remain in the queue before being timed out
	<u>IPV6_TASK_PROCESS_RATE</u>	default 1 second
	<u>IPV6_ULA_NTP_ACCESS_TMO</u>	NTP access timeout for the IPv6 ULA address generation, ms
	<u>IPV6_ULA_NTP_VALID_WINDOW</u>	the NTP time stamp validity window, ms if a stamp was obtained outside this interval from the moment of the request a new request will be issued
	<u>IPV6_DEFAULT_BASE_REACHABLE_TIME</u>	30 seconds
	<u>IPV6_DEFAULT_CUR_HOP_LIMIT</u>	(IPv4 Time-to-Live parameter)
	<u>IPV6_DEFAULT_LINK_MTU</u>	Default Maximum Transmission Unit
	<u>IPV6_DEFAULT_RETRANSMIT_TIME</u>	1 second

Description

The configuration of the IPv6 TCP/IP Stack is based on the file `sys_config.h`.

This header file contains the configuration selection for the IPv6 TCP/IP Stack. Based on the selections made, the IPv6 TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the IPv6 TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.15.5.1 IPV6_INIT_TASK_PROCESS_RATE Macro

C

```
#define IPV6_INIT_TASK_PROCESS_RATE (32)           // default 32 ms
```

Description

default 32 ms

5.8.15.5.2 IPV6_MINIMUM_LINK_MTU Macro

C

```
#define IPV6_MINIMUM_LINK_MTU 1280u
```

Description

Sets the lower bounds of the the Maximum Transmission Unit

5.8.15.5.3 IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT Macro

C

```
#define IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT 600ul           // 10 minutes
```

Description

10 minutes

5.8.15.5.4 IPV6_QUEUE_MCAST_PACKET_LIMIT Macro

C

```
#define IPV6_QUEUE_MCAST_PACKET_LIMIT 4
```

Description

This option defines the maximum number of multicast queued IPv6. If an additional packet is queued, the oldest packet in the queue will be removed.

5.8.15.5.5 IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT Macro

C

```
#define IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT 1
```

Description

This option defines the maximum number of queued packets per remote. If an additional packet needs to be queued, the oldest packet in the queue will be removed.

5.8.15.5.6 IPV6_QUEUED_MCAST_PACKET_TIMEOUT Macro

C

```
#define IPV6_QUEUED_MCAST_PACKET_TIMEOUT 10u
```

Description

This option defines the number of seconds an IPv6 multicast packet will remain in the queue before being timed out

5.8.15.5.7 IPV6_TASK_PROCESS_RATE Macro

C

```
#define IPV6_TASK_PROCESS_RATE (1000)           // default 1 second
```

Description

default 1 second

5.8.15.5.8 IPV6_ULA_NTP_ACCESS_TMO Macro

C

```
#define IPV6_ULA_NTP_ACCESS_TMO (12000)
```

Description

NTP access timeout for the IPv6 ULA address generation, ms

5.8.15.5.9 IPV6_ULA_NTP_VALID_WINDOW Macro

C

```
#define IPV6_ULA_NTP_VALID_WINDOW (1000)
```

Description

the NTP time stamp validity window, ms if a stamp was obtained outside this interval from the moment of the request a new request will be issued

5.8.15.5.10 IPV6_DEFAULT_BASE_REACHABLE_TIME Macro

C

```
#define IPV6_DEFAULT_BASE_REACHABLE_TIME 30u
```

Description

30 seconds

5.8.15.5.11 IPV6_DEFAULT_CUR_HOP_LIMIT Macro

C

```
#define IPV6_DEFAULT_CUR_HOP_LIMIT 64u
```

Description

(IPv4 Time-to-Live parameter)

5.8.15.5.12 IPV6_DEFAULT_LINK_MTU Macro

C

```
#define IPV6_DEFAULT_LINK_MTU 1500u
```

Description

Default Maximum Transmission Unit

5.8.15.5.13 IPV6_DEFAULT_RETRANSMIT_TIME Macro

C

```
#define IPV6_DEFAULT_RETRANSMIT_TIME 1000u
```

Description

1 second

5.8.15.6 Building the Library

This section list the files that are available in the \src of the IP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- ip.c

5.8.15.7 Library Interface

Data Types and Constants

	Name	Description
	<u>IP_VERSION_4</u>	Using IPv4
	<u>IP_VERSION_6</u>	Using IPv6
	<u>IPV6_DATA_DYNAMIC_BUFFER</u>	Data to transmit is allocated in dynamically allocated RAM
	<u>IPV6_DATA_NETWORK_FIFO</u>	Data to transmit is stored in the Network Controller's FIFOs
	<u>IPV6_DATA_NONE</u>	The data segment is unused
	<u>IPV6_DATA_PIC_RAM</u>	Data to transmit is stored in PIC RAM
	<u>IPV6_HEADER_OFFSET_DEST_ADDR</u>	Header offset for destination address
	<u>IPV6_HEADER_OFFSET_NEXT_HEADER</u>	Header offset for next header
	<u>IPV6_HEADER_OFFSET_PAYLOAD_LENGTH</u>	Header offset for payload length
	<u>IPV6_HEADER_OFFSET_SOURCE_ADDR</u>	Header offset for source address
	<u>IPV6_NO_UPPER_LAYER_CHECKSUM</u>	Value flag for no upper layer checksum
	<u>IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM</u>	IPv6 Type-length-value type code for the Hop-by-hop "Jumbogram Payload" option
	<u>IPV6_TLV_HBHO_ROUTER_ALERT</u>	IPv6 Type-length-value type code for the Hop-by-hop "Router Alert" option
	<u>IPV6_TLV_PAD_1</u>	IPv6 Type-length-value type code for the Pad 1 option
	<u>IPV6_TLV_PAD_N</u>	IPv6 Type-length-value type code for the Pad N option
	<u>IPV6_TLV_UNREC_OPT_DISCARD_PP</u>	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message
	<u>IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC</u>	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message is the destination addr isn't a multicast address
	<u>IPV6_TLV_UNREC_OPT_DISCARD_SILENT</u>	IPv6 action code for the unrecognized option reaction to discard the packet silently
	<u>IPV6_TLV_UNREC_OPT_SKIP_OPTION</u>	IPv6 action code for the unrecognized option reaction to skip the option
	<u>TCPIP_IPV6_PutArray</u>	Writes data to a packet
	<u>IPV6_ACTION</u>	Provides a list of possible IPv6 actions.
	<u>IPV6_ADDRESS_POLICY</u>	
	<u>IPV6_ADDRESS_PREFERENCE</u>	Provides selection of public versus temporary addresses.
	<u>IPV6_ADDRESS_TYPE</u>	Data structure for IPv6 addresses.
	<u>IPV6_DATA_SEGMENT_HEADER</u>	
	<u>IPV6_EVENT_HANDLER</u>	Clients can register a handler with the IPv6 service.
	<u>IPV6_EVENT_TYPE</u>	None

	IPV6_FRAGMENT_HEADER	
	IPV6_HANDLE	Pointer to IPv6 object
	IPV6_HEADER	None
	IPV6_MODULE_CONFIG	Provides a placeholder for IPv6 configuration.
	IPV6_NEXT_HEADER_TYPE	Defines a list of next header types
	IPV6_PACKET	Packet structure/state tracking for IPv6 packets
	IPV6_PACKET_ACK_FNC	None
	IPV6_RX_FRAGMENT_BUFFER	
	IPV6_SEGMENT_TYPE	Provides an enumeration of IPv6 segment types.
	IPV6_TLV_OPTION_TYPE	Data structure for IPv6 TLV options.
	IPV6_ULA_FLAGS	Provides a list of possible flags for the Unique Local Address (ULA) generation.
	IPV6_ULA_RESULT	Provides a list of possible results for the Unique Local Address (ULA) generation.

Functions

	Name	Description
	TCPIP_IPV6_AddressUnicastRemove	Removed a configured unicast address from an interface.
	TCPIP_IPV6_ArrayGet	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_ArrayPutHelper	Helper function to write data to a packet.
	TCPIP_IPV6_DASSourceAddressSelect	Determines the appropriate source address for a given destination address.
	TCPIP_IPV6_DestAddressGet	Gets the destination address for an IPv6 packet.
	TCPIP_IPV6_DestAddressSet	Sets the destination address for an IPv6 packet.
	TCPIP_IPV6_Flush	Flushes an IP TX packet.
	TCPIP_IPV6_Get	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_HandlerDeregister	Deregisters an IPv6 event handler callback function.
	TCPIP_IPV6_HandlerRegister	Registers an IPv6 event handler callback function.
	TCPIP_IPV6_InterfaceIsReady	Determines if an interface is ready for IPv6 transactions.
	TCPIP_IPV6_MulticastListenerAdd	Adds a multicast listener to an interface.
	TCPIP_IPV6_MulticastListenerRemove	Removes a multicast listener from a given interface.
	TCPIP_IPV6_PacketFree	Frees a TCP/IP Packet structure from dynamic memory.
	TCPIP_IPV6_PayloadSet	Allocates a segment on the end of a packet segment chain and uses it to address pre-buffered data.
	TCPIP_IPV6_Put	Writes a character of data to a packet.
	TCPIP_IPV6_SourceAddressGet	Gets the source address for an IPv6 packet.
	TCPIP_IPV6_SourceAddressSet	Sets the source address for an IPv6 packet.
	TCPIP_IPV6_TxIsPutReady	Determines whether a TX packet can be written to.
	TCPIP_IPV6_TxPacketAllocate	Dynamically allocates a packet for transmitting IP protocol data.
	TCPIP_IPV6_UnicastAddressAdd	Adds a unicast address to a specified interface
	TCPIP_IPV6_UniqueLocalUnicastAddressAdd	Adds a Unique Local Unicast Address (ULA) to a specified interface

Description

This section describes the Application Programming Interface (API) functions of the IPv6 TCP/IP Stack Library.

Refer to each section for a detailed description.

5.8.15.7.1 Functions

5.8.15.7.1.1 TCPIP_IPV6_AddressUnicastRemove Function

C

```
void TCPIP_IPV6_AddressUnicastRemove(  
    TCPIP_NET_HANDLE netH,  
    IPV6_ADDR * address  
);
```

Description

Removed a configured unicast address from an interface.

Preconditions

None

Parameters

Parameters	Description
netH	The interface.
address	The address

Returns

None

Remarks

None

5.8.15.7.1.2 TCPIP_IPV6_ArrayGet Function

C

```
uint8_t TCPIP_IPV6_ArrayGet(  
    TCPIP_MAC_PACKET* pRxPkt,  
    uint8_t * val,  
    uint16_t len  
);
```

Description

Reads a character of data from a packet.

Preconditions

None

Parameters

Parameters	Description
pRxPkt	The MAC RX packet to read data from
val	The buffer to store the data
len	The amount of data to read

Returns

uint8_t - The number of bytes read.

Remarks

None

5.8.15.7.1.3 TCPIP_IPV6_ArrayPutHelper Function**C**

```
unsigned short TCPIP_IPV6_ArrayPutHelper(  
    IPV6_PACKET * pkt,  
    const void * dataSource,  
    uint8_t dataType,  
    unsigned short len  
);
```

Description

Helper function to write data to a packet.

Preconditions

None

Parameters

Parameters	Description
ptrPacket	The packet.
dataSource	The address of the data on its medium.
dataType	Descriptor of the data type (dynamic memory on PIC, in a network FIFO, in static PIC RAM)
len	Length of the data.

Returns

unsigned short - The number of bytes of data written.

Remarks

None

5.8.15.7.1.4 TCPIP_IPV6_DASSourceAddressSelect Function**C**

```
IPV6_ADDR_STRUCT * TCPIP_IPV6_DASSourceAddressSelect(  
    TCPIP_NET_HANDLE hNetIf,  
    IPV6_ADDR * dest,  
    IPV6_ADDR * requestedSource  
);
```

Description

Determines the appropriate source address for a given destination address.

Preconditions

None

Parameters

Parameters	Description
hNetIf	The given interface.
dest	The destination address.
requestedSource	A specified source.

Returns

IPV6_ADDR_STRUCT * - The selected source address.

Remarks

None

5.8.15.7.1.5 TCPIP_IPV6_DestAddressGet Function**C**

```
IPV6_ADDR * TCPIP_IPV6_DestAddressGet(  
    IPV6_PACKET * p  
);
```

Description

Gets the destination address for an IPv6 packet.

Preconditions

None.

Parameters

Parameters	Description
p	pointer to IPv6 packet

Returns

Destination address.

Remarks

None.

5.8.15.7.1.6 TCPIP_IPV6_DestAddressSet Function**C**

```
void TCPIP_IPV6_DestAddressSet(  
    IPV6_PACKET * p,  
    const IPV6_ADDR * addr  
);
```

Description

Sets the destination address for an IPv6 packet.

Preconditions

None.

Parameters

Parameters	Description
p	pointer to IPv6 packet
addrValue	destination address

Returns

None.

Remarks

None

5.8.15.7.1.7 TCPIP_IPV6_Flush Function

C

```
int TCPIP_IPV6_Flush(  
    IPV6_PACKET * pkt  
);
```

Description

Flushes an IP TX packet. Determines the link-layer address if necessary. Calculates the upper-layer checksum if necessary.

Preconditions

None

Parameters

Parameters	Description
ptrPacket	The packet to flush.

Returns

1 - if the packet has been transmitted 0 - if the packet has been queued. < 0 - if the packet has been discarded for some error

Remarks

None

5.8.15.7.1.8 TCPIP_IPV6_Get Function

C

```
uint8_t TCPIP_IPV6_Get(  
    TCPIP_MAC_PACKET* pRxPkt,  
    uint8_t* pData  
);
```

Description

Reads a character of data from a packet.

Preconditions

None

Parameters

Parameters	Description
pRxPkt	The MAC RX packet to read data from

Returns

The data read.

Remarks

None

5.8.15.7.1.9 TCPIP_IPV6_HandlerDeregister Function

C

```
bool TCPIP_IPV6_HandlerDeregister(  
    IPV6_HANDLE hIpv6  
);
```


Description

Deregisters an IPv6 event handler callback function.

Preconditions

None

Parameters

Parameters	Description
hIpv6	Handle to registered callback

Returns

true - if deregister successful false - otherwise.

Remarks

None

5.8.15.7.1.10 TCPIP_IPV6_HandlerRegister Function**C**

```
IPv6_HANDLE TCPIP_IPV6_HandlerRegister(  
    TCPIP_NET_HANDLE hNet,  
    IPV6_EVENT_HANDLER handler,  
    const void* hParam  
);
```

Description

Registers an IPv6 event handler callback function.

Preconditions

None

Parameters

Parameters	Description
netH	The interface
handler	event handler
hParam	

Returns

Handle to registered callback.

Remarks

None

5.8.15.7.1.11 TCPIP_IPV6_InterfaceIsReady Function**C**

```
bool TCPIP_IPV6_InterfaceIsReady(  
    TCPIP_NET_HANDLE netH  
);
```

Description

Determines if an interface is ready for IPv6 transactions.

Preconditions

None

Parameters

Parameters	Description
pNetIf	The interface to check

Returns

true if the interface has IPv6 functionality available, false otherwise

Remarks

None

5.8.15.7.1.12 TCPIP_IPV6_MulticastListenerAdd Function**C**

```
IPv6_ADDR_STRUCT * TCPIP_IPV6_MulticastListenerAdd(  
    TCPIP_NET_HANDLE hNet,  
    IPV6_ADDR * address  
);
```

Description

Adds a multicast listener to an interface.

Preconditions

None

Parameters

Parameters	Description
pNetIf	The interface to add the address to.
address	The new listener

Returns

IPv6_ADDR_STRUCT * - Pointer to the new listener, or NULL

Remarks

None

5.8.15.7.1.13 TCPIP_IPV6_MulticastListenerRemove Function**C**

```
void TCPIP_IPV6_MulticastListenerRemove(  
    TCPIP_NET_HANDLE neth,  
    IPV6_ADDR * address  
);
```

Description

Removes a multicast listener from a given interface.

Preconditions

None

Parameters

Parameters	Description
neth	The interface
address	The address

Returns

None

Remarks

None

5.8.15.7.1.14 TCPIP_IPV6_PacketFree Function**C**

```
void TCPIP_IPV6_PacketFree(  
    IPV6_PACKET * pkt  
)  
;
```

Description

Frees a TCP/IP Packet structure from dynamic memory.

Preconditions

None

Parameters

Parameters	Description
ptrPacket	The packet to free.

Returns

None

Remarks

None

5.8.15.7.1.15 TCPIP_IPV6_PayloadSet Function**C**

```
unsigned short TCPIP_IPV6_PayloadSet(  
    IPV6_PACKET * pkt,  
    uint8_t* payload,  
    unsigned short len  
)  
;
```

Description

This function will allocate a data segment header and append it to the end of a chain of segments in a TX packet. It will set the data ptr in the packet segment to a pre-existing buffer of data.

Preconditions

None

Parameters

Parameters	Description
ptrPacket	The packet.

payload	Address of the data payload.
len	Length of the data payload

Returns

unsigned short - The amount of data added to the packet length.

Remarks

This function is useful for adding payloads to outgoing packets without copying them if the data is in another preexisting buffer (i.e. TCP).

5.8.15.7.1.16 TCPIP_IPV6_Put Function**C**

```
bool TCPIP_IPV6_Put(  
    IPV6_PACKET * pkt,  
    unsigned char v  
);
```

Description

Writes a character of data to a packet.

Preconditions

None

Parameters

Parameters	Description
pkt	The packet.
v	The character.

Returns

true if the character was written, false otherwise

Remarks

None

5.8.15.7.1.17 TCPIP_IPV6_SourceAddressGet Function**C**

```
IPV6_ADDR * TCPIP_IPV6_SourceAddressGet(  
    IPV6_PACKET * p  
);
```

Description

Gets the source address for an IPv6 packet.

Parameters

Parameters	Description
p	pointer to IPv6 packet

Returns

Source address.

Remarks

None.

5.8.15.7.1.18 TCPIP_IPV6_SourceAddressSet Function**C**

```
void TCPIP_IPV6_SourceAddressSet(  
    IPV6_PACKET * p,  
    const IPV6_ADDR * addr  
);
```

Description

Sets the source address for an IPv6 packet.

Preconditions

None.

Parameters

Parameters	Description
p	pointer to IPv6 packet
addrValue	source address

Returns

None.

Remarks

None

5.8.15.7.1.19 TCPIP_IPV6_TxIsPutReady Function**C**

```
unsigned short TCPIP_IPV6_TxIsPutReady(  
    IPV6_PACKET * pkt,  
    unsigned short count  
);
```

Description

Determines whether a TX packet can be written to. This function will allocate additional space to the packet to accomodate the user.

Preconditions

None

Parameters

Parameters	Description
ptrPacket	The packet to check.
count	The amount of writable space to check for,

Returns

unsigned short - The amount of space available.

Remarks

None

5.8.15.7.1.20 TCPIP_IPV6_TxPacketAllocate Function

C

```
IPV6_PACKET * TCPIP_IPV6_TxPacketAllocate(  
    TCPIP_NET_HANDLE netH,  
    IPV6_PACKET_ACK_FNC ackFnc,  
    void* ackParam  
);
```

Description

Dynamically allocates a packet for transmitting IP protocol data.

Preconditions

None

Parameters

Parameters	Description
netH	Interface of the outgoing packet.
ackFnc	function to be called when IP is done with the TX packet (finished transmitting)
ackParam	parameter to be used for this callback This has meaning only for the caller of the TCPIP_IPV6_TxPacketAllocate

Returns

[IPV6_PACKET](#) * - Pointer to the allocated packet.

Remarks

None

5.8.15.7.1.21 TCPIP_IPV6_UnicastAddressAdd Function

C

```
IPV6_ADDR_STRUCT * TCPIP_IPV6_UnicastAddressAdd(  
    TCPIP_NET_HANDLE netH,  
    IPV6_ADDR * address,  
    uint8_t skipProcessing  
);
```

Description

Adds a unicast address to a specified interface. Starts duplicate address detection if necessary.

Preconditions

None

Parameters

Parameters	Description
netH	The interface to add the address to.
address	The address to add.
skipProcessing	true to skip Duplicate address detection, false otherwise

Returns

IPV6_ADDR_STRUCT * - Pointer to the structure of the newly allocated address

Remarks

None

5.8.15.7.1.22 TCPIP_IPV6_UniqueLocalUnicastAddressAdd Function

C

```
IPV6_ULA_RESULT TCPIP_IPV6_UniqueLocalUnicastAddressAdd(
    TCPIP_NET_HANDLE netH,
    uint16_t subnetID,
    IPV6_ULA_FLAGS genFlags,
    IP_MULTI_ADDRESS* ntpAddress
);
```

Description

This function starts the process of adding an ULA address to the specified interface. The segments of the generated address are as follows: FC00::7 - ULA prefix L - 1 bit set to 1, locally assigned Global ID - 40 bit random generated identifier subnet ID - 16 bit subnet identifier Interface ID - 64 bit interface identifier generated as a EUI64 from the specified interface MAC.

The randomness of the "Global ID" prefix of the generated IPv6 address is obtained by using an NTP server. The supplied NTP server will be contacted to obtain an NTP timestamp. This timestamp together with the EUI64 identifier obtained from the interface MAC are passed through a 160 bits hash algorithm (SHA1) and the least seighbificant 40 bits are used as the GlobalID of the interface.

Preconditions

None

Parameters

Parameters	Description
netH	the interface to add the address to.
subnetID	the subnet ID to be used.

5.8.15.7.2 Data Types and Constants

5.8.15.7.2.1 IP_VERSION_4 Macro

C

```
#define IP_VERSION_4 (0u) // Using IPv4
```

Description

Using IPv4

5.8.15.7.2.2 IP_VERSION_6 Macro

C

```
#define IP_VERSION_6 (1u) // Using IPv6
```

Description

Using IPv6

5.8.15.7.2.3 IPV6_DATA_DYNAMIC_BUFFER Macro

C

```
#define IPV6_DATA_DYNAMIC_BUFFER (0x1u) // Data to transmit is allocated in dynamically allocated RAM
```

Description

Data to transmit is allocated in dynamically allocated RAM

5.8.15.7.2.4 IPV6_DATA_NETWORK_FIFO Macro**C**

```
#define IPV6_DATA_NETWORK_FIFO (0x2u)           // Data to transmit is stored in the Network  
Controller's FIFOs
```

Description

Data to transmit is stored in the Network Controller's FIFOs

5.8.15.7.2.5 IPV6_DATA_NONE Macro**C**

```
#define IPV6_DATA_NONE (0x0u)                   // The data segment is unused
```

Description

The data segment is unused

5.8.15.7.2.6 IPV6_DATA_PIC_RAM Macro**C**

```
#define IPV6_DATA_PIC_RAM (0x3u)                // Data to transmit is stored in PIC RAM
```

Description

Data to transmit is stored in PIC RAM

5.8.15.7.2.7 IPV6_HEADER_OFFSET_DEST_ADDR Macro**C**

```
#define IPV6_HEADER_OFFSET_DEST_ADDR (0x08u + sizeof (IPV6_ADDR)) // Header offset for  
destination address
```

Description

Header offset for destination address

5.8.15.7.2.8 IPV6_HEADER_OFFSET_NEXT_HEADER Macro**C**

```
#define IPV6_HEADER_OFFSET_NEXT_HEADER (0x06u) // Header offset for next header
```

Description

Header offset for next header

5.8.15.7.2.9 IPV6_HEADER_OFFSET_PAYLOAD_LENGTH Macro**C**

```
#define IPV6_HEADER_OFFSET_PAYLOAD_LENGTH (0x04u) // Header offset for payload length
```

Description

Header offset for payload length

5.8.15.7.2.10 IPV6_HEADER_OFFSET_SOURCE_ADDR Macro

C

```
#define IPV6_HEADER_OFFSET_SOURCE_ADDR (0x08u) // Header offset for source address
```

Description

Header offset for source address

5.8.15.7.2.11 IPV6_NO_UPPER_LAYER_CHECKSUM Macro

C

```
#define IPV6_NO_UPPER_LAYER_CHECKSUM (0xFFFFu) // Value flag for no upper layer checksum
```

Description

Value flag for no upper layer checksum

5.8.15.7.2.12 IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM Macro

C

```
#define IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM 0xC2u
```

Description

IPv6 Type-length-value type code for the Hop-by-hop "Jumbogram Payload" option

5.8.15.7.2.13 IPV6_TLV_HBHO_ROUTER_ALERT Macro

C

```
#define IPV6_TLV_HBHO_ROUTER_ALERT 0x05u
```

Description

IPv6 Type-length-value type code for the Hop-by-hop "Router Alert" option

5.8.15.7.2.14 IPV6_TLV_PAD_1 Macro

C

```
#define IPV6_TLV_PAD_1 0u
```

Description

IPv6 Type-length-value type code for the Pad 1 option

5.8.15.7.2.15 IPV6_TLV_PAD_N Macro

C

```
#define IPV6_TLV_PAD_N 1u
```

Description

IPv6 Type-length-value type code for the Pad N option

5.8.15.7.2.16 IPV6_TLV_UNREC_OPT_DISCARD_PP Macro

C

```
#define IPV6_TLV_UNREC_OPT_DISCARD_PP 0b10
```

Description

IPv6 action code for the unrecognized option reaction to discard the packet and [send](#) an ICMP parameter problem message

5.8.15.7.2.17 IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC Macro

C

```
#define IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC 0b11
```

Description

IPv6 action code for the unrecognized option reaction to discard the packet and [send](#) an ICMP parameter problem message is the destination addr isn't a multicast address

5.8.15.7.2.18 IPV6_TLV_UNREC_OPT_DISCARD_SILENT Macro

C

```
#define IPV6_TLV_UNREC_OPT_DISCARD_SILENT 0b01
```

Description

IPv6 action code for the unrecognized option reaction to discard the packet silently

5.8.15.7.2.19 IPV6_TLV_UNREC_OPT_SKIP_OPTION Macro

C

```
#define IPV6_TLV_UNREC_OPT_SKIP_OPTION 0b00
```

Description

IPv6 action code for the unrecognized option reaction to skip the option

5.8.15.7.2.20 TCPIP_IPV6_PutArray Macro

C

```
#define TCPIP_IPV6_PutArray(pkt,data,len) TCPIP_IPV6_ArrayPutHelper(pkt, data, IPV6_DATA_PIC_RAM, len)
```

Description

Writes data to an outgoing packet.

Preconditions

The [TCPIP_IPV6_TxIsPutReady](#) function must have returned a value greater than or equal to 'len.'

Parameters

Parameters	Description
ptrPacket	The packet.
dataSource	Pointer to the data to copy to the packet.
len	Length of the data.

Returns

unsigned short - The number of bytes of data written.

Remarks

None

5.8.15.7.2.21 IPV6_ACTION Enumeration

C

```
typedef enum {  
    IPV6_ACTION_NONE = 0,  
    IPV6_ACTION_DISCARD_SILENT,  
    IPV6_ACTION_DISCARD_PP_0,  
    IPV6_ACTION_DISCARD_PP_2,  
    IPV6_ACTION_DISCARD_PP_2_NOT_MC,  
    IPV6_ACTION_BEGIN_EX_HEADER_PROCESSING  
} IPV6_ACTION;
```

Description

IPV6_ACTION Enumeration

Provides a list of possible IPv6 actions.

Remarks

None

5.8.15.7.2.22 IPV6_ADDRESS_POLICY Structure

C

```
typedef struct {  
    IPV6_ADDR address;  
    unsigned char prefixLength;  
    unsigned char precedence;  
    unsigned char label;  
} IPV6_ADDRESS_POLICY;
```

Description

IPV6_ADDRESS_POLICY Structure Typedef

Remarks

None

5.8.15.7.2.23 IPV6_ADDRESS_PREFERENCE Enumeration

C

```
typedef enum {  
    IPV6_PREFER_PUBLIC_ADDRESSES = 0,  
    IPV6_PREFER_TEMPORARY_ADDRESSES  
} IPV6_ADDRESS_PREFERENCE;
```

Description

IPV6_ADDRESS_PREFERENCE Enumeration

Provides selection of public versus temporary addresses.

Remarks

None

5.8.15.7.2.24 IPV6_ADDRESS_TYPE Union

C

```
typedef union {
```

```

    unsigned char byte;
    struct {
        unsigned scope : 4;
        unsigned type : 2;
    } bits;
} IPV6_ADDRESS_TYPE;

```

Description

IPV6_ADDRESS_TYPE Union Typedef

Data structure for IPv6 addresses.

Remarks

None

5.8.15.7.2.25 IPV6_DATA_SEGMENT_HEADER Type**C**

```

typedef struct _IPV6_DATA_SEGMENT_HEADER IPV6_DATA_SEGMENT_HEADER;

```

Description

IPV6_DATA_SEGMENT_HEADER Structure Typedef

Remarks

None

5.8.15.7.2.26 IPV6_EVENT_HANDLER Type**C**

```

typedef void (* IPV6_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, IPV6_EVENT_TYPE evType, const void*
evParam, const void* usrParam);

```

Description

Prototype of an IPv6 event handler

Once an IPv6 event occurs the IPv6 service will call the registered handler. The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

evParam is a parameter that's associated to an IPv6 event

- For an address event (IPV6_EVENT_ADDRESS_ADDED, IPV6_EVENT_ADDRESS_REMOVED) it should be typecasted to (const IPV6_ADDR_STRUCT*)
- For an IPV6_EVENT_ULA_ADDRESS_GENERATED ULA event it should be typecasted to (const IPV6_ADDR*)
- For an IPV6_EVENT_ULA_ADDRESS_FAILED ULA event the evParam is an [IPV6_ULA_RESULT](#) error code

The evParam is invalid outside of the IPV6_EVENT_HANDLER context call and should not be stored by the caller. Info that's needed has to be copied into caller's own context.

usrParam is a user supplied parameter

Remarks

For address related events the passed (const IPV6_ADDR_STRUCT*) parameter is invalid after the notification call.

5.8.15.7.2.27 IPV6_EVENT_TYPE Enumeration**C**

```

typedef enum {

```

```

    IPV6_EVENT_ADDRESS_ADDED = 1,
    IPV6_EVENT_ADDRESS_REMOVED,
    IPV6_EVENT_ULA_ADDRESS_GENERATED,
    IPV6_EVENT_ULA_ADDRESS_FAILED
} IPV6_EVENT_TYPE;

```

Description

IPV6_EVENT_TYPE Enumeration

None

Remarks

None

5.8.15.7.2.28 IPV6_FRAGMENT_HEADER Structure**C**

```

typedef struct {
    uint8_t nextHeader;
    uint8_t reserved;
    union {
        struct {
            unsigned m : 1;
            unsigned reserved2 : 2;
            unsigned fragmentOffset : 13;
        } bits;
        uint16_t w;
    } offsetM;
    uint32_t identification;
} IPV6_FRAGMENT_HEADER;

```

Description

IPV6_FRAGMENT_HEADER Structure Typedef

Remarks

None

5.8.15.7.2.29 IPV6_HANDLE Type**C**

```

typedef const void * IPV6_HANDLE;

```

Description

Pointer to IPv6 object

5.8.15.7.2.30 IPV6_HEADER Structure**C**

```

typedef struct {
    unsigned long V_T_F;
    unsigned short PayloadLength;
    unsigned char NextHeader;
    unsigned char HopLimit;
    IPV6_ADDR SourceAddress;
    IPV6_ADDR DestAddress;
} IPV6_HEADER;

```

Description

IPv6 packet header definition

None

Remarks

None

5.8.15.7.2.31 IPV6_MODULE_CONFIG Structure

C

```
typedef struct {  
} IPV6_MODULE_CONFIG;
```

Description

IPV6_MODULE_CONFIG Structure Typedef

Provides a placeholder for IPv6 configuration.

Remarks

None

5.8.15.7.2.32 IPV6_NEXT_HEADER_TYPE Enumeration

C

```
typedef enum {  
    IPV6_PROT_HOP_BY_HOP_OPTIONS_HEADER = (0u),  
    IPV6_PROT_ICMP = (1u),  
    IPV6_PROT_TCP = (6u),  
    IPV6_PROT_UDP = (17u),  
    IPV6_PROT_IPV6 = (41u),  
    IPV6_PROT_ROUTING_HEADER = (43u),  
    IPV6_PROT_FRAGMENTATION_HEADER = (44u),  
    IPV6_PROT_ENCAPSULATING_SECURITY_PAYLOAD_HEADER = (50u),  
    IPV6_PROT_AUTHENTICATION_HEADER = (51u),  
    IPV6_PROT_ICMPV6 = (58u),  
    IPV6_PROT_NONE = (59u),  
    IPV6_PROT_DESTINATION_OPTIONS_HEADER = (60u)  
} IPV6_NEXT_HEADER_TYPE;
```

Description

IPV6_NEXT_HEADER_TYPE Enumeration

Defines a list of next header types

Members

Members	Description
IPV6_PROT_HOP_BY_HOP_OPTIONS_HEADER = (0u)	IPv6 Hop-by-Hop Opt. Header
IPV6_PROT_IPV6 = (41u)	IPv6 Protocol
IPV6_PROT_ROUTING_HEADER = (43u)	IPv6 Routing Header
IPV6_PROT_FRAGMENTATION_HEADER = (44u)	IPv6 Fragmentation Header
IPV6_PROT_ENCAPSULATING_SECURITY_PAYLOAD_HEADER = (50u)	Encapsulating Security Payload Header
IPV6_PROT_AUTHENTICATION_HEADER = (51u)	Authentication Header

IPV6_PROT_ICMPV6 = (58u)	ICMPv6 Protocol
IPV6_PROT_NONE = (59u)	No next header
IPV6_PROT_DESTINATION_OPTIONS_HEADER = (60u)	Destination Options Header

Remarks

None

5.8.15.7.2.33 IPV6_PACKET Type**C**

```
typedef struct _IPV6_PACKET IPV6_PACKET;
```

Description

Packet structure/state tracking for IPv6 packets

5.8.15.7.2.34 IPV6_PACKET_ACK_FNC Type**C**

```
typedef void (* IPV6_PACKET_ACK_FNC)(void*, bool, const void*);
```

Description

Packet allocation callback function

1st parameter will be an [IPV6_PACKET](#)* 2nd parameter is supplied by the caller**Remarks**

None

5.8.15.7.2.35 IPV6_RX_FRAGMENT_BUFFER Type**C**

```
typedef struct _IPV6_RX_FRAGMENT_BUFFER IPV6_RX_FRAGMENT_BUFFER;
```

Description

IPV6_RX_FRAGMENT_BUFFER Structure Typedef

Remarks

None

5.8.15.7.2.36 IPV6_SEGMENT_TYPE Enumeration**C**

```
typedef enum {
    TYPE_IPV6_HEADER = 1u,
    TYPE_IPV6_EX_HEADER_HOP_BY_HOP_OPTIONS,
    TYPE_IPV6_EX_HEADER_DESTINATION_OPTIONS_1,
    TYPE_IPV6_EX_HEADER_ROUTING,
    TYPE_IPV6_EX_HEADER_FRAGMENT,
    TYPE_IPV6_EX_HEADER_AUTHENTICATION_HEADER,
    TYPE_IPV6_EX_HEADER_ENCAPSULATING_SECURITY_PAYLOAD,
    TYPE_IPV6_EX_HEADER_DESTINATION_OPTIONS_2,
    TYPE_IPV6_UPPER_LAYER_HEADER,
    TYPE_IPV6_UPPER_LAYER_PAYLOAD,
    TYPE_IPV6_BEGINNING_OF_WRITABLE_PART,
    TYPE_IPV6_END_OF_LIST
}
```

```
} IPV6_SEGMENT_TYPE;
```

Description

IPV6_SEGMENT_TYPE Enumeration

Provides an enumeration of IPv6 segment types.

Remarks

None

5.8.15.7.2.37 IPV6_TLV_OPTION_TYPE Union

C

```
typedef union {
    unsigned char b;
    struct {
        unsigned option : 6;
        unsigned unrecognizedAction : 2;
    } bits;
} IPV6_TLV_OPTION_TYPE;
```

Description

IPV6_TLV_OPTION_TYPE Union Typedef

Data structure for IPv6 TLV options.

Remarks

None

5.8.15.7.2.38 IPV6_ULA_FLAGS Enumeration

C

```
typedef enum {
    IPV6_ULA_FLAG_NTPV4 = 0x01,
    IPV6_ULA_FLAG_GENERATE_ONLY = 0x02,
    IPV6_ULA_FLAG_SKIP_DAD = 0x04
} IPV6_ULA_FLAGS;
```

Description

IPV6_ULA_FLAGS Enumeration

Provides a list of possible ULA action flags.

Members

Members	Description
IPV6_ULA_FLAG_NTPV4 = 0x01	use an IPv4 NTP server access in Unique Local Address generation default is an IPv6 server
IPV6_ULA_FLAG_GENERATE_ONLY = 0x02	generate an address only, don't add it to the interface addresses
IPV6_ULA_FLAG_SKIP_DAD = 0x04	when adding the address to the interface, skip the Duplicate Address Detection

Remarks

None

5.8.15.7.2.39 IPV6_ULA_RESULT Enumeration

C

```
typedef enum {
    IPV6_ULA_RES_OK,
    IPV6_ULA_RES_BUSY = -1,
    IPV6_ULA_RES_IF_ERR = -2,
    IPV6_ULA_RES_NTP_ACCESS_ERR = -3,
    IPV6_ULA_RES_NTP_TSTAMP_ERR = -4
} IPV6_ULA_RESULT;
```

Description

IPV6_ULA_RESULT Enumeration

Provides a list of possible ULA results.

Members

Members	Description
IPV6_ULA_RES_OK	the address generation was started successfully
IPV6_ULA_RES_BUSY = -1	address generation module is busy
IPV6_ULA_RES_IF_ERR = -2	interface non existent
IPV6_ULA_RES_NTP_ACCESS_ERR = -3	NTP module could not be accessed
IPV6_ULA_RES_NTP_TSTAMP_ERR = -4	wrong NTP time stamp received

Remarks

None

5.8.15.8 Files

Files

Name	Description
ipv6.h	

Description

5.8.15.8.1 ipv6.h

IPv6 Defs for Microchip TCP/IP Stack

Enumerations

	Name	Description
	IPV6_ACTION	Provides a list of possible IPv6 actions.
	IPV6_ADDRESS_PREFERENCE	Provides selection of public versus temporary addresses.
	IPV6_EVENT_TYPE	None
	IPV6_NEXT_HEADER_TYPE	Defines a list of next header types
	IPV6_SEGMENT_TYPE	Provides an enumeration of IPv6 segment types.
	IPV6_ULA_FLAGS	Provides a list of possible flags for the Unique Local Address (ULA) generation.
	IPV6_ULA_RESULT	Provides a list of possible results for the Unique Local Address (ULA) generation.

Functions

	Name	Description
	TCPIP_IPV6_AddressUnicastRemove	Removed a configured unicast address from an interface.
	TCPIP_IPV6_ArrayGet	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_ArrayPutHelper	Helper function to write data to a packet.
	TCPIP_IPV6_DASSourceAddressSelect	Determines the appropriate source address for a given destination address.
	TCPIP_IPV6_DestAddressGet	Gets the destination address for an IPv6 packet.
	TCPIP_IPV6_DestAddressSet	Sets the destination address for an IPv6 packet.
	TCPIP_IPV6_Flush	Flushes an IP TX packet.
	TCPIP_IPV6_Get	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_HandlerDeregister	Deregisters an IPv6 event handler callback function.
	TCPIP_IPV6_HandlerRegister	Registers an IPv6 event handler callback function.
	TCPIP_IPV6_InterfacesReady	Determines if an interface is ready for IPv6 transactions.
	TCPIP_IPV6_MulticastListenerAdd	Adds a multicast listener to an interface.
	TCPIP_IPV6_MulticastListenerRemove	Removes a multicast listener from a given interface.
	TCPIP_IPV6_PacketFree	Frees a TCP/IP Packet structure from dynamic memory.
	TCPIP_IPV6_PayloadSet	Allocates a segment on the end of a packet segment chain and uses it to address pre-buffered data.
	TCPIP_IPV6_Put	Writes a character of data to a packet.
	TCPIP_IPV6_SourceAddressGet	Gets the source address for an IPv6 packet.
	TCPIP_IPV6_SourceAddressSet	Sets the source address for an IPv6 packet.
	TCPIP_IPV6_TxIsPutReady	Determines whether a TX packet can be written to.
	TCPIP_IPV6_TxPacketAllocate	Dynamically allocates a packet for transmitting IP protocol data.
	TCPIP_IPV6_UnicastAddressAdd	Adds a unicast address to a specified interface
	TCPIP_IPV6_UniqueLocalUnicastAddressAdd	Adds a Unique Local Unicast Address (ULA) to a specified interface

Macros

	Name	Description
	IP_VERSION_4	Using IPv4
	IP_VERSION_6	Using IPv6
	IPV6_DATA_DYNAMIC_BUFFER	Data to transmit is allocated in dynamically allocated RAM
	IPV6_DATA_NETWORK_FIFO	Data to transmit is stored in the Network Controller's FIFOs
	IPV6_DATA_NONE	The data segment is unused
	IPV6_DATA_PIC_RAM	Data to transmit is stored in PIC RAM
	IPV6_HEADER_OFFSET_DEST_ADDR	Header offset for destination address
	IPV6_HEADER_OFFSET_NEXT_HEADER	Header offset for next header
	IPV6_HEADER_OFFSET_PAYLOAD_LENGTH	Header offset for payload length
	IPV6_HEADER_OFFSET_SOURCE_ADDR	Header offset for source address
	IPV6_NO_UPPER_LAYER_CHECKSUM	Value flag for no upper layer checksum
	IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM	IPv6 Type-length-value type code for the Hop-by-hop "Jumbogram Payload" option
	IPV6_TLV_HBHO_ROUTER_ALERT	IPv6 Type-length-value type code for the Hop-by-hop "Router Alert" option
	IPV6_TLV_PAD_1	IPv6 Type-length-value type code for the Pad 1 option
	IPV6_TLV_PAD_N	IPv6 Type-length-value type code for the Pad N option

	IPV6_TLV_UNREC_OPT_DISCARD_PP	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message
	IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message is the destination addr isn't a multicast address
	IPV6_TLV_UNREC_OPT_DISCARD_SILENT	IPv6 action code for the unrecognized option reaction to discard the packet silently
	IPV6_TLV_UNREC_OPT_SKIP_OPTION	IPv6 action code for the unrecognized option reaction to skip the option
	TCPIP_IPV6_PutArray	Writes data to a packet

Structures

	Name	Description
	IPV6_ADDRESS_POLICY	
	IPV6_FRAGMENT_HEADER	
	IPV6_HEADER	None
	IPV6_MODULE_CONFIG	Provides a placeholder for IPv6 configuration.

Types

	Name	Description
	IPV6_DATA_SEGMENT_HEADER	
	IPV6_EVENT_HANDLER	Clients can register a handler with the IPv6 service.
	IPV6_HANDLE	Pointer to IPv6 object
	IPV6_PACKET	Packet structure/state tracking for IPv6 packets
	IPV6_PACKET_ACK_FNC	None
	IPV6_RX_FRAGMENT_BUFFER	

Unions

	Name	Description
	IPV6_ADDRESS_TYPE	Data structure for IPv6 addresses.
	IPV6_TLV_OPTION_TYPE	Data structure for IPv6 TLV options.

5.8.15.8.2 ip_config.h

Refer to the IPv4 Help for this file: [ip_config.h](#)

5.8.16 MAC TCP/IP Stack Library

5.8.16.1 Introduction

Media Access Control (MAC) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the MAC TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

In the seven-layer OSI model of computer networking, media access control (MAC) data communication protocol is a sublayer of the data link layer, which itself is layer 2. The MAC sublayer provides addressing and channel access control mechanisms that make it possible for several terminals or network nodes to communicate within a multiple access network that incorporates a shared medium, e.g. Ethernet. The hardware that implements the MAC is referred to as a medium access controller.

The MAC sublayer acts as an interface between the logical link control (LLC) sublayer and the network's physical layer. The MAC layer emulates a full-duplex logical communication channel in a multi-point network. This channel may provide unicast, multicast or broadcast communication service.

5.8.16.2 Release Notes

MPLAB Harmony Version: v0.70b **MAC TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.16.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.16.4 Using the Library

This topic describes the basic architecture of the MAC TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [tcpip_mac.h](#)

The interface to the MAC TCP/IP Stack library is defined in the "[tcpip_mac.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the MAC TCP/IP Stack library should include "tcpip.h".

Library File:

The MAC TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.16.4.1 Abstraction Model

This library provides the API of the MAC TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

MAC Software Abstraction Block Diagram

Note: This diagram was not available at time of release and will be added in a future version of MPLAB Harmony.

5.8.16.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the MAC module.

Library Interface Section	Description
Functions	Routines for configuring the MAC TCP/IP Stack
Data Types and Constants	This section provides various definitions describing this API

5.8.16.5 Configuring the Library

Macros

	Name	Description
	EMAC_RX_BUFF_SIZE	size of a RX buffer. should be multiple of 16 this is the size of all receive buffers processed by the ETHC The size should be enough to accomodate any network received packet If the packets are larger, they will have to take multiple RX buffers and the packet manipulation is less efficient <code>#define EMAC_RX_BUFF_SIZE 512</code>
	EMAC_RX_DESCRIPTOR	number of the RX descriptors and RX buffers to be created
	EMAC_RX_FRAGMENTS	MAC maximum number of supported fragments Based on the values of EMAC_RX_MAX_FRAME and EMAC_RX_BUFF_SIZE an incoming frame may span multiple RX buffers (fragments). Note that excessive fragmentation leads to performance degradation. The default and recommended value should be 1. <code>#define EMAC_RX_FRAGMENTS 1</code> Alternatively you can use the calculation of the number of fragments based on the selected RX sizes:
	EMAC_RX_MAX_FRAME	Maximum MAC supported RX frame size. Any incoming ETH frame that's longer than this size will be discarded. The default value is 1536 (allows for VLAN tagged frames, although the VLAN tagged frames are discarded) Normally you shouldn't need to touch this value unless you know exactly the maximum size of the frames you want to process or you need to control packets fragmentation (together with the EMAC_RX_BUFF_SIZE).
	EMAC_TX_DESCRIPTOR	number of the TX descriptors to be created
	ETH_CFG_10	use/advertise 10 Mbps capability

	ETH_CFG_100	use/advertise 100 Mbps capability
	ETH_CFG_AUTO	use auto negotiation
	ETH_CFG_AUTO_MDIX	use/advertise auto MDIX capability
	ETH_CFG_FDUPLEX	use/advertise full duplex capability
	ETH_CFG_HDUPLEX	use/advertise half duplex capability
	ETH_CFG_LINK	set to 1 if you need to config the link to specific following parameters otherwise the default connection will be attempted depending on the selected PHY
	ETH_CFG_LINK_INIT_DELAY	some PHYs need an initialization delay for insuring that the PHY is ready to transmit data. milliseconds
	ETH_CFG_SWAP_MDIX	use swapped MDIX. else normal MDIX

Description

The configuration of the MAC TCP/IP Stack is based on the file `tcpip_config.h`

This header file contains the configuration selection for the MAC TCP/IP Stack. Based on the selections made, the MAC TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the MAC TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.16.5.1 EMAC_RX_BUFF_SIZE Macro

C

```
#define EMAC_RX_BUFF_SIZE 1536
```

Description

size of a RX buffer. should be multiple of 16 this is the size of all receive buffers processed by the ETHC The size should be enough to accomodate any network received packet If the packets are larger, they will have to take multiple RX buffers and the packet manipulation is less efficient #define EMAC_RX_BUFF_SIZE 512

5.8.16.5.2 EMAC_RX_DESCRIPTORs Macro

C

```
#define EMAC_RX_DESCRIPTORs 6
```

Description

number of the RX descriptors and RX buffers to be created

5.8.16.5.3 EMAC_RX_FRAGMENTS Macro

C

```
#define EMAC_RX_FRAGMENTS ((EMAC_RX_MAX_FRAME + (EMAC_RX_BUFF_SIZE - 1)) / (EMAC_RX_BUFF_SIZE))
```

Description

MAC maximum number of supported fragments Based on the values of [EMAC_RX_MAX_FRAME](#) and [EMAC_RX_BUFF_SIZE](#) an incoming frame may span multiple RX buffers (fragments). Note that excessive fragmentation leads to performance degradation. The default and recommended value should be 1. #define EMAC_RX_FRAGMENTS 1 Alternatively you can use the calculation of the number of fragments based on the selected RX sizes:

5.8.16.5.4 EMAC_RX_MAX_FRAME Macro

C

```
#define EMAC_RX_MAX_FRAME 1536
```

Description

Maximum MAC supported RX frame size. Any incoming ETH frame that's longer than this size will be discarded. The default value is 1536 (allows for VLAN tagged frames, although the VLAN tagged frames are discarded) Normally you shouldn't need to touch this value unless you know exactly the maximum size of the frames you want to process or you need to control packets fragmentation (together with the [EMAC_RX_BUFF_SIZE](#)).

Remarks

Always multiple of 16!

5.8.16.5.5 EMAC_TX_DESCRIPTOR Macro

C

```
#define EMAC_TX_DESCRIPTOR 8
```

Description

number of the TX descriptors to be created

5.8.16.5.6 ETH_CFG_10 Macro

C

```
#define ETH_CFG_10 1
```

Description

use/advertise 10 Mbps capability

5.8.16.5.7 ETH_CFG_100 Macro

C

```
#define ETH_CFG_100 1
```

Description

use/advertise 100 Mbps capability

5.8.16.5.8 ETH_CFG_AUTO Macro

C

```
#define ETH_CFG_AUTO 1
```

Description

use auto negotiation

5.8.16.5.9 ETH_CFG_AUTO_MDIX Macro

C

```
#define ETH_CFG_AUTO_MDIX 1
```

Description

use/advertise auto MDIX capability

5.8.16.5.10 ETH_CFG_FDUPLEX Macro

C

```
#define ETH_CFG_FDUPLEX 1
```

Description

use/advertise full duplex capability

5.8.16.5.11 ETH_CFG_HDUPLEX Macro

C

```
#define ETH_CFG_HDUPLEX 1
```

Description

use/advertise half duplex capability

5.8.16.5.12 ETH_CFG_LINK Macro

C

```
#define ETH_CFG_LINK 0
```

Description

set to 1 if you need to config the link to specific following parameters otherwise the default connection will be attempted depending on the selected PHY

5.8.16.5.13 ETH_CFG_LINK_INIT_DELAY Macro

C

```
#define ETH_CFG_LINK_INIT_DELAY 500
```

Description

some PHYs need an initialization delay for insuring that the PHY is ready to transmit data. milliseconds

5.8.16.5.14 ETH_CFG_SWAP_MDIX Macro

C

```
#define ETH_CFG_SWAP_MDIX 1
```

Description

use swapped MDIX. else normal MDIX

5.8.16.6 Building the Library

This section list the files that are available in the \src of the MAC driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.16.7 Library Interface

Data Types and Constants

	Name	Description

5.8.16.7.1 Functions

5.8.16.7.1.1 TCPIP_MAC_Close Function

C

```
TCPIP_MAC_RES TCPIP_MAC_Close(
    TCPIP_MAC_HANDLE hMac
);
```

Description

closes a MAC client

5.8.16.7.1.2 TCPIP_MAC_Deinitialize Function

C

```
TCPIP_MAC_RES TCPIP_MAC_Deinitialize(
    const TCPIP_MAC_MODULE_CTRL * const
);
```

Description

MAC de-initialization function Returns a result to indicate that everything has been cleaned up.

5.8.16.7.1.3 TCPIP_MAC_EventAcknowledge Function

C

```
bool TCPIP_MAC_EventAcknowledge(
    TCPIP_MAC_HANDLE hMac,
    TCPIP_MAC_EVENT macEvents
);
```

Description

acknowledges reported event(s) returns false if no event to be acknowledged true otherwise

5.8.16.7.1.4 TCPIP_MAC_EventMaskSet Function

C

```
bool TCPIP_MAC_EventMaskSet(
    TCPIP_MAC_HANDLE hMac,
    TCPIP_MAC_EVENT macEvents,
    bool enable
);
```

```
);
```

Description

enables/disables the events to be reported multiple events could be or-ed returns false if some error occurred

5.8.16.7.1.5 TCPIP_MAC_EventPendingGet Function**C**

```
TCPIP_MAC_EVENT TCPIP_MAC_EventPendingGet(  
    TCPIP_MAC_HANDLE hMac  
);
```

Description

returns current pending events

5.8.16.7.1.6 TCPIP_MAC_Initialize Function**C**

```
TCPIP_MAC_RES TCPIP_MAC_Initialize(  
    TCPIP_MAC_MODULE_CTRL* const stackData,  
    const void* const moduleData  
);
```

Description

TCPIP_MAC_Initialize

5.8.16.7.1.7 TCPIP_MAC_LinkCheck Function**C**

```
bool TCPIP_MAC_LinkCheck(  
    TCPIP_MAC_HANDLE hMac  
);
```

Description

Link Interface functions

5.8.16.7.1.8 TCPIP_MAC_Open Function**C**

```
TCPIP_MAC_HANDLE TCPIP_MAC_Open(  
    unsigned int macId  
);
```

Description

function to open a MAC and get a client handle

5.8.16.7.1.9 TCPIP_MAC_PacketRx Function**C**

```
TCPIP_MAC_PACKET* TCPIP_MAC_PacketRx(  
    TCPIP_MAC_HANDLE hMac,  
    TCPIP_MAC_RES* pRes,  
    const TCPIP_MAC_PACKET_RX_STAT** ppPktStat  
);
```

Description

TCPIP_MAC_PacketRx

This is the MAC receive function.

Once a pending packet is available in the MAC driver internal RX queues this function will dequeue the packet and hand it over to the MAC driver's client - i.e. the stack - for further processing.

The flags for a RX packet have to be updated by the MAC driver:

- TCPIP_MAC_PKT_FLAG_RX has to be set If the MAC supports it, it should set:
 - TCPIP_MAC_PKT_FLAG_UNICAST has to be set if that packet is a unicast packet
 - TCPIP_MAC_PKT_FLAG_BCAST has to be set if that packet is a broadcast packet
 - TCPIP_MAC_PKT_FLAG_MCAST has to be set if that packet is a multicast packet
 - TCPIP_MAC_PKT_FLAG_QUEUED has to be set
 - TCPIP_MAC_PKT_FLAG_SPLIT has to be set if the packet has multiple data segments
- Additional information about the packet is available by providing the pRes and ppPktStat fields.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
pRes	optional pointer to an address that will receive an additional result associated with the operation. Can be 0 if not needed.
ppPktStat	optional pointer to an address that will receive the received packet status. Note that this pointer cannot be used once the packet acknowledgement function was called. Can be 0 if not needed.

Returns

a valid pointer to an available RX packet 0 if no packet pending/available

Remarks

The MAC driver should dequeue and return to the caller just one single packet, and not multiple chained packets!

Once the higher level layers in the stack are done with processing the RX packet, they have to call the corresponding packet acknowledgement function that tells the owner of that packet that it can resume control of that packet.

Once the stack modules are done processing the RX packets and the acknowledge function is called it is up to the driver design to reuse the RX packets, or simply return them to the pool they were allocated from (assuming that some sort of allocation is implemented). This document makes no requirement about how the MAC RX packets are obtained, using dynamic or static allocation techniques. This is up to the design of the MAC.

The MAC driver can use the [TCPIP_MAC_Process\(\)](#) for obtaining new RX packets if needed.

Not all the MACs have hardware support for the received packet status. If the MAC driver cannot supply the [TCPIP_MAC_PACKET_RX_STAT](#) info, it should set the ppPktStat to 0.

5.8.16.7.1.10 TCPIP_MAC_PacketTx Function**C**

```
TCPIP_MAC_RES TCPIP_MAC_PacketTx(
    TCPIP_MAC_HANDLE hMac,
    TCPIP_MAC_PACKET * ptrPacket
);
```

Description**TCPIP_MAC_PacketTx**

This is the MAC transmit function. The MAC driver has to support internal queueing! A packet is to be rejected only if it's not properly formatted. Otherwise it has to be scheduled for transmission in an internal queue!

Once the packet is scheduled for transmission the MAC driver has to set the `TCPIP_MAC_PKT_FLAG_QUEUED` flag so that the stack is aware that this packet is under processing and cannot be modified!

Once the packet is transmitted, the `TCPIP_MAC_PKT_FLAG_QUEUED` has to be cleared, the proper packet acknowledgement result (`ackRes`) has to be set and the packet acknowledgement function (`ackFunc`) has to be called. It is implementation dependant if all these steps are implemented as part of the `ackFunc` itself or as discrete steps.

Parameters

Parameters	Description
<code>hMac</code>	handle identifying the MAC driver client
<code>ptrPacket</code>	pointer to a TCPIP_MAC_PACKET that's completely formatted and ready to be transmitted over the network

Remarks

See notes for the `segLoadOffset` member.

On 32 bit machines the 1st segment payload of a packet is allocated so that it is always 32 bit aligned and its size is 32 bits multiple. The `segLoadOffset` adds to the payload address and insures that the network layer data is 32 bit aligned.

- PIC32 MAC driver specific : the driver checks that the `segLoadOffset` ≥ 2 .

The packet is not required to contain the Frame Check Sequence (FCS/CRC32) field. The MAC driver/controller will insert that field itself, if it's required.

The MAC driver is required to support the transmission of multiple chained packets.

5.8.16.7.1.11 TCPIP_MAC_Process Function**C**

```
TCPIP_MAC_RES TCPIP_MAC_Process(
    TCPIP_MAC_HANDLE hMac
);
```

Description**TCPIP_MAC_Process**

This is a function that allows for internal processing by the MAC driver. It is meant for processing that cannot be done from within ISR.

Some of the processing that this is intended for:

- the MAC driver can process its pending TX queues (although it should do that preferably from within the TX ISR)
- RX buffers replenishing. If the number of packets in the RX queue falls below a specified limit, the MAC driver can use this function to allocate some extra RX packets. Similarly, if there are too many allocated RX packets, the MAC driver can free some of them.

Normally this function will be called in response to an TX and/or RX event signalled by the driver. This is specified by the MAC driver at initialization time using [TCPIP_MAC_MODULE_CTRL](#).

An alternative approach is that the MAC driver uses a system service to create a timer signal that will call the `TCPIP_MAC_Process` on a periodic basis.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client

Returns

TCPIP_MAC_RES_OK if all processing went on OK a [TCPIP_MAC_RES](#) error code if processing failed for some reason

Remarks

None.

5.8.16.7.1.12 TCPIP_MAC_Reinitialize Function

C

```
TCPIP_MAC_RES TCPIP_MAC_Reinitialize(
    const TCPIP_MAC_MODULE_CTRL* const stackData,
    const void* const moduleData
);
```

Description

MAC re-initialization function Allows re-initialization of the MAC with different power modes, etc. probably the same signature as [TCPIP_MAC_Initialize](#) Optional function

5.8.16.7.1.13 TCPIP_MAC_RxFilterHashTableEntrySet Function

C

```
TCPIP_MAC_RES TCPIP_MAC_RxFilterHashTableEntrySet(
    TCPIP_MAC_HANDLE hMac,
    TCPIP_MAC_ADDR* DestMACAddr
);
```

Description

set the hash table filter

5.8.16.7.2 Data Types and Constants**5.8.16.7.2.1 TCPIP_MAC_ACTION Enumeration**

C

```
typedef enum {
    TCPIP_MAC_ACTION_INIT,
    TCPIP_MAC_ACTION_REINIT,
    TCPIP_MAC_ACTION_DEINIT,
    TCPIP_MAC_ACTION_IF_UP,
    TCPIP_MAC_ACTION_IF_DOWN
} TCPIP_MAC_ACTION;
```

Description

- Interface functions

network interface action for initialization/de-initialization

Members

Members	Description
TCPIP_MAC_ACTION_INIT	stack is initialized
TCPIP_MAC_ACTION_REINIT	stack is re-initialized
TCPIP_MAC_ACTION_DEINIT	stack is de-initialized
TCPIP_MAC_ACTION_IF_UP	interface is brought up
TCPIP_MAC_ACTION_IF_DOWN	interface is brought down

5.8.16.7.2.2 TCPIP_MAC_ADDR Structure**C**

```
typedef struct {
    uint8_t v[6];
} TCPIP_MAC_ADDR;
```

Description

Structure to contain a MAC address

5.8.16.7.2.3 TCPIP_MAC_DATA_SEGMENT Type**C**

```
typedef struct _tag_MAC_DATA_SEGMENT TCPIP_MAC_DATA_SEGMENT;
```

Description

TCPIP MAC Data Segment

Structure of a segment buffer transferred with the MAC. A MAC TX or RX packet can consist of multiple data segments. On TX the MAC has to be able to transmit packets that span multiple data segments. On RX of a network frame the MAC may have to use multiple segments to construct a packet. (For performance reasons, a contiguous MAC packet, with just one segment, if possible, is preferred).

Remarks

See notes for the segLoadOffset member.

On 32 bit machines the segment payload is allocated so that it is always 32 bit aligned and its size is 32 bits multiple. The segLoadOffset adds to the payload address and insures that the network layer data is 32 bit aligned.

5.8.16.7.2.4 TCPIP_MAC_ETHERNET_HEADER Structure**C**

```
typedef struct {
    TCPIP_MAC_ADDR DestMACAddr;
    TCPIP_MAC_ADDR SourceMACAddr;
    uint16_t Type;
} TCPIP_MAC_ETHERNET_HEADER;
```

Description

A generic structure representing the Ethernet header starting all Ethernet frames

5.8.16.7.2.5 TCPIP_MAC_EVENT Enumeration**C**

```
typedef enum {
```

```
TCPIP_MAC_EV_NONE = 0x0000,
TCPIP_MAC_EV_RX_PKTEND = 0x0001,
TCPIP_MAC_EV_RX_OVFLOW = 0x0002,
TCPIP_MAC_EV_RX_BUFNA = 0x0004,
TCPIP_MAC_EV_RX_ACT = 0x0008,
TCPIP_MAC_EV_RX_DONE = 0x0010,
TCPIP_MAC_EV_RX_FWMARK = 0x0020,
TCPIP_MAC_EV_RX_EWMARK = 0x0040,
TCPIP_MAC_EV_RX_BUSERR = 0x0080,
TCPIP_MAC_EV_TX_DONE = 0x0100,
TCPIP_MAC_EV_TX_ABORT = 0x0200,
TCPIP_MAC_EV_TX_BUSERR = 0x0400,
TCPIP_MAC_EV_CONN_ESTABLISHED = 0x0800,
TCPIP_MAC_EV_CONN_LOST = 0x1000,
TCPIP_MAC_EV_RX_ALL =
(TCPIP_MAC_EV_RX_PKTEND|TCPIP_MAC_EV_RX_OVFLOW|TCPIP_MAC_EV_RX_BUFNA|TCPIP_MAC_EV_RX_ACT|
TCPIP_MAC_EV_RX_DONE|TCPIP_MAC_EV_RX_FWMARK|TCPIP_MAC_EV_RX_EWMARK|TCPIP_MAC_EV_RX_BUSERR),
TCPIP_MAC_EV_TX_ALL = (TCPIP_MAC_EV_TX_DONE|TCPIP_MAC_EV_TX_ABORT|TCPIP_MAC_EV_TX_BUSERR),
TCPIP_MAC_EV_RTX_ERRORS =
(TCPIP_MAC_EV_RX_OVFLOW|TCPIP_MAC_EV_RX_BUFNA|TCPIP_MAC_EV_RX_BUSERR|TCPIP_MAC_EV_TX_ABORT|TCPIP_MAC_EV_TX_BUSERR),
TCPIP_MAC_EV_CONN_ALL = (TCPIP_MAC_EV_CONN_ESTABLISHED|TCPIP_MAC_EV_CONN_LOST)
} TCPIP_MAC_EVENT;
```

Description

TCPIP MAC Events Codes

This enumeration defines all the possible events that can be reported by the MAC to the stack.

Depending on the type of the hardware Ethernet/WiFi interface, etc., not all events are possible.

Members

Members		Description
TCPIP_MAC_EV_NONE = 0x0000		no event

```
TCPIP_MAC_EV_RX_PKTEND = 0x0001
```

R
X

t
r
i
g
g
e
r
e
d

e
v
e
n
t
s

A

r
e
c
e
i
v
e

p
a
c
k
e
t

i
s

p
e
n
d
i
n
g

5.8.16.7.2.6 TCPIP_MAC_EventF Type

C

```
typedef void (* TCPIP_MAC_EventF)(TCPIP_MAC_EVENT event, const void* eventParam);
```

Description

event notification Function: typedef void (*TCPIP_MAC_EventF)([TCPIP_MAC_EVENT](#) event, const void* eventParam);

This function describes the MAC event notification handler. This is a handler specified by the user of the MAC (the TCP/IP stack). The stack can use the handler to be notified of MAC events. Whenever a notification occurs the passed events have to be eventually processed:

- Stack should process the TCPIP_EV_RX_PKTEND/TCPIP_EV_RX_DONE, TCPIP_EV_TX_DONE events
- Process the specific (error) condition
- Acknowledge the events by calling [TCPIP_MAC_EventAcknowledge\(\)](#) so that they can be re-enabled.

Preconditions

None

Parameters

Parameters	Description
event	event that's reported (multiple events can be OR-ed)
eventParam	user parameter that's used in the notification handler

Returns

None

Remarks

The notification handler will be called from the ISR which detects the corresponding event. The event notification handler has to be kept as short as possible and non-blocking. Mainly useful for RTOS integration where this handler will wake-up a thread that waits for a MAC event to occur.

The event notification system also enables the user of the TCPIP stack to call into the stack for processing only when there are relevant events rather than being forced to periodically call from within a loop at unknown moments.

Without a notification handler the stack user can still call [TCPIP_MAC_EventPendingGet\(\)](#) to see if processing by the stack needed.

This is a default way of adding MAC interrupt processing to the TCP/IP stack.

5.8.16.7.2.7 TCPIP_MAC_HANDLE Type

C

```
typedef const void* TCPIP_MAC_HANDLE;
```

Description

handle to a MAC

5.8.16.7.2.8 TCPIP_MAC_HEAP_CallocF Type

C

```
typedef void* (* TCPIP_MAC_HEAP_CallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nElems, size_t elemSize);
```

Description

calloc function

5.8.16.7.2.9 TCPIP_MAC_HEAP_FreeF Type

C

```
typedef int (* TCPIP_MAC_HEAP_FreeF)(TCPIP_MAC_HEAP_HANDLE heapH, const void* pBuffer);
```

Description

free function

5.8.16.7.2.10 TCPIP_MAC_HEAP_HANDLE Type

C

```
typedef const void* TCPIP_MAC_HEAP_HANDLE;
```

Description

handle to a heap

5.8.16.7.2.11 TCPIP_MAC_HEAP_MallocF Type

C

```
typedef void* (* TCPIP_MAC_HEAP_MallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nBytes);
```

Description

malloc function

5.8.16.7.2.12 TCPIP_MAC_MODULE_CTRL Structure

C

```
typedef struct {
    int nIfs;
    TCPIP_MAC_HEAP_MallocF mallocF;
    TCPIP_MAC_HEAP_CallocF callocF;
    TCPIP_MAC_HEAP_FreeF freeF;
    TCPIP_MAC_HEAP_HANDLE memH;
    TCPIP_MAC_PKT_AllocF pktAllocF;
    TCPIP_MAC_PKT_FreeF pktFreeF;
    TCPIP_MAC_PKT_AckF pktAckF;
    TCPIP_MAC_EventF eventF;
    const void* eventParam;
    unsigned int moduleId;
    int netIx;
    TCPIP_MAC_ACTION macAction;
    TCPIP_MAC_POWER_MODE powerMode;
    TCPIP_MAC_ADDR ifPhyAddress;
    TCPIP_MAC_PROCESS_FLAGS processFlags;
} TCPIP_MAC_MODULE_CTRL;
```

Description

data that's passed as reference to MAC driver module init/deinit.

Members

Members	Description
int nIfs;	permanent data; this data is maintained by the stack for one full session i.e. accross Initialize() -> DeInitialize() calls number of the interfaces supported in this seession
TCPIP_MAC_HEAP_MallocF mallocF;	transient data; contains info for a specific module call allocation functions/parameters
TCPIP_MAC_HEAP_HANDLE memH;	handle to be used in the stack allocation service calls
TCPIP_MAC_PKT_AllocF pktAllocF;	packet allocation functions
TCPIP_MAC_EventF eventF;	function to be used by the MAC for event reporting
const void* eventParam;	parameter to be used when the event function is called
unsigned int moduleId;	module identifier; allows multiple channels/ports, etc. MAC support
int netIx;	index of the current interface

TCPIP_MAC_ACTION macAction;	current action for the MAC/stack
TCPIP_MAC_POWER_MODE powerMode;	the power mode for this interface to go to valid only if stackAction == init/re-init; ignored for de-init
TCPIP_MAC_ADDR ifPhyAddress;	physical address of the interface

5.8.16.7.2.13 TCPIP_MAC_PACKET Type

C

```
typedef struct _tag_TCPIP_MAC_PACKET TCPIP_MAC_PACKET;
```

Description

forward declaration

5.8.16.7.2.14 TCPIP_MAC_PACKET_ACK_FUNC Type

C

```
typedef void (* TCPIP_MAC_PACKET_ACK_FUNC)(TCPIP_MAC_PACKET* pkt, const void* param);
```

Description

This is type TCPIP_MAC_PACKET_ACK_FUNC.

5.8.16.7.2.15 TCPIP_MAC_PACKET_FLAGS Enumeration

C

```
typedef enum {
    TCPIP_MAC_PKT_FLAG_STATIC = 0x0001,
    TCPIP_MAC_PKT_FLAG_TX = 0x0002,
    TCPIP_MAC_PKT_FLAG_SPLIT = 0x0004,
    TCPIP_MAC_PKT_FLAG_QUEUED = 0x0008,
    TCPIP_MAC_PKT_FLAG_UNICAST = 0x0010,
    TCPIP_MAC_PKT_FLAG_BCAST = 0x0020,
    TCPIP_MAC_PKT_FLAG_MCAST = 0x0040,
    TCPIP_MAC_PKT_FLAG_CAST_MASK = 0x0070,
    TCPIP_MAC_PKT_FLAG_CAST_DISABLED = 0x0000,
    TCPIP_MAC_PKT_FLAG_USER = 0x0100
} TCPIP_MAC_PACKET_FLAGS;
```

Description

16 bits only packet flags!

Members

Members	Description
TCPIP_MAC_PKT_FLAG_STATIC = 0x0001	packet/segment can not be dynamically deallocated set when the packet is allocated
TCPIP_MAC_PKT_FLAG_TX = 0x0002	if set, it's an TX packet/segment else is an RX packet
TCPIP_MAC_PKT_FLAG_SPLIT = 0x0004	packet data spans multiple segments - ZC functionality if not set the packet has only one data segment set by the MAC driver when a RX packet spans multiple data segments
TCPIP_MAC_PKT_FLAG_QUEUED = 0x0008	packet data is queued somewhere, cannot be freed flag is set by the packet source when the packet is passed for further processing to a destination cleared by the packet destination when the packet processing was completed.
TCPIP_MAC_PKT_FLAG_UNICAST = 0x0010	MAC updated: it's an unicast packet

TCPIP_MAC_PKT_FLAG_BCAST = 0x0020	MAC updated: it's an broadcast packet
TCPIP_MAC_PKT_FLAG_MCAST = 0x0040	MAC updated: it's an multicast packet
TCPIP_MAC_PKT_FLAG_CAST_MASK = 0x0070	packet cast mask bits
TCPIP_MAC_PKT_FLAG_CAST_DISABLED = 0x0000	if all zeroes MAC packet MCAST/BCAST fields are not updated by the MAC RX process
TCPIP_MAC_PKT_FLAG_USER = 0x0100	available user flags start here

5.8.16.7.2.16 TCPIP_MAC_PACKET_RX_STAT Structure

C

```
typedef struct {
    unsigned chksumOk : 1;
    unsigned pktChecksum : 16;
    unsigned runtPkt : 1;
    unsigned notMeUcast : 1;
    unsigned htMatch : 1;
    unsigned magicMatch : 1;
    unsigned pmMatch : 1;
    unsigned uMatch : 1;
    unsigned bMatch : 1;
    unsigned mMatch : 1;
    unsigned rxBytes : 16;
    unsigned crcError : 1;
    unsigned lenError : 1;
    unsigned lenRange : 1;
    unsigned rxOk : 1;
    unsigned mcast : 1;
    unsigned bcast : 1;
    unsigned rxCtrl : 1;
    unsigned rxVLAN : 1;
} TCPIP_MAC_PACKET_RX_STAT;
```

Description

Received Packet Status

This structure contains the status of a received packet.

Members

Members	Description
unsigned chksumOk : 1;	correct checksum filled in
unsigned pktChecksum : 16;	Packet payload checksum
unsigned runtPkt : 1;	Runt packet received
unsigned notMeUcast : 1;	Unicast, not me packet,
unsigned htMatch : 1;	Hash table match
unsigned magicMatch : 1;	Magic packet match
unsigned pmMatch : 1;	Pattern match match
unsigned uMatch : 1;	Unicast match
unsigned bMatch : 1;	Broadcast match
unsigned mMatch : 1;	Multicast match
unsigned rxBytes : 16;	Received bytes
unsigned crcError : 1;	CRC error in packet
unsigned lenError : 1;	Receive length check error

unsigned lenRange : 1;	Receive length out of range
unsigned rxOk : 1;	Receive OK
unsigned mcast : 1;	Multicast packet
unsigned bcast : 1;	Broadcast packet
unsigned rxCtrl : 1;	Control frame received
unsigned rxVLAN : 1;	Received VLAN tagged frame

Remarks

Not all the MACs have hardware support for the received packet status.

5.8.16.7.2.17 TCPIP_MAC_PKT_ACK_RES Enumeration**C**

```
typedef enum {
    TCPIP_MAC_PKT_ACK_NONE = 0,
    TCPIP_MAC_PKT_ACK_TX_OK = 1,
    TCPIP_MAC_PKT_ACK_RX_OK = 2,
    TCPIP_MAC_PKT_ACK_LINK_DOWN = -1,
    TCPIP_MAC_PKT_ACK_NET_DOWN = -2,
    TCPIP_MAC_PKT_ACK_BUFFER_ERR = -3,
    TCPIP_MAC_PKT_ACK_ARP_TMO = -4,
    TCPIP_MAC_PKT_ACK_ARP_NET_ERR = -5,
    TCPIP_MAC_PKT_ACK_CHKSUM_ERR = -10,
    TCPIP_MAC_PKT_ACK_SOURCE_ERR = -11,
    TCPIP_MAC_PKT_ACK_TYPE_ERR = -12,
    TCPIP_MAC_PKT_ACK_STRUCT_ERR = -13,
    TCPIP_MAC_PKT_ACK_PROTO_DEST_ERR = -14,
    TCPIP_MAC_PKT_ACK_FRAGMENT_ERR = -15,
    TCPIP_MAC_PKT_ACK_PROTO_DEST_CLOSE = -16,
    TCPIP_MAC_PKT_ACK_ALLOC_ERR = -17
} TCPIP_MAC_PKT_ACK_RES;
```

Description

list of return codes for a packet acknowledge function 16 bits only acknowledge results!

Members

Members	Description
TCPIP_MAC_PKT_ACK_NONE = 0	packet result unknown, unspecified TX success code - positive
TCPIP_MAC_PKT_ACK_TX_OK = 1	packet was transmitted successfully RX success code - positive
TCPIP_MAC_PKT_ACK_RX_OK = 2	packet was received/processed successfully TX error codes - negative
TCPIP_MAC_PKT_ACK_LINK_DOWN = -1	packet was dropped because the link was down
TCPIP_MAC_PKT_ACK_NET_DOWN = -2	packet was dropped because the network is down
TCPIP_MAC_PKT_ACK_BUFFER_ERR = -3	packet was dropped because the buffer type is not supported
TCPIP_MAC_PKT_ACK_ARP_TMO = -4	packet was dropped because of an ARP timeout
TCPIP_MAC_PKT_ACK_ARP_NET_ERR = -5	packet associated interface is down or non existent RX error codes - negative
TCPIP_MAC_PKT_ACK_CHKSUM_ERR = -10	packet was dropped because the checksum was incorrect
TCPIP_MAC_PKT_ACK_SOURCE_ERR = -11	packet was dropped because of wrong interface source address
TCPIP_MAC_PKT_ACK_TYPE_ERR = -12	packet was dropped because the type was unknown
TCPIP_MAC_PKT_ACK_STRUCT_ERR = -13	internal packet structure error

TCPIP_MAC_PKT_ACK_PROTO_DEST_ERR = -14	the packet protocol couldn't find a destination for it
TCPIP_MAC_PKT_ACK_FRAGMENT_ERR = -15	the packet too fragmented
TCPIP_MAC_PKT_ACK_PROTO_DEST_CLOSE = -16	the packet destination is closing
TCPIP_MAC_PKT_ACK_ALLOC_ERR = -17	memory allocation error

5.8.16.7.2.18 TCPIP_MAC_PKT_AckF Type

C

```
typedef void (* TCPIP_MAC_PKT_AckF)(TCPIP_MAC_PACKET* pPkt, TCPIP_MAC_PKT_ACK_RES ackRes);
```

Description

acknowledges a packet clears the TCPIP_MAC_PKT_FLAG_QUEUED flag a packet should always have an acknowledgment function packet's ackRes is updated only if the parameter ackRes != TCPIP_MAC_PKT_ACK_NONE.

5.8.16.7.2.19 TCPIP_MAC_PKT_AllocF Type

C

```
typedef TCPIP_MAC_PACKET* (* TCPIP_MAC_PKT_AllocF)(uint16_t pktLen, uint16_t segLoadLen, TCPIP_MAC_PACKET_FLAGS flags);
```

Description

allocates a [TCPIP_MAC_PACKET](#) packet ([TCPIP_MAC_ETHERNET_HEADER](#) always added); pktLen - the size of the packet (it will be 32 bits rounded up) segLoadLen - the payload size for the segment associated to this packet; Payload is always 32 bit aligned if 0 no segment is created/attached to the packet

5.8.16.7.2.20 TCPIP_MAC_PKT_FreeF Type

C

```
typedef void (* TCPIP_MAC_PKT_FreeF)(TCPIP_MAC_PACKET* pPkt);
```

Description

frees a previously allocated packet that neither the packet nor segments marked with TCPIP_MAC_PKT_FLAG_STATIC/TCPIP_MAC_SEG_FLAG_STATIC are not freed Also note that this function does not free explicetely the external segment payload. A payload that was created contiguously when the segment was created will be automatically freed by this function.

5.8.16.7.2.21 TCPIP_MAC_POWER_MODE Enumeration

C

```
typedef enum {
    TCPIP_MAC_POWER_NONE,
    TCPIP_MAC_POWER_FULL,
    TCPIP_MAC_POWER_LOW,
    TCPIP_MAC_POWER_DOWN
} TCPIP_MAC_POWER_MODE;
```

Description

supported MAC power mode state

Members

Members	Description
TCPIP_MAC_POWER_NONE	unknown power mode;
TCPIP_MAC_POWER_FULL	up and running; valid for init/re-init
TCPIP_MAC_POWER_LOW	low power mode; valid for init/re-init
TCPIP_MAC_POWER_DOWN	interface is down;

5.8.16.7.2.22 TCPIP_MAC_PROCESS_FLAGS Enumeration**C**

```
typedef enum {
    TCPIP_MAC_PROCESS_FLAG_NONE = 0x0000,
    TCPIP_MAC_PROCESS_FLAG_RX = 0x0001,
    TCPIP_MAC_PROCESS_FLAG_TX = 0x0002,
    TCPIP_MAC_PROCESS_FLAG_ANY = 0x0100
} TCPIP_MAC_PROCESS_FLAGS;
```

Description

specific MAC process flags multiple flags can be orred

Members

Members	Description
TCPIP_MAC_PROCESS_FLAG_NONE = 0x0000	the stack never has to call the TCPIP_MAC_Process function
TCPIP_MAC_PROCESS_FLAG_RX = 0x0001	the stack has to call the TCPIP_MAC_Process after an RX signal
TCPIP_MAC_PROCESS_FLAG_TX = 0x0002	the stack has to call the TCPIP_MAC_Process after an TX signal
TCPIP_MAC_PROCESS_FLAG_ANY = 0x0100	the stack has to call the TCPIP_MAC_Process after any type of signal

5.8.16.7.2.23 TCPIP_MAC_RES Enumeration**C**

```
typedef enum {
    TCPIP_MAC_RES_OK = 0,
    TCPIP_MAC_RES_PENDING = 1,
    TCPIP_MAC_RES_TYPE_ERR = -1,
    TCPIP_MAC_RES_IS_BUSY = -2,
    TCPIP_MAC_RES_INIT_FAIL = -3,
    TCPIP_MAC_RES_PHY_INIT_FAIL = -4,
    TCPIP_MAC_RES_EVENT_INIT_FAIL = -5,
    TCPIP_MAC_RES_OP_ERR = -6,
    TCPIP_MAC_RES_ALLOC_ERR = -7,
    TCPIP_MAC_RES_INSTANCE_ERR = -8,
    TCPIP_MAC_RES_FRAGMENT_ERR = -9,
    TCPIP_MAC_RES_PACKET_ERR = -10,
    TCPIP_MAC_RES_QUEUE_TX_FULL = -11
} TCPIP_MAC_RES;
```

Description

list of return codes from MAC functions

Members

Members	Description
TCPIP_MAC_RES_OK = 0	operation successful benign operation results - positive codes
TCPIP_MAC_RES_PENDING = 1	operation is pending upon some hardware resource call again to completion error codes - negative
TCPIP_MAC_RES_TYPE_ERR = -1	unsupported type
TCPIP_MAC_RES_IS_BUSY = -2	device is in use
TCPIP_MAC_RES_INIT_FAIL = -3	generic initialization failure
TCPIP_MAC_RES_PHY_INIT_FAIL = -4	PHY initialization failure
TCPIP_MAC_RES_EVENT_INIT_FAIL = -5	Event system initialization failure
TCPIP_MAC_RES_OP_ERR = -6	unsupported operation
TCPIP_MAC_RES_ALLOC_ERR = -7	memory allocation error
TCPIP_MAC_RES_INSTANCE_ERR = -8	already instantiated, initialized error
TCPIP_MAC_RES_FRAGMENT_ERR = -9	too fragmented, RX buffer too small
TCPIP_MAC_RES_PACKET_ERR = -10	unsupported/corrupted packet error
TCPIP_MAC_RES_QUEUE_TX_FULL = -11	TX queue exceeded the limit

5.8.16.7.2.24 TCPIP_MAC_SEGMENT_FLAGS Enumeration**C**

```
typedef enum {
    TCPIP_MAC_SEG_FLAG_STATIC = 0x0001,
    TCPIP_MAC_SEG_FLAG_TX = 0x0002,
    TCPIP_MAC_SEG_FLAG_USER = 0x0100
} TCPIP_MAC_SEGMENT_FLAGS;
```

Description

16 bits only segment flags!

Members

Members	Description
TCPIP_MAC_SEG_FLAG_STATIC = 0x0001	segment can not be dynamically deallocated set when the segment is allocated
TCPIP_MAC_SEG_FLAG_TX = 0x0002	if set, it's an TX segment else is an RX packet available user segment flags start here

5.8.16.7.2.25 TCPIP_MODULE_MAC_97J60_CONFIG Structure**C**

```
typedef struct {
} TCPIP_MODULE_MAC_97J60_CONFIG;
```

Description

This is type TCPIP_MODULE_MAC_97J60_CONFIG.

5.8.16.7.2.26 TCPIP_MODULE_MAC_ENCJ60_CONFIG Structure**C**

```
typedef struct {
} TCPIP_MODULE_MAC_ENCJ60_CONFIG;
```


Description

MAC initialization data Each supported MAC has its own specific init/configuration data

5.8.16.7.2.27 TCPIP_MODULE_MAC_ENCJ600_CONFIG Structure**C**

```
typedef struct {  
} TCPIP_MODULE_MAC_ENCJ600_CONFIG;
```

Description

This is type TCPIP_MODULE_MAC_ENCJ600_CONFIG.

5.8.16.7.2.28 TCPIP_MODULE_MAC_MRF24W_CONFIG Structure**C**

```
typedef struct {  
} TCPIP_MODULE_MAC_MRF24W_CONFIG;
```

Description

This is type TCPIP_MODULE_MAC_MRF24W_CONFIG.

5.8.16.7.2.29 TCPIP_MODULE_MAC_PIC32INT_CONFIG Structure**C**

```
typedef struct {  
    int nTxDescriptors;  
    int rxBuffSize;  
    int nRxDescriptors;  
} TCPIP_MODULE_MAC_PIC32INT_CONFIG;
```

Description

This is type TCPIP_MODULE_MAC_PIC32INT_CONFIG.

Members

Members	Description
int nTxDescriptors;	number of TX descriptors
int rxBuffSize;	size of the corresponding RX buffer
int nRxDescriptors;	number of RX descriptors

5.8.16.8 Files**Files**

Name	Description
tcpip_mac.h	
tcpip_mac_config.h	configuration file

Description

5.8.16.8.1 tcpip_mac.h

MAC Module Defs for Microchip Stack

Enumerations

	Name	Description

5.8.16.8.2 tcpip_mac_config.h

MAC Configuration file

This file contains the MAC module configuration options

Macros

	Name	Description
	<u>EMAC_RX_BUFF_SIZE</u>	size of a RX buffer. should be multiple of 16 this is the size of all receive buffers processed by the ETHC The size should be enough to accomodate any network received packet If the packets are larger, they will have to take multiple RX buffers and the packet manipulation is less efficient #define EMAC_RX_BUFF_SIZE 512
	<u>EMAC_RX_DESCRIPTOR</u>	number of the RX descriptors and RX buffers to be created
	<u>EMAC_RX_FRAGMENTS</u>	MAC maximum number of supported fragments Based on the values of <u>EMAC_RX_MAX_FRAME</u> and <u>EMAC_RX_BUFF_SIZE</u> an incoming frame may span multiple RX buffers (fragments). Note that excessive fragmentation leads to performance degradation. The default and recommended value should be 1. #define EMAC_RX_FRAGMENTS 1 Alternatively you can use the calculation of the number of fragments based on the selected RX sizes:
	<u>EMAC_RX_MAX_FRAME</u>	Maximum MAC supported RX frame size. Any incoming ETH frame that's longer than this size will be discarded. The default value is 1536 (allows for VLAN tagged frames, although the VLAN tagged frames are discarded) Normally you shouldn't need to touch this value unless you know exactly the maximum size of the frames you want to process or you need to control packets fragmentation (together with the <u>EMAC_RX_BUFF_SIZE</u>).
	<u>EMAC_TX_DESCRIPTOR</u>	number of the TX descriptors to be created
	<u>ETH_CFG_10</u>	use/advertise 10 Mbps capability
	<u>ETH_CFG_100</u>	use/advertise 100 Mbps capability
	<u>ETH_CFG_AUTO</u>	use auto negotiation
	<u>ETH_CFG_AUTO_MDIX</u>	use/advertise auto MDIX capability
	<u>ETH_CFG_FDUPLEX</u>	use/advertise full duplex capability
	<u>ETH_CFG_HDUPLEX</u>	use/advertise half duplex capability
	<u>ETH_CFG_LINK</u>	set to 1 if you need to config the link to specific following parameters otherwise the default connection will be attempted depending on the selected PHY
	<u>ETH_CFG_LINK_INIT_DELAY</u>	some PHYs need an initialization delay for insuring that the PHY is ready to transmit data. milliseconds
	<u>ETH_CFG_SWAP_MDIX</u>	use swapped MDIX. else normal MDIX

5.8.17 Manager TCP/IP Stack Library

5.8.17.1 Introduction

Manager TCP/IP Stack Library for Microchip Microcontrollers

This library provides the API of the Manager TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Handles internal RX packet pre-processing prior to dispatching to upper application layers.

5.8.17.2 Release Notes

MPLAB Harmony Version: v0.70b **Manager TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.17.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.17.4 Using the Library

This topic describes the basic architecture of the Manager TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [tcpip_manager.h](#)

The interface to the Manager TCPIP Stack library is defined in the "[tcpip_manager.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the Manager TCP/IP Stack library should include "tcpip.h".

Library File:

The Manager TCPIP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

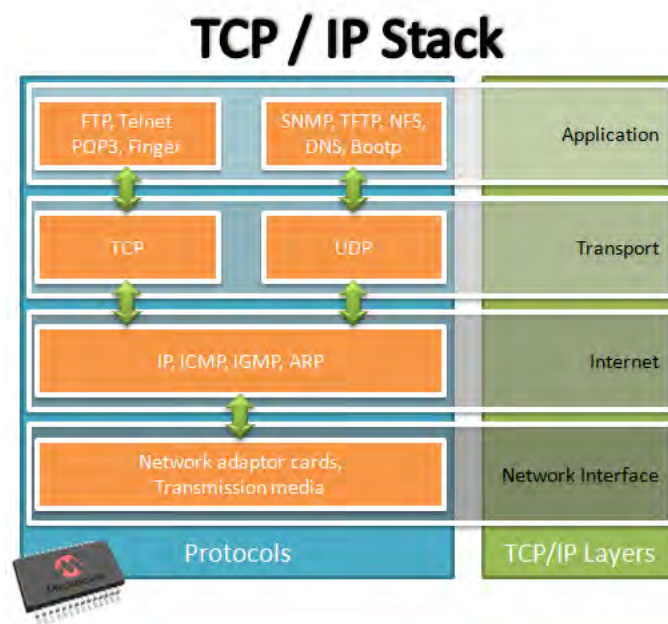
5.8.17.4.1 Abstraction Model

This library provides the API of the Manager TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Manager Software Abstraction Block Diagram

Handles internal RX packet pre-processing prior to dispatching to upper application layers.



5.8.17.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the MAC module.

Library Interface Section	Description
Task and Initialize Functions	Routines to Initialize and De-Initialize this Module
Network Up/Down/Linked Functions	Routines to Set and Get the Network's Up/Down Status
Network Status and Control Functions	Routines to Set and Get the Various Network Controls
Network Address Status and Control Functions	Routines to Set and Get the Network's Address Status
Network Structure Storage Functions	Network Information Routines.

5.8.17.5 Configuring the Library

The configuration of the Manager TCP/IP Stack is based on the file `sys_config.h`

This header file contains the configuration selection for the Manager TCP/IP Stack. Based on the selections made, the Manager TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the Manager TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.17.6 Building the Library

This section list the files that are available in the `\src` of the Manager driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.17.7 Library Interface

Data Types and Constants

	Name	Description
	TCPIP_LOCAL_MASK_TYPE	types of masks used in the detection of a local/nonlocal network
	TCPIP_NET_HANDLE	a network interface handle

Network Address Status and Control Functions

	Name	Description
	TCPIP_STACK_ModuleGetConfig	This function returns the current configuration data of the stack module specified by the corresponding module ID.
	TCPIP_STACK_NetAddress	
	TCPIP_STACK_NetAddressBcast	
	TCPIP_STACK_NetAddressDnsPrimary	
	TCPIP_STACK_NetAddressDnsPrimarySet	
	TCPIP_STACK_NetAddressDnsSecond	
	TCPIP_STACK_NetAddressDnsSecondSet	
	TCPIP_STACK_NetAddressGateway	
	TCPIP_STACK_NetAddressGatewaySet	
	TCPIP_STACK_NetAddressMac	
	TCPIP_STACK_NetAddressMacSet	
	TCPIP_STACK_NetAddressSet	
	TCPIP_STACK_NetIPv6AddressGet	This function allows the listing of the IPv6 addresses associated with an interface.

Network Status and Control Functions

	Name	Description
	TCPIP_STACK_IndexToNet	

	TCPIP_STACK_NetBIOSName	
	TCPIP_STACK_NetBiosNameSet	
	TCPIP_STACK_NetDefaultGet	
	TCPIP_STACK_NetDefaultSet	
	TCPIP_STACK_NetHandleGet	
	TCPIP_STACK_NetIndexGet	
	TCPIP_STACK_NetMask	
	TCPIP_STACK_NetNameGet	
	TCPIP_STACK_NumberOfNetworksGet	

Network Structure Storage Functions

	Name	Description
	TCPIP_STACK_NetConfigGet	This function dumps the current configuration data of the network interface specified by the corresponding network handle into the supplied buffer.

5.8.17.7.1 Task and Initialize Functions

5.8.17.7.1.1 TCPIP_STACK_Deinitialize Function

C

```
void TCPIP_STACK_Deinitialize();
```

Description

This function performs the de-initialization of the TCPIP stack

Preconditions

None

Returns

None

Side Effects

None

Remarks

None

5.8.17.7.1.2 TCPIP_STACK_Initialize Function

C

```
bool TCPIP_STACK_Initialize(
    const TCPIP_NETWORK_CONFIG* pNetConf,
    int nNets,
    const TCPIP_STACK_MODULE_CONFIG* pModConfig,
    int nModules
);
```

Description

The function initializes the stack. If an error occurs, the SYS_ERROR() is called and the function de-initialize itself and will return false.

Preconditions

None

Parameters

Parameters	Description
pNetConf	pointer to an array of TCPIP_NETWORK_CONFIG to support
nNets	number of network configurations in the array
pModConfig	pointer to an array of TCPIP_STACK_MODULE_CONFIG
nModules	number of modules to initialize

Returns

true if Stack and its componets are initialized false otherwise

Side Effects

None

Remarks

This function must be called before any of the stack or its component routines are used.

New TCPIP_NETWORK_CONFIG types should be added/removed at run time for implementations that support dynamic network interface creation.

5.8.17.7.1.3 TCPIP_STACK_SetLocalMasks Function

C

```
bool TCPIP_STACK_SetLocalMasks(  
    TCPIP_NET_HANDLE netH,  
    uint32_t andMask,  
    uint32_t orMask  
);
```

Description

This function sets the masks used in the stack decision of a destination address being a local or nonlocal address. These masks will be used when the corresponding mask type is set to TCPIP_LOCAL_MASK_SET.

Preconditions

None

Parameters

Parameters	Description
netH	handle of the interface to use
andMask	AND mask to use for local network detection, big endian (BE) format
orType	OR mask to use for local network detection , BE format

Returns

if interface exists then this function will set the masks and return true. otherwise it will return false

Remarks

None

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT"); TCPIP_STACK_SetLocalMasks(netH, 0x0101a0c0,  
0x0);
```

5.8.17.7.1.4 TCPIP_STACK_SetLocalMasksType Function

C

```
bool TCPIP_STACK_SetLocalMasksType(  
    TCPIP_NET_HANDLE netH,  
    TCPIP_LOCAL_MASK_TYPE andType,  
    TCPIP_LOCAL_MASK_TYPE orType  
);
```

Description

This function sets the types of masks used in the stack decision of a destination address being a local or nonlocal address. For example, when a TCP connection is made, the advertised MSS usually has different values for local versus nonlocal networks.

In order to decide if a destination IP address (destAdd) is local to a network interface (having the netAdd address) the following calculation is performed: if $((\text{destAdd} \& \text{andMask}) | \text{orMask}) == ((\text{netAdd} \& \text{andMask}) | \text{orMask})$ then the destination address is considered to be a local address. else the destination address is nonlocal.

This function sets the types of masks used for the AND and OR operations, as follows:

- TCPIP_LOCAL_MASK_ZERO: use the all 0's mask (0x00000000)
- TCPIP_LOCAL_MASK_ONE: use the all 1's mask (0xffffffff)
- TCPIP_LOCAL_MASK_NET: use the current network mask
- TCPIP_LOCAL_MASK_SET: use a mask that's set by the application There are operations to set a specific AND and OR masks

Using different valuse for the AND and OR masks an application can select various destination networks to be considered as local/nonlocal:

- only the own network
- any network
- no network
- specific (range of) networks
- etc.

Preconditions

None

Parameters

Parameters	Description
netH	handle of the interface to use
andType	AND type of mask to use for local network detection
orType	OR type of mask to use for local network detection

Returns

if interface exists then this function will set the mask types and return true. otherwise it will return false

Remarks

The default value for both AND and OR masks is set to TCPIP_LOCAL_MASK_ZERO so that any destination network is considered to be local!

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT"); TCPIP_STACK_SetLocalMasksType(netH,  
TCPIP_LOCAL_MASK_NET, TCPIP_LOCAL_MASK_ZERO);
```


5.8.17.7.1.5 TCPIP_STACK_Task Function

C

```
void TCPIP_STACK_Task();
```

Preconditions

TCPIP_STACK_Initialize() is already called.

Returns

Stack Finite-state Machine (FSM) is executed.

Side Effects

None

Remarks

This FSM checks for new incoming packets, and routes it to appropriate stack components. It also performs timed operations.

This function must be called periodically to ensure timely responses.

5.8.17.7.2 Network Status and Control Functions

5.8.17.7.2.1 TCPIP_STACK_IndexToNet Function

C

```
TCPIP_NET_HANDLE TCPIP_STACK_IndexToNet(
    int netIx
);
```

Preconditions

TCPIP stack should have been initialized by TCPIP_STACK_Initialize()

Parameters

Parameters	Description
netH	Interface handle.

Returns

Resolves an index to an network handle.

Side Effects

None

Remarks

None

5.8.17.7.2.2 TCPIP_STACK_NetBIOSName Function

C

```
const char* TCPIP_STACK_NetBIOSName(
    TCPIP_NET_HANDLE netH
);
```

Preconditions

None

Parameters

Parameters	Description
netH	Interface handle to get name of.

Returns

if interface is enabled then it returns the NetBIOS name of that interface else return 0

Remarks

None

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT"); const char* biosName =  
TCPIP_STACK_NetBIOSName(netH);
```

5.8.17.7.2.3 TCPIP_STACK_NetBiosNameSet Function**C**

```
bool TCPIP_STACK_NetBiosNameSet(  
    TCPIP_NET_HANDLE netH,  
    const char* biosName  
);
```

Preconditions

None

Parameters

Parameters	Description
netH	Interface handle to set name of.

Returns

if interface exists then it sets the NetBIOS name of that interface and returns true else return false

Remarks

See Important Note above!

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT"); TCPIP_STACK_NetBiosNameSet(netH,  
myBiosName);
```

5.8.17.7.2.4 TCPIP_STACK_NetDefaultGet Function**C**

```
TCPIP_NET_HANDLE TCPIP_STACK_NetDefaultGet();
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Returns

The default net interface for multi-homed hosts

Side Effects

None

Remarks

Function to dynamically change the default interface will be added.

5.8.17.7.2.5 TCPIP_STACK_NetDefaultSet Function**C**

```
bool TCPIP_STACK_NetDefaultSet(  
    TCPIP_NET_HANDLE netH  
);
```

Preconditions

None

Parameters

Parameters	Description
netH	Interface handle.

Returns

true if success false if failed (the old interface does not change)

Side Effects

None

Remarks

sets the default interface

5.8.17.7.2.6 TCPIP_STACK_NetHandleGet Function**C**

```
TCPIP_NET_HANDLE TCPIP_STACK_NetHandleGet(  
    const char* interface  
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
interface	The names specified in tcpip_config.h::TCPIP_NETWORK_CONFIG.

Returns

Resolves an interface name to a handle.

Side Effects

None

Remarks

None

Example

```
TCPIP\_NET\_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
```

5.8.17.7.2.7 TCPIP_STACK_NetIndexGet Function

C

```
int TCPIP_STACK_NetIndexGet(  
    TCPIP_NET_HANDLE hNet  
);
```

Preconditions

None

Parameters

Parameters	Description
hNet	Interface handle.

Returns

Index of this entry in the stack network handles

Side Effects

None

Remarks

None

5.8.17.7.2.8 TCPIP_STACK_NetMask Function

C

```
uint32_t TCPIP_STACK_NetMask(  
    TCPIP_NET_HANDLE netH  
);
```

Preconditions

None

Parameters

Parameters	Description
netH	Interface handle to get mask of.

Returns

if interface is enabled then it returns the IP address mask of that interface else return 0

Remarks

None

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT"); uint32_t subMask = TCPIP_STACK_NetMask(netH);
```

5.8.17.7.2.9 TCPIP_STACK_NetNameGet Function

C

```
const char* TCPIP_STACK_NetNameGet(  
    TCPIP_NET_HANDLE netH  
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to get the name of.

Returns

it returns the name associated to that interface handle returns 0 if no such name

Side Effects

None

Remarks

None

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_IndexToNet(0); const char* netName = TCPIP\_STACK\_NetNameGet(netH);
```

5.8.17.7.2.10 TCPIP_STACK_NumberOfNetworksGet Function

C

```
int TCPIP\_STACK\_NumberOfNetworksGet();
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Returns

Number of network interfaces

Side Effects

None

Remarks

None

5.8.17.7.3 Network Up/Down/Linked Functions

5.8.17.7.3.1 TCPIP_STACK_NetDown Function

C

```
bool TCPIP\_STACK\_NetDown(  
    TCPIP\_NET\_HANDLE netH  
);
```

Description

This function performs the de-initialization of a net interface

Preconditions

None

Parameters

Parameters	Description
neth	Interface handle.

Returns

true if success false if no such network

Side Effects

None

Remarks

None

5.8.17.7.3.2 TCPIP_STACK_NetIsLinked Function**C**

```
bool TCPIP_STACK_NetIsLinked(  
    TCPIP_NET_HANDLE hNet  
);
```

Preconditions

None

Parameters

Parameters	Description
hNet	Interface handle.

Returns

true if interface exists and the corresponding MAC is linked false otherwise

Side Effects

None

Remarks

None

5.8.17.7.3.3 TCPIP_STACK_NetIsUp Function**C**

```
bool TCPIP_STACK_NetIsUp(  
    TCPIP_NET_HANDLE hNet  
);
```

Preconditions

None

Parameters

Parameters	Description
hNet	Interface handle.

Returns

true if interface exists and is enabled false otherwise

Side Effects

None

Remarks

None

5.8.17.7.3.4 TCPIP_STACK_NetUp Function

C

```
bool TCPIP_STACK_NetUp(
    TCPIP_NET_HANDLE neth,
    const TCPIP_NETWORK_CONFIG* pUsrConfig
);
```

Description

This function brings the desired interface up.

Preconditions

None

Parameters

Parameters	Description
neth	Interface handle.
pUsrConfig	pointer to a TCPIP_NETWORK_CONFIG for the interface initialization

Returns

true if success false if no such network or an error occurred

Side Effects

None

Remarks

None

5.8.17.7.4 Network Address Status and Control Functions

5.8.17.7.4.1 TCPIP_STACK_ModuleGetConfig Function

C

```
size_t TCPIP_STACK_ModuleGetConfig(
    TCPIP_STACK_MODULE modId,
    void* configBuff,
    size_t buffSize,
    size_t* pNeededSize
);
```

Description

This function returns the current configuration data of the stack module specified by the corresponding module ID.

Preconditions

None

Parameters

Parameters	Description
modId	the ID that identifies the requested module
configBuff	pointer to a buffer that will receive the configuration data. If this pointer is 0, just the pNeededSize will be updated.
buffSize	size of the provided buffer
pNeededSize	pointer to an address to store the number of bytes needed to store this module configuration data. Can be NULL if not needed.

Returns

number of bytes copied to the user buffer: -1 if the module ID is invalid, 0 if the configBuff is NULL or buffSize is less than required, >0 if the call succeeded and the configuration was copied.

Remarks

None

Example

```
uint8_t      configBuffer[200];      size_t      configSize;      size_t      copiedSize;      copiedSize      =
TCPIP_STACK_ModuleGetConfig(TCPIP_MODULE_MAC_MRF24W, configBuffer, sizeof(configBuffer), &configSize);
```

5.8.17.7.4.2 TCPIP_STACK_NetAddress Function**C**

```
uint32_t TCPIP_STACK_NetAddress(
    TCPIP_NET_HANDLE netH
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to get address of.

Returns

if interface is enabled then it returns the IP address of that interface else return 0

Side Effects

None

Remarks

None

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); uint32_t ipAdd = TCPIP_STACK_NetAddress(netH);
```

5.8.17.7.4.3 TCPIP_STACK_NetAddressBcast Function**C**

```
uint32_t TCPIP_STACK_NetAddressBcast(
    TCPIP_NET_HANDLE netH
);
```


Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to get address of.

Returns

if interface is enabled then it returns the broadcast IP address mask of that interface else return 0

Side Effects

None

Remarks

None

5.8.17.7.4.4 TCPIP_STACK_NetAddressDnsPrimary Function**C**

```
uint32_t TCPIP_STACK_NetAddressDnsPrimary(  
    TCPIP_NET_HANDLE netH  
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to get the DNS address of.

Returns

the primary DNS address if succes false if not such interface or interface is down

Side Effects

None

Remarks

None

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); myIPAddress =  
TCPIP\_STACK\_NetAddressDnsPrimary(netH);
```

5.8.17.7.4.5 TCPIP_STACK_NetAddressDnsPrimarySet Function**C**

```
bool TCPIP_STACK_NetAddressDnsPrimarySet(  
    TCPIP_NET_HANDLE netH,  
    IPV4_ADDR* ipAddress  
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to set the DNS address of.
ipAddress	IP address to set

Returns

true if succes false if not such interface

Side Effects

None

Remarks

See Important Note above!

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); TCPIP\_STACK\_NetAddressDnsPrimarySet(netH,  
&myIPAddress);
```

5.8.17.7.4.6 TCPIP_STACK_NetAddressDnsSecond Function**C**

```
uint32_t TCPIP\_STACK\_NetAddressDnsSecond(  
    TCPIP\_NET\_HANDLE netH  
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to get the DNS address of.

Returns

the secondary DNS address if succes false if not such interface or interface is down

Side Effects

None

Remarks

None

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); myIPAddress =  
TCPIP\_STACK\_NetAddressDnsSecond(netH);
```

5.8.17.7.4.7 TCPIP_STACK_NetAddressDnsSecondSet Function**C**

```
bool TCPIP\_STACK\_NetAddressDnsSecondSet(  
    TCPIP\_NET\_HANDLE netH,  
    IPV4\_ADDR\* ipAddress  
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to set the secondary DNS address of.
ipAddress	IP address to set

Returns

true if succes false if not such interface

Side Effects

None

Remarks

See Important Note above!

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); TCPIP\_STACK\_NetAddressDnsSecondSet(netH,  
&myIPAddress);
```

5.8.17.7.4.8 TCPIP_STACK_NetAddressGateway Function**C**

```
uint32_t TCPIP\_STACK\_NetAddressGateway(  
    TCPIP\_NET\_HANDLE netH  
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to get address of.

Returns

if interface is enabled then it returns the gateway address of that interface else return 0

Side Effects

None

Remarks

None

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); uint32_t ipAdd =  
TCPIP\_STACK\_NetAddressGateway(netH);
```

5.8.17.7.4.9 TCPIP_STACK_NetAddressGatewaySet Function**C**

```
bool TCPIP\_STACK\_NetAddressGatewaySet(
```

```
    TCPIP_NET_HANDLE netH,  
    IPV4_ADDR* ipAddress  
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to set the gateway address of.
ipAddress	IP address to set

Returns

true if succes false if not such interface

Side Effects

None

Remarks

See Important Note above!

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); TCPIP\_STACK\_NetAddressGatewaySet(netH,  
&myIPAddress);
```

5.8.17.7.4.10 TCPIP_STACK_NetAddressMac Function

C

```
const uint8_t* TCPIP\_STACK\_NetAddressMac(  
    TCPIP_NET_HANDLE netH  
);
```

Preconditions

None

Parameters

Parameters	Description
netH	Interface handle to get the address of.

Returns

if interface is enabled then it returns a constant pointer to the MAC address of that interface else return 0

Remarks

None

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); const TCPIP\_MAC\_ADDR\* pAdd =  
TCPIP\_STACK\_NetAddressMac(netH);
```

5.8.17.7.4.11 TCPIP_STACK_NetAddressMacSet Function

C

```
bool TCPIP\_STACK\_NetAddressMacSet(  
    TCPIP_NET_HANDLE netH,
```

```
    const TCPIP_MAC_ADDR* pAddr  
);
```

Preconditions

None

Parameters

Parameters	Description
netH	Interface handle to set the address of.

Returns

if interface is enabled then it returns a constant pointer to the MAC address of that interface else return 0

Remarks

See Important Note above!

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); TCPIP\_STACK\_NetAddressMacSet(netH,  
&myMacAddress);
```

5.8.17.7.4.12 TCPIP_STACK_NetAddressSet Function**C**

```
bool TCPIP_STACK_NetAddressSet(  
    TCPIP_NET_HANDLE netH,  
    IPV4_ADDR* ipAddress,  
    IPV4_ADDR* mask,  
    bool setDefault  
);
```

Preconditions

TCPIP stack should have been initialized by [TCPIP_STACK_Initialize\(\)](#)

Parameters

Parameters	Description
netH	Interface handle to set address of.
ipAddress	IP address to set (could be NULL to set only the mask)
mask	corresponding network mask to set (could be NULL to set only the IP address)
setDefault	if true, the interface default address/mask is also set

Returns

true if succes false if not such interface

Side Effects

None

Remarks

See Important Note above!

This function sets the associated network IP address and/or mask. If you're changing the network then it's preferred that you set both these values simultaneously to avoid having the stack running with a mismatch between its IP address and mask.

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); TCPIP\_STACK\_NetAddressSet(netH,  
&myIPAddress);
```

5.8.17.7.4.13 TCPIP_STACK_NetIPv6AddressGet Function

C

```
IPV6_ADDR_HANDLE TCPIP_STACK_NetIPv6AddressGet(
    TCPIP_NET_HANDLE netH,
    IPV6_ADDR_TYPE addType,
    IPV6_ADDR_STRUCT* pAddStruct,
    IPV6_ADDR_HANDLE addHandle
);
```

Description

This function allows the listing of the IPv6 addresses associated with an interface.

Preconditions

None

Parameters

Parameters	Description
netH	handle of the interface to retrieve the addresses for
addType	type of address to request: IPV6_ADDR_TYPE_UNICAST and IPV6_ADDR_TYPE_MULTICAST supported for now
pAddStruct	structure provided by the user that will be filled with corresponding IPV6_ADDR_STRUCT data
addHandle	an address handle that allows iteration across multiple IPv6 addresses.
On the first call	has to be 0; it will begin the listing of the IPv6 addresses
On subsequent calls	has to be a handle previously returned by a call to this function

Returns

- a non NULL IPV6_ADDR_HANDLE if an valid IPv6 address was found and the pAddStruct structure was filled with data
- 0 if no other IPv6 exists or if the supplied IPV6_ADDR_HANDLE is invalid

Remarks

None

Example

```
IPV6_ADDR_STRUCT    currAddr;    IPV6_ADDR_HANDLE    currHandle;    TCPIP_NET_HANDLE    hNet    =
TCPIP_STACK_NetHandleGet("PIC32INT"); char ipv6AddBuff[44];

currHandle = 0; do { currHandle = TCPIP_STACK_NetIPv6AddressGet(netH, IPV6_ADDR_TYPE_UNICAST, &currAddr,
currHandle); if(currHandle) { // have a valid address; display it TCPIP_HELPER_IPv6AddressToString(&currAddr.address,
ipv6AddBuff, sizeof(ipv6AddBuff)); } }while(currHandle != 0);
```

5.8.17.7.5 Network Structure Storage Functions

5.8.17.7.5.1 TCPIP_STACK_NetConfigGet Function

C

```
size_t TCPIP_STACK_NetConfigGet(
    TCPIP_NET_HANDLE netH,
    void* configStoreBuff,
    size_t configStoreSize,
    size_t* pNeededSize
);
```

Description

This function dumps the current configuration data of the network interface specified by the corresponding network handle into the supplied buffer.

Preconditions

None

Parameters

Parameters	Description
netH	the handle that identifies the requested interface
configStoreBuff	pointer to a buffer that will receive the current configuration data. All the data that's needed to restore a TCPIP_NETWORK_CONFIG structure is stored in this buffer. Can be NULL if only the storage size is needed.
configStoreSize	size of the supplied buffer
pNeededSize	pointer to store the size needed for storage; Can be NULL if not needed

Returns

-1 if the interface is invalid or the stack is not initialized 0 if no data is copied (no supplied buffer or buffer too small) >0 for success, indicating the amount of data copied.

Remarks

The function is a helper for retrieving the network configuration data. Its companion function, [TCPIP_STACK_NetConfigSet](#), restores the TCPIP_NETWORK_CONFIG from the dump buffer.

Currently the data is saved in plain binary format into the supplied buffer. However the application must not make use of this assumption as it may change in the future releases (some compression scheme may be implemented).

Example

```
uint8_t currConfig[100]; size_t neededSize, result; TCPIP\_NET\_HANDLE hNet = TCPIP\_STACK\_NetHandleGet("PIC32INT");
result = TCPIP_STACK_NetConfigGet(hNet, currConfig, sizeof(currConfig), &neededSize); if(result > 0) { // store the currConfig
to some external storage }
```

5.8.17.7.5.2 TCPIP_STACK_NetConfigSet Function**C**

```
TCPIP_NETWORK_CONFIG* TCPIP\_STACK\_NetConfigSet(
    void* configStoreBuff,
    void* netConfigBuff,
    size_t buffSize,
    size_t* pNeededSize
);
```

Description

This function restores data from a previously dump buffer and updates the supplied interface configuration. All the data is recovered and constructed into the netConfigBuff (supposing this buffer is large enough). If this operation succeeded, the netConfigBuff can be safely cast to a (TCPIP_NETWORK_CONFIG*).

The structure of the netConfigBuff is as follows:

- a TCPIP_NETWORK_CONFIG sturcture is created at the very beginning of the buffer
- all the needed fields that are part of the TCPIP_NETWORK_CONFIG will be placed in the buffer itself.

Preconditions

None

Parameters

Parameters	Description
configStoreBuff	pointer to a buffer that received configuration data from a TCPIP_STACK_NetConfigGet call.
netConfigBuff	pointer to a buffer that will receive the TCPIP_NETWORK_CONFIG data
buffSize	size of the supplied netConfigBuff buffer
pNeededSize	pointer to store the size needed for storage; Can be NULL if not needed

Returns

a valid TCPIP_NETWORK_CONFIG pointer (netConfigBuff) if the netConfigBuff is successfully updated 0 - if the netConfigBuff is not supplied or not big enough

If supplied, the pNeededSize will be updated with the actual size that's needed for the netConfigBuff

Remarks

The function is a helper for being able to restore the configuration data. Its companion function, [TCPIP_STACK_NetConfigGet](#), saves the TCPIP_NETWORK_CONFIG to a dump buffer.

Example

```
uint8_t currConfig[100]; uint8_t restoreBuff[100]; size_t neededSize, result; TCPIP\_NET\_HANDLE hNet =
TCPIP\_STACK\_NetHandleGet("PIC32INT"); result = TCPIP\_STACK\_NetConfigGet(hNet, currConfig, sizeof(currConfig),
&neededSize); if(result > 0) { // store the currConfig buffer to some external storage (neededSize bytes needed)

// later on restore the configuration TCPIP_NETWORK_CONFIG* netConfig; // extract the network configuration from the
previously saved buffer netConfig = TCPIP_STACK_NetConfigSet(currConfig, restoreBuff, sizeof(restoreBuff), neededSize);
if(netConfig) { // use this netConfig to initialize a network interface TCPIP\_STACK\_NetUp(hNet, netConfig); }
```

5.8.17.7.5.3 TCPIP_STACK_NetMACId Function**C**

```
TCPIP_STACK_MODULE TCPIP\_STACK\_NetMACId(
    TCPIP\_NET\_HANDLE netH
);
```

Description

This function returns the module ID of the MAC that's attached to the specified network interface.

Preconditions

None

Parameters

Parameters	Description
netH	handle of the interface to use

Returns

a TCPIP_STACK_MODULE ID that belongs to the MAC of that network interface.

Remarks

None

Example

```
TCPIP\_NET\_HANDLE netH = TCPIP\_STACK\_NetHandleGet("PIC32INT"); TCPIP\_STACK\_MODULE modId =
TCPIP\_STACK\_NetMACId(netH); if(modId == TCPIP\_MODULE\_MAC\_PIC32INT) { // an internal PIC32 MAC attached to this
```


interface }

5.8.17.7.6 Data Types and Constants

5.8.17.7.6.1 TCPIP_LOCAL_MASK_TYPE Enumeration

C

```
typedef enum {
    TCPIP_LOCAL_MASK_ZERO,
    TCPIP_LOCAL_MASK_ONE,
    TCPIP_LOCAL_MASK_NET,
    TCPIP_LOCAL_MASK_SET
} TCPIP_LOCAL_MASK_TYPE;
```

Description

types of masks used in the detection of a local/nonlocal network

Members

Members	Description
TCPIP_LOCAL_MASK_ZERO	use an all zero network mask
TCPIP_LOCAL_MASK_ONE	use an all ones network mask
TCPIP_LOCAL_MASK_NET	use the current network mask
TCPIP_LOCAL_MASK_SET	use the set value for the network mask there are operations available that set the local/nonlocal detection network mask

5.8.17.7.6.2 TCPIP_NET_HANDLE Type

C

```
typedef const void* TCPIP_NET_HANDLE;
```

Description

a network interface handle

5.8.17.8 Files

Files

Name	Description
tcpip_manager.h	

Description

5.8.17.8.1 tcpip_manager.h

Microchip TCP/IP Stack Definitions

Enumerations

	Name	Description
	TCPIP_LOCAL_MASK_TYPE	types of masks used in the detection of a local/nonlocal network

Functions

	Name	Description
≡	TCPIP_STACK_Deinitialize	This function performs the de-initialization of the TCPIP stack
≡	TCPIP_STACK_IndexToNet	
≡	TCPIP_STACK_Initialize	The function initializes the stack. If an error occurs, the SYS_ERROR() is called and the function de-initialize itself and will return false.
≡	TCPIP_STACK_ModuleGetConfig	This function returns the current configuration data of the stack module specified by the corresponding module ID.
≡	TCPIP_STACK_NetAddress	
≡	TCPIP_STACK_NetAddressBcast	
≡	TCPIP_STACK_NetAddressDnsPrimary	
≡	TCPIP_STACK_NetAddressDnsPrimarySet	
≡	TCPIP_STACK_NetAddressDnsSecond	
≡	TCPIP_STACK_NetAddressDnsSecondSet	
≡	TCPIP_STACK_NetAddressGateway	
≡	TCPIP_STACK_NetAddressGatewaySet	
≡	TCPIP_STACK_NetAddressMac	
≡	TCPIP_STACK_NetAddressMacSet	
≡	TCPIP_STACK_NetAddressSet	
≡	TCPIP_STACK_NetBIOSName	
≡	TCPIP_STACK_NetBiosNameSet	
≡	TCPIP_STACK_NetConfigGet	This function dumps the current configuration data of the network interface specified by the corresponding network handle into the supplied buffer.

5.8.18 NBNS TCP/IP Stack Library

5.8.18.1 Introduction

NetBIOS (NBNS) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the NBNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet. Practically, this allows the assignment of human-name hostnames to access boards on the same subnet. For example, in the "TCP/IP Demo App" demonstration project, the demo board is programmed with the human name 'mchpboard' so it can be accessed directly instead of with its IP address.

5.8.18.2 Release Notes

MPLAB Harmony Version: v0.70b **NBNS TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.18.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.18.4 Using the Library

This topic describes the basic architecture of the NBNS TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [nbns.h](#)

The interface to the NBNS TCPIP Stack library is defined in the "[nbns.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the NBNS TCP/IP Stack library should include "tcpip.h".

Library File:

The NBNS TCPIP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.18.4.1 Abstraction Model

This library provides the API of the NBNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description**NBNS Software Abstraction Block Diagram**

Note: This image was not available at time of release and will be added in a future release of MPLAB Harmony.

5.8.18.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the NBNS module.

Library Interface Section	Description
Data Types and Constants	This section provides various definitions describing This API

5.8.18.5 Configuring the Library

Macros

	Name	Description
	NBNS_PORT	NetBIOS Name Service port

Description

The configuration of the NBNS TCP/IP Stack is based on the file tcpip_config.h

This header file contains the configuration selection for the NBNS TCP/IP Stack. Based on the selections made, the NBNS TCP/IP Stack will support or not support selected features. These configuration settings will apply to all instances of the NBNS TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.18.5.1 NBNS_PORT Macro

C

```
#define NBNS_PORT (137u)
```

Description

NetBIOS Name Service port

5.8.18.6 Building the Library

This section list the files that are available in the \src of the NBNS driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.18.7 Library Interface

Data Types and Constants

	Name	Description
	NBNS_MODULE_CONFIG	Provides a placeholder for NBNS configuration upgrades.

Description

5.8.18.7.1 Data Types and Constants

5.8.18.7.1.1 NBNS_MODULE_CONFIG Structure

C

```
typedef struct {  
} NBNS_MODULE_CONFIG;
```

Description

NBNS_MODULE_CONFIG Structure Typedef

Provides a placeholder for NBNS configuration upgrades.

Remarks

None

5.8.18.8 Files

Files

Name	Description
nbns.h	
nbns_config.h	NBNS configuration file

Description

5.8.18.8.1 nbns.h

NetBIOS Name Service (NBNS) Server public API

-Responds to NBNS name requests to allow human name assignment to the board

Structures

	Name	Description
	NBNS_MODULE_CONFIG	Provides a placeholder for NBNS configuration upgrades.

5.8.18.8.2 nbns_config.h

NetBIOS Name Service (NBNS) Configuration file

This file contains the NBNS module configuration options

Macros

	Name	Description
	NBNS_PORT	NetBIOS Name Service port

5.8.19 NDP TCP/IP Stack Library

5.8.19.1 Introduction

NDP TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the NDP (Neighbor Discovery Protocol) TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

This document provides NDP(Neighbor Discovery Protocol) for IP version 6.IPv6 nodes on same link use neighbor discovery to discover each other's presence. RFC - 4861.

It is responsible for

1. Address Auto configuration of nodes
2. Discovery of other nodes in the link(It can be a Router Discovery or Neighbor Discovery)
3. Determining the link layer address of other nodes
4. Duplicate address detection (DAD)
5. Finding available routers and Domain Name System (DNS) servers
6. Address Prefix discovery
7. Parameter Discovery (Such as Link MTU or Hop limit)

Comparing with IPv4

- NDP is a substitute of ARP (Address Resolution protocol). This new mechanism uses a mix of ICMPv6 and multicast addresses to discover the IPv6 node on same link.
- NDP includes Neighbor Unreachability Detection (NUD) , thus improving the robustness of packet delivery.
- Unlike IPv4 broadcast addresses, IPv6 address resolution multicasts are spread over 4 billion (2^{32}) multicast addresses, greatly reducing address resolution-related interrupts on nodes other than the target. Moreover, non-IPv6 machines should not be interrupted at all.
- Neighbor Discovery defines five different ICMPv6 packet types.

Five different ICMPv6 Packet types -

1. **Router Solicitation** - Hosts inquire with Router Solicitation message to locate routers on the attached link.
2. **Router Advertisement** - Router advertise their presence periodically to all the nodes or in response to the Router Solicitation

message.

3. **Neighbor Solicitation** - Neighbor solicitations are used by nodes to determine the Link Layer address of a neighbor, or to verify that a neighbor is still reachable via a cached Link Layer address.
4. **Neighbor Advertisement** - Neighbor advertisements are used by nodes to respond to a Neighbor Solicitation message
5. **Redirect** - Routers may inform hosts of a better first hop router for a destination.

5.8.19.2 Release Notes

MPLAB Harmony Version: v0.70b **NDP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.19.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.19.4 Using the Library

This topic describes the basic architecture of the NDP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [ndp.h](#)

The interface to the NDP TCP/IP Stack Library is defined in the "[ndp.h](#)" header file. Any C language source (.c) file that uses the NDP TCP/IP library should include "IPv6.h", [icmpv6.h](#) and [ndp.h](#)".

Library File:

The NDP TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack Library interacts with the framework.

5.8.19.4.1 Abstraction Model

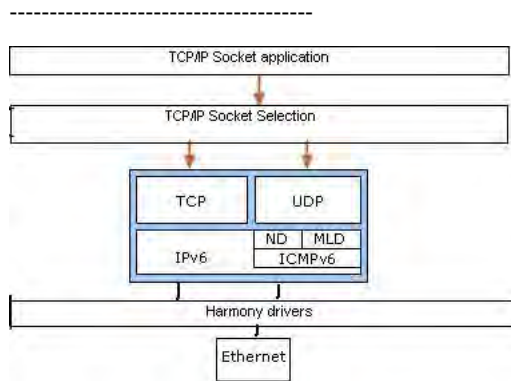
This library provides the API of the NDP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

NDP with IPv6 Software Abstraction Block Diagram

This module provides software abstraction of the IPv6 module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

IPv6 Block Diagram



Link Layer Neighbor Discovery

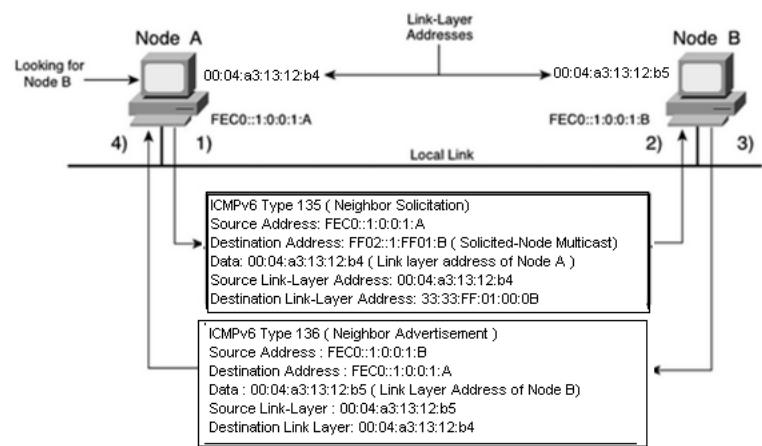
1) Using the address FEC0::1:0:0:1 :A, node A wants to deliver packets to destination node B using the IPv6 address FEC0::1 :0:0:1 :B on the same local link. However, node A does not know node B's link-layer address. Node A sends an ICMPv6 Type

135 message (neighbor solicitation) on the local link using its site-local address FEC0::1:0:0:1:A as the IPv6 source address, the solicited-node multicast address FF02::1 :FF01:B corresponding to the target address FEC0::1 :0:0:1 :B as the destination IPv6 address, and the source link-layer address 00:04:a3:13:12:b4 of the sender, node A, as data of the ICMPv6 message. The source link-layer address of this frame is the link-layer address 00:04:a3:13:12:b4 of node A. The destination link-layer address 33:33:FF:01 :00:0B of this frame uses multicast mapping of the destination IPv6 address FF02::1 :FF01:B.

2) Node B, which is listening to the local link for multicast addresses, intercepts the neighbor solicitation message because the destination IPv6 address FF02::1:FF01:B represents the solicited-node multicast address corresponding to its IPv6 address FEC0::1:0:0:1:B.

3) Node B replies by sending a neighbor advertisement message using its site-local address FEC0::1 :0:0:1 :B as the IPv6 source address and the site-local address FEC0::1 :0:0:1 :A as the destination IPv6 address. It also includes its link-layer address 00:04:a3:13:12:b5 in the ICMPv6 message. After receiving neighbor solicitation and neighbor advertisement messages, node A and node B know each other's link-layer addresses.

Learned link-layer addresses are kept in a neighbor discovery table (neighbor cache). Therefore, the nodes can communicate on the local link. The neighbor solicitation message is also used by nodes to verify the reachability of neighbor nodes in the neighbor discovery table (neighbor cache). However, the unicast addresses of the neighbor nodes are used as destination IPv6 addresses in ICMPv6 messages instead of solicited-node multicast addresses in this situation. It is possible for a node that changes its link-layer address to inform all other neighbor nodes on the local link by sending a neighbor advertisement message using the all-nodes multicast address FF02::1 . The neighbor discovery table of the nodes on the local link is updated with the new linklayer address.



5.8.19.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the IPv6 , ICMPv6 and NDP module.

Library Interface Section	Description
Reachability Functions	Reachability routine functionality

5.8.19.5 Configuring the Library

Macros

	Name	Description
	DELAY_FIRST_PROBE_TIME	5 s
	IPV6_MTU_INCREASE_TIMEOUT	600 seconds
	MAX_ANYCAST_DELAY_TIME	1 s
	MAX_MULTICAST_SOLICIT	3 transmissions
	MAX_NEIGHBOR_ADVERTISEMENT	3 transmissions
	MAX_RTR_SOLICITATION_DELAY	1 s
	MAX_RTR_SOLICITATIONS	3 transmissions
	MAX_UNICAST_SOLICIT	3 transmissions
	NDP_TASK_TIMER_RATE	default 32 ms
	NDP_VALID_LIFETIME_TWO_HOURS	Sets the lifetime to 2 hours
	REACHABLE_TIME	30 s
	RETRANS_TIMER	1 s
	RTR_SOLICITATION_INTERVAL	4 s

Description

The configuration of the TCPIP Stack IPv6 is based on the file [ndp_config.h](#)

This header file contains the configuration selection for the TCPIP Stack IP. Based on the selections made, the TCPIP Stack IP will support or not support selected features. These configuration settings will apply to all instances of the TCPIP Stack IP.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a

successful build.

5.8.19.5.1 DELAY_FIRST_PROBE_TIME Macro

C

```
#define DELAY_FIRST_PROBE_TIME 5u           // 5 s
```

Description

5 s

5.8.19.5.2 IPV6_MTU_INCREASE_TIMEOUT Macro

C

```
#define IPV6_MTU_INCREASE_TIMEOUT 600u1     // 600 seconds
```

Description

600 seconds

5.8.19.5.3 MAX_ANYCAST_DELAY_TIME Macro

C

```
#define MAX_ANYCAST_DELAY_TIME 1u           // 1 s
```

Description

1 s

5.8.19.5.4 MAX_MULTICAST_SOLICIT Macro

C

```
#define MAX_MULTICAST_SOLICIT 3u            // 3 transmissions
```

Description

3 transmissions

5.8.19.5.5 MAX_NEIGHBOR_ADVERTISEMENT Macro

C

```
#define MAX_NEIGHBOR_ADVERTISEMENT 3u      // 3 transmissions
```

Description

3 transmissions

5.8.19.5.6 MAX_RTR_SOLICITATION_DELAY Macro

C

```
#define MAX_RTR_SOLICITATION_DELAY 1u       // 1 s
```

Description

1 s

5.8.19.5.7 MAX_RTR_SOLICITATIONS Macro

C

```
#define MAX_RTR_SOLICITATIONS 3u           // 3 transmissions
```

Description

3 transmissions

5.8.19.5.8 MAX_UNICAST_SOLICIT Macro

C

```
#define MAX_UNICAST_SOLICIT 3u           // 3 transmissions
```

Description

3 transmissions

5.8.19.5.9 NDP_TASK_TIMER_RATE Macro

C

```
#define NDP_TASK_TIMER_RATE (32)         // default 32 ms
```

Description

default 32 ms

5.8.19.5.10 NDP_VALID_LIFETIME_TWO_HOURS Macro

C

```
#define NDP_VALID_LIFETIME_TWO_HOURS (60 * 60 * 2)
```

Description

Sets the lifetime to 2 hours

5.8.19.5.11 REACHABLE_TIME Macro

C

```
#define REACHABLE_TIME 30u              // 30 s
```

Description

30 s

5.8.19.5.12 RETRANS_TIMER Macro

C

```
#define RETRANS_TIMER 1u                // 1 s
```

Description

1 s

5.8.19.5.13 RTR_SOLICITATION_INTERVAL Macro

```
C
#define RTR_SOLICITATION_INTERVAL 4u           // 4 s
```

Description

4 s

5.8.19.6 Building the Library

This section list the files that are available in the \src of the IPv6 and ICMPv6 driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- ipv6.c
- icmpv6.c
- ndp.c

TCPIP_STACK_USE_IPV6 is enabled from tcpip_config.h .

5.8.19.7 Library Interface

Reachability Functions

	Name	Description
	TCPIP_NDP_NborReachConfirm	Confirms that a neighbor is reachable.

Description

This section describes the Application Programming Interface (API) functions of the NDP TCP/IP Stack Library.

Refer to each section for a detailed description.

5.8.19.7.1 Reachability Functions

5.8.19.7.1.1 TCPIP_NDP_NborReachConfirm Function

```
C
void TCPIP_NDP_NborReachConfirm(
    TCPIP_NET_HANDLE netH,
    IPV6_ADDR * address
);
```

Description

This function is used by upper-layer protocols to indicate that round-trip communications were confirmed with a neighboring node.

Preconditions

None

Parameters

Parameters	Description
pNetIf	The interface the neighbor is on.

address	The address of the neighbor.

Returns

None.

Remarks

None

5.8.19.8 Files

Files

Name	Description
ndp.h	IPv6 Internet Communication Message Protocol
ndp_config.h	NDP configuration file

Description

5.8.19.8.1 ndp.h

Neighbor Discovery Protocol (NDP) in IPv6 is the substitute of as ARP(Which is used in IPv4 for address resolve).NDP is used discover link local addresses of the IPv6 nodes present in the local link using a mix of ICMPv6 messages and multicast addresses, Stateless auto-configuration and router redirection.

Functions

	Name	Description
	TCPIP_NDP_NborReachConfirm	Confirms that a neighbor is reachable.

File Name

ndp.h

Company

Microchip Technology Incorporated

5.8.19.8.2 ndp_config.h

Neighbor Discovery Protocol (NDP) Configuration file

This file contains the NDP module configuration options

Macros

	Name	Description
	DELAY_FIRST_PROBE_TIME	5 s
	IPV6_MTU_INCREASE_TIMEOUT	600 seconds
	MAX_ANYCAST_DELAY_TIME	1 s
	MAX_MULTICAST_SOLICIT	3 transmissions
	MAX_NEIGHBOR_ADVERTISEMENT	3 transmissions
	MAX_RTR_SOLICITATION_DELAY	1 s
	MAX_RTR_SOLICITATIONS	3 transmissions

	MAX_UNICAST_SOLICIT	3 transmissions
	NDP_TASK_TIMER_RATE	default 32 ms
	NDP_VALID_LIFETIME_TWO_HOURS	Sets the lifetime to 2 hours
	REACHABLE_TIME	30 s
	RETRANS_TIMER	1 s
	RTR_SOLICITATION_INTERVAL	4 s

5.8.20 Reboot TCP/IP Stack Library

5.8.20.1 Introduction

Reboot TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the Reboot TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The Reboot module will allow a user to remotely reboot the PIC microcontroller that is running the TCP/IP stack. This feature is primarily used for bootloader applications, which must reset the microcontroller to enter the bootloader code section. This module will execute a task that listens on a specified UDP port for a packet, and then reboots if it receives one.

5.8.20.2 Release Notes

Harmony Version: v0.70b **Reboot TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.20.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR

ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.20.4 Using the Library

This topic describes the basic architecture of the Reboot TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [tcpip_reboot_config.h](#)

The interface to the Reboot TCP/IP Stack library is defined in the "[tcpip_reboot_config.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the Reboot TCP/IP Stack library should include "tcpip.h".

Library File:

The Reboot TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.20.4.1 Abstraction Model

This library provides the API of the Reboot TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Reboot Software Abstraction Block Diagram

Note: This image was not available at time of release and will be added in a future release of MPLAB Harmony.

5.8.20.5 Configuring the Library

Macros

	Name	Description
	REBOOT_PORT	UDP TFTP port
	REBOOT_SAME_SUBNET_ONLY	For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define REBOOT_SAME_SUBNET_ONLY to enable this access restriction.

Description

The configuration of the Reboot TCP/IP Stack Library is based on the file `tcpip_config.h`

This header file contains the configuration selection for the Reboot TCP/IP Stack Library. Based on the selections made, the Reboot TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the Reboot TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a

successful build. Refer to the [Applications Overview](#) section for more details.

5.8.20.5.1 REBOOT_PORT Macro

C

```
#define REBOOT_PORT 69
```

Description

UDP TFTP port

5.8.20.5.2 REBOOT_SAME_SUBNET_ONLY Macro

C

```
#define REBOOT_SAME_SUBNET_ONLY
```

Description

For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define REBOOT_SAME_SUBNET_ONLY to enable this access restriction.

5.8.20.6 Building the Library

This section list the files that are available in the \src of the Reboot driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.20.7 Library Interface

This section describes the Application Programming Interface (API) functions of the Reboot TCP/IP Stack, if applicable.

5.8.20.8 Files

Files

Name	Description
tcpip_reboot_config.h	Configuration file

Description

5.8.20.8.1 tcpip_reboot_config.h

Reboot Configuration file

This file contains the Reboot module configuration options

Macros

	Name	Description
	REBOOT_PORT	UDP TFTP port

	REBOOT_SAME_SUBNET_ONLY	For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define REBOOT_SAME_SUBNET_ONLY to enable this access restriction.

5.8.21 SMTP TCP/IP Stack Library

5.8.21.1 Introduction

Simple Mail Transfer Protocol (SMTP) TCP/IP Stack Library for Microchip Microcontrollers

This library provides the API of the SMTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The SMTP client module in the TCP/IP Stack lets applications **send** e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

Using the SMTP client requires access to a local mail server (such as mail.yourdomain.com) for reliable operation. Your MPLAB Harmony or network administrator can provide the correct address, but end-user applications will need an interface to provide this data.

5.8.21.2 Release Notes

MPLAB Harmony Version: v0.70b **SMTP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.21.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.21.4 Using the Library

This topic describes the basic architecture of the SMTP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [smtp.h](#)

The interface to the SMTP TCP/IP Stack library is defined in the "[smtp.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the SMTP TCP/IP Stack library should include "tcpip.h".

Library File:

The SMTP TCPIP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.21.4.1 Abstraction Model

This library provides the API of the SMTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SMTP Software Abstraction Block Diagram

Note: This image was not available at time of release and will be added in a future release of MPLAB Harmony.

5.8.21.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the SMTP module.

Library Interface Section	Description
Functions	Routines to configure this module
Data Types and Constants	This section provides various definitions describing this API

5.8.21.4.3 SMTP Client Examples

Module

SMTP Client

Description

The following two examples demonstrate the use of the SMTP client in different scenarios. The first, and simpler example, sends a short message whose contents are all located in RAM at once.

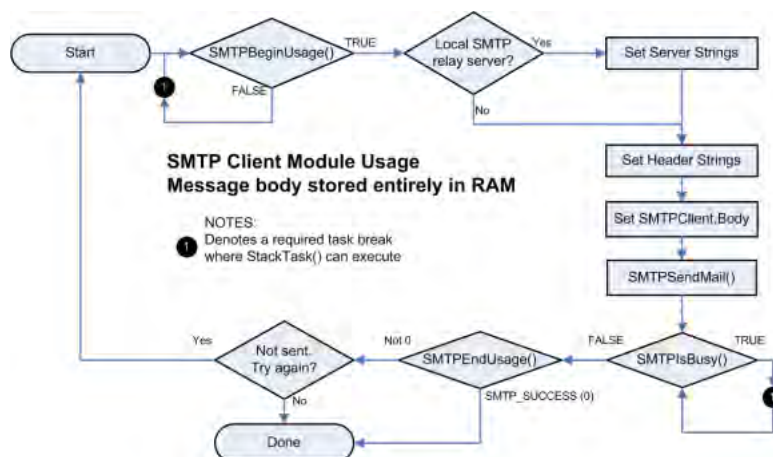
The second example is more involved and demonstrates generating a message on the fly in the case where the entire message cannot fit into RAM at once. In this case, the message

5.8.21.4.3.1 SMTP Client Short Message Example

The SMTP client API is simplified when messages can be buffered entirely in RAM. The SMTPDemo (see page 99) example provided in MainDemo.c sends a brief e-mail message indicating the current status of the board's buttons. This document will walk through that example.

Make sure `STACK_USE_SMTP_CLIENT` is uncommented in `TCPIPConfig.h` before continuing.

The following diagram provides an overview of the process:



First, call `SMTPBeginUsage` (see page 304) to verify that the SMTP client is available and to begin a new message. If `FALSE` is returned, the SMTP client is busy and the application must return to the main loop to allow `StackTask` to execute again.

Next, set the local relay server to use as `SMTPClient.Server`. If the local relay server requires a user name and password, set `SMTPClient.Username` and `SMTPClient.Password` to the appropriate credentials.

If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to spam prevention measures put in place by most ISPs and network administrators.

Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need to add. Finally, set `SMTPClient.Body` to the message to be sent.

At this point, verify that `SMTPClient.ROMPointers` is correctly configured for any strings that are stored in program memory. Once the message is ready to send, call `SMTPSendMail` (see page 310) to instruct the SMTP client to begin transmission.

The application must now call `SMTPIsBusy` (see page 306) until it returns `FALSE`. Each time `TRUE` is returned, return to the main loop and wait for `StackTask` to execute again. This allows the SMTP server to continue its work in a cooperative multitasking manner. Once `FALSE` is returned, call `SMTPEndUsage` (see page 305) to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in `MainDemo.c` needs minor modifications to use your e-mail address. The `Server` and `To` fields must be set in `SMTPDemo` (see page 99) in order for the message to be properly delivered. Once this is done, holding down `BUTTON2` and `BUTTON3` simultaneously (the left-most two buttons) will begin sending the message. `LED1` will light as the message is being

processed, and will extinguish when the SMTP state machine completes. If the transmission was successful LED2 will light, otherwise it will remain dark.

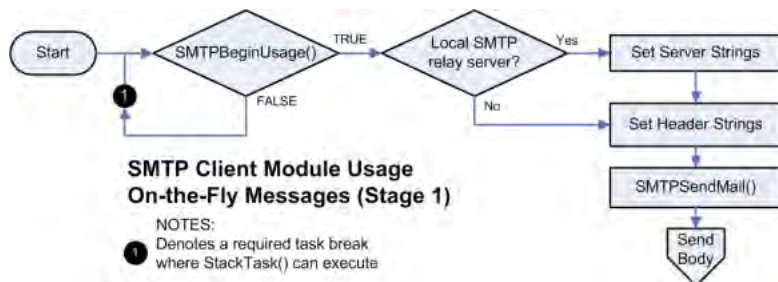
5.8.21.4.3.2 SMTP Client Long Message Example

The SMTP client API is capable of sending messages that do not fit entirely in RAM. To do so, the application must manage its output state and only write as many bytes as are available in the buffer at a time. The second SMTPDemo (see page 99) example provided in MainDemo.c sends a message that is a dump of all contents of the PIC's RAM. This example is currently commented out. Comment out the previous Short Message Example and uncomment the Long Message Example. This document will walk through sending a longer message.

Make sure `STACK_USE_SMTP_CLIENT` is uncommented in `TCIPConfig.h` before continuing.

Sending longer messages is divided into three stages. The first stage configures the SMTP client to **send** the message. The second stage sends the message in small chunks as buffer space is available. The final stage finishes the transmission and determines whether or not the message was successful.

The following diagram illustrates the first stage:



The first stage is largely similar to the first few steps in sending a short message. First, call `SMTPBeginUsage` (see page 304) to verify that the SMTP client is available and to begin a new message. If `FALSE` is returned, the SMTP client is busy and the application must return to the main loop to allow `StackTask` to execute again.

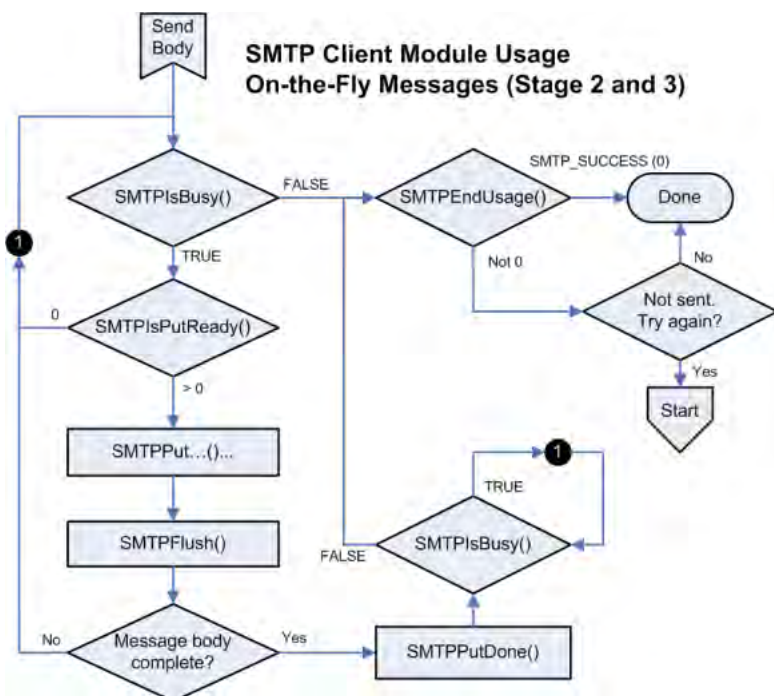
Next, set the local relay server to use as `SMTPClient.Server`. If the local relay server requires a user name and password, set `SMTPClient.Username` and `SMTPClient.Password` to the appropriate credentials.

If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to spam prevention measures put in place by most ISPs and network administrators.

Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need to add.

The next portion of the process differs. Ensure that `SMTPClient.Body` remains set to its default (`NULL`). At this point, call `SMTPSendMail` (see page 310) to open a connection to the remote server and transmit the headers. The application is now ready to proceed to the second stage and **send** the message body.

The following diagram provides an overview of stage two and three:



Upon entering stage two, the application should call `SMTPIsBusy` (see page 306) to verify that the connection to the remote server is active and has not been lost. If the call succeeds, call `SMTPIsPutReady` (see page 306) to determine how many bytes are available in the TX buffer. If no bytes are available, return to the main loop so that `StackTask` can transmit the data to the remote node and free up the buffer.

If space is available, any combination of the `SMTPPut` (see page 307), `SMTPPutArray` (see page 307), `SMTPPutROMArray` (see page 308), `SMTPPutString` (see page 309), and `SMTPPutROMString` (see page 309) functions may be called to transmit the message. These functions return the number of bytes successfully written. Use this value, along with the value originally returned from `SMTPIsPutReady` (see page 306) to track how much free space remains in the TX buffer. Once the buffer is depleted, call `SMTPFlush` (see page 305) to force the data written to be sent.

The SMTP client module can [accept](#) (see page 170) as much data as the TCP TX FIFO can hold. This is determined by the [socket](#) initializer for `TCP_PURPOSE_DEFAULT` type sockets in `TCPIPConfig.h`, which defaults to 200 bytes.

If the TX buffer is exhausted before the message is complete, return to the main loop so that `StackTask` may transmit the data to the remote node and free up the buffer. Upon return, go to the beginning of the second stage to transmit the next portion of the message.

Once the message is complete, the application will move to the third stage. Call `SMTPPutDone` (see page 308) to inform the SMTP client that no more data remains. Then call `SMTPIsBusy` (see page 306) repeatedly. Each time TRUE is returned, return to the main loop and wait for `StackTask` to execute again. Once FALSE is returned, the message transmission has completed and the application must call `SMTPEndUsage` (see page 305) to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in `MainDemo.c` needs minor modifications to use your e-mail address. Set the Server and To fields in `SMTPDemo` (see page 99), and ensure that these fields are being properly assigned to `SMTPClient` (see page 305) struct. The demo works exactly the same way as the previous one, with `BUTTON2` and `BUTTON3` held down simultaneously (the left-most two buttons) kicking off the state machine. LED1 will light as the message is being processed, and will extinguish when the SMTP state machine completes. If the transmission was successful LED2 will light, otherwise it will remain dark.

5.8.21.5 Configuring the Library

Macros

	Name	Description
	MAX_SMTP_CONNECTIONS	Maximum number of SMTP connections allowed
	SMTP_PORT	Default port to use when unspecified
	SMTP_SERVER_REPLY_TIMEOUT	How long to wait before assuming the connection has been dropped (default 8 seconds)

Description

The configuration of the SMTP TCP/IP Stack Library is based on the file `tcip_config.h`

This header file contains the configuration selection for the SMTP TCP/IP Stack Library. Based on the selections made, the SMTP TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the SMTP TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.21.5.1 MAX_SMTP_CONNECTIONS Macro

C

```
#define MAX_SMTP_CONNECTIONS (1)
```

Description

Maximum number of SMTP connections allowed

5.8.21.5.2 SMTP_PORT Macro

C

```
#define SMTP_PORT 25
```

Description

Default port to use when unspecified

5.8.21.5.3 SMTP_SERVER_REPLY_TIMEOUT Macro

C

```
#define SMTP_SERVER_REPLY_TIMEOUT (8uL)
```

Description

How long to wait before assuming the connection has been dropped (default 8 seconds)

5.8.21.6 Building the Library

This section list the files that are available in the `\src` of the SMTP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.21.7 Library Interface

Functions

	Name	Description
	TCPIP_SMTP_ArrayPut	Writes a series of bytes to the SMTP client.
	TCPIP_SMTP_Flush	Flushes the SMTP socket and forces all data to be sent.
	TCPIP_SMTP_IsBusy	Determines if the SMTP client is busy.
	TCPIP_SMTP_IsPutReady	Determines how much data can be written to the SMTP client.
	TCPIP_SMTP_MailSend	Initializes the message sending process.
	TCPIP_SMTP_Put	Writes a single byte to the SMTP client.
	TCPIP_SMTP_PutIsDone	Indicates that the on-the-fly message is complete.
	TCPIP_SMTP_StringPut	Writes a string to the SMTP client.
	TCPIP_SMTP_UsageBegin	Requests control of the SMTP client module.
	TCPIP_SMTP_UsageEnd	Releases control of the SMTP client module.

Data Types and Functions

	Name	Description
	SMTP_CONNECT_ERROR	Connection to SMTP server failed
	SMTP_RESOLVE_ERROR	DNS lookup for SMTP server failed
	SMTP_SUCCESS	Message was successfully sent
	SMTP_CLIENT_MODULE_CONFIG	This is type SMTP_CLIENT_MODULE_CONFIG.
	SMTP_POINTERS	Configures the SMTP client to send a message

Description

This section describes the Application Programming Interface (API) functions of the SMTP TCP/IP Stack.

Refer to each section for a detailed description.

5.8.21.7.1 Functions

5.8.21.7.1.1 TCPIP_SMTP_ArrayPut Function

C

```
uint16_t TCPIP_SMTP_ArrayPut(
    uint8_t* Data,
    uint16_t Len
);
```

Description

Writes a series of bytes to the SMTP client.

Preconditions

[TCPIP_SMTP_UsageBegin](#) returned true on a previous call.

Parameters

Parameters	Description
Data	The data to be written
Len	How many bytes should be written

Returns

The number of bytes written. If less than Len, then the TX FIFO became full before all bytes could be written.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, [TCPIP_SMTP_MailSend](#) was called with SMTPClient.Body set to NULL.)

5.8.21.7.1.2 TCPIP_SMTP_Flush Function**C**

```
void TCPIP_SMTP_Flush();
```

Description

Flushes the SMTP [socket](#) and forces all data to be sent.

Preconditions

[TCPIP_SMTP_UsageBegin](#) returned true on a previous call.

Returns

None

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, [TCPIP_SMTP_MailSend](#) was called with SMTPClient.Body set to NULL.)

5.8.21.7.1.3 TCPIP_SMTP_IsBusy Function**C**

```
bool TCPIP_SMTP_IsBusy();
```

Description

Call this function to determine if the SMTP client is busy performing background tasks. This function should be called after any call to [TCPIP_SMTP_MailSend](#), [TCPIP_SMTP_PutIsDone](#) to determine if the stack has finished performing its internal tasks. It should also be called prior to any call to [TCPIP_SMTP_IsPutReady](#) to verify that the SMTP client has not prematurely disconnected. When this function returns false, the next call should be to [TCPIP_SMTP_UsageEnd](#) to release the module and obtain the status code for the operation.

Preconditions

[TCPIP_SMTP_UsageBegin](#) returned true on a previous call.

Return Values

Return Values	Description
true	The SMTP Client is busy with internal tasks or sending an on-the-fly message.
false	The SMTP Client is terminated and is ready to be released.

5.8.21.7.1.4 TCPIP_SMTP_IsPutReady Function**C**

```
uint16_t TCPIP_SMTP_IsPutReady();
```

Description

Use this function to determine how much data can be written to the SMTP client when generating an on-the-fly message.

Preconditions

[TCPIP_SMTP_UsageBegin](#) returned true on a previous call, and an on-the-fly message is being generated. This requires that [TCPIP_SMTP_MailSend](#) was called with SMTPClient.Body set to NULL.

Returns

The number of free bytes the SMTP TX FIFO.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, [TCPIP_SMTP_MailSend](#) was called with SMTPClient.Body set to NULL.)

5.8.21.7.1.5 TCPIP_SMTP_MailSend Function**C**

```
void TCPIP_SMTP_MailSend(  
    SMTP_POINTERS* smtpClientMessage  
);
```

Description

This function starts the state machine that performs the actual transmission of the message. Call this function after all the fields in SMTPClient have been set.

Preconditions

[TCPIP_SMTP_UsageBegin](#) returned true on a previous call.

Parameters

Parameters	Description
smtpClientMessage	message to send

Returns

None

5.8.21.7.1.6 TCPIP_SMTP_Put Function**C**

```
bool TCPIP_SMTP_Put(  
    char c  
);
```

Description

Writes a single byte to the SMTP client.

Preconditions

[TCPIP_SMTP_UsageBegin](#) returned true on a previous call.

Parameters

Parameters	Description
c	The byte to be written

Return Values

Return Values	Description
true	The byte was successfully written

false	The byte was not written, most likely because the buffer was full

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, [TCPIP_SMTP_MailSend](#) was called with SMTPClient.Body set to NULL.)

5.8.21.7.1.7 TCPIP_SMTP_PutIsDone Function

C

```
void TCPIP_SMTP_PutIsDone();
```

Description

Indicates that the on-the-fly message is complete.

Preconditions

[TCPIP_SMTP_UsageBegin](#) returned true on a previous call, and the SMTP client is generated an on-the-fly message. (That is, [TCPIP_SMTP_MailSend](#) was called with SMTPClient.Body set to NULL.)

Returns

None

5.8.21.7.1.8 TCPIP_SMTP_StringPut Function

C

```
uint16_t TCPIP_SMTP_StringPut(
    char* Data
);
```

Description

Writes a string to the SMTP client.

Preconditions

[TCPIP_SMTP_UsageBegin](#) returned true on a previous call.

Parameters

Parameters	Description
Data	The data to be written

Returns

The number of bytes written. If less than the length of Data, then the TX FIFO became full before all bytes could be written.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, [TCPIP_SMTP_MailSend](#) was called with SMTPClient.Body set to NULL.)

5.8.21.7.1.9 TCPIP_SMTP_UsageBegin Function

C

```
bool TCPIP_SMTP_UsageBegin();
```

Description

Call this function before calling any other SMTP Client APIs. This function obtains a lock on the SMTP Client, which can only be used by one stack application at a time. Once the application is finished with the SMTP client, it must call

[TCPIP_SMTP_UsageEnd](#) to release control of the module to any other waiting applications.

This function initializes all the SMTP state machines and variables back to their default state.

Preconditions

None

Return Values

Return Values	Description
true	The application has successfully obtained control of the module
false	The SMTP module is in use by another application. Call TCPIP_SMTP_UsageBegin again later, after returning to the main program loop

5.8.21.7.1.10 TCPIP_SMTP_UsageEnd Function

C

```
uint16_t TCPIP_SMTP_UsageEnd();
```

Description

Call this function to release control of the SMTP client module once an application is finished using it. This function releases the lock obtained by [TCPIP_SMTP_UsageBegin](#), and frees the SMTP client to be used by another application.

Preconditions

[TCPIP_SMTP_UsageBegin](#) returned true on a previous call.

Return Values

Return Values	Description
SMTP_SUCCESS	A message was successfully sent
SMTP_RESOLVE_ERROR	The SMTP server could not be resolved
SMTP_CONNECT_ERROR	The connection to the SMTP server failed or was prematurely terminated
1-199 and 300-399	The last SMTP server response code

5.8.21.7.2 Data Types and Functions

5.8.21.7.2.1 SMTP_CONNECT_ERROR Macro

C

```
#define SMTP_CONNECT_ERROR (0x8001u)    // Connection to SMTP server failed
```

Description

Connection to SMTP server failed

5.8.21.7.2.2 SMTP_RESOLVE_ERROR Macro

C

```
#define SMTP_RESOLVE_ERROR (0x8000u)    // DNS lookup for SMTP server failed
```

Description

DNS lookup for SMTP server failed

5.8.21.7.2.3 SMTP_SUCCESS Macro

C

```
#define SMTP_SUCCESS (0x0000u)    // Message was successfully sent
```

Description

Message was successfully sent

5.8.21.7.2.4 SMTP_CLIENT_MODULE_CONFIG Structure

C

```
typedef struct {
} SMTP_CLIENT_MODULE_CONFIG;
```

Description

This is type SMTP_CLIENT_MODULE_CONFIG.

5.8.21.7.2.5 SMTP_POINTERS Structure

C

```
typedef struct {
    char* Server;
    char* Username;
    char* Password;
    char* To;
    char* CC;
    char* BCC;
    char* From;
    char* Subject;
    char* OtherHeaders;
    char* Body;
    bool UseSSL;
    uint16_t ServerPort;
} SMTP_POINTERS;
```

Description

This structure of pointers configures the SMTP Client to [send](#) an e-mail message.

Parameters

Parameters	Description
Server	the SMTP server to relay the message through
Username	the user name to use when logging into the SMTP server, if any is required
Password	the password to supply when logging in, if any is required
To	the destination address for this message. May be a comma-separated list of addresss, and/or formatted.
CC	The CC addresses for this message, if any. May be a comma-separated list of addresss, and/or formatted.
BCC	The BCC addresses for this message, if any. May be a comma-separated list of addresss, and/or formatted.
From	The From address for this message. May be formatted.
Subject	The Subject header for this message.
OtherHeaders	Any additional headers for this message. Each additional header, including the last one, must be terminated with a CRLF pair.

Body	When sending a message from memory, the location of the body of this message in memory. Leave as NULL to build a message on-the-fly.
UseSSL	When TCPIP_STACK_USE_SSL_CLIENT is enabled, this flag causes the SMTP client to make an SSL connection to the server.
ServerPort	(uint16_t value) Indicates the port on which to connect to the remote SMTP server.

Remarks

When formatting an e-mail address, the SMTP standard format for associating a printable name may be used. This format places the printable name in quotation marks, with the address following in pointed brackets, such as "John Smith" <john.smith@domain.com>

5.8.21.8 Files

Files

Name	Description
smtp.h	Module for Microchip TCP/IP Stack
smtp_config.h	SMTP configuration file

Description

5.8.21.8.1 smtp.h

Simple Mail Transfer Protocol (SMTP) Client

Functions

	Name	Description
≡	TCPIP_SMTP_ArrayPut	Writes a series of bytes to the SMTP client.
≡	TCPIP_SMTP_Flush	Flushes the SMTP socket and forces all data to be sent.
≡	TCPIP_SMTP_IsBusy	Determines if the SMTP client is busy.
≡	TCPIP_SMTP_IsPutReady	Determines how much data can be written to the SMTP client.
≡	TCPIP_SMTP_MailSend	Initializes the message sending process.
≡	TCPIP_SMTP_Put	Writes a single byte to the SMTP client.
≡	TCPIP_SMTP_PutIsDone	Indicates that the on-the-fly message is complete.
≡	TCPIP_SMTP_StringPut	Writes a string to the SMTP client.
≡	TCPIP_SMTP_UsageBegin	Requests control of the SMTP client module.
≡	TCPIP_SMTP_UsageEnd	Releases control of the SMTP client module.

Macros

	Name	Description
≡	SMTP_CONNECT_ERROR	Connection to SMTP server failed
≡	SMTP_RESOLVE_ERROR	DNS lookup for SMTP server failed
≡	SMTP_SUCCESS	Message was successfully sent

Structures

	Name	Description
≡	SMTP_CLIENT_MODULE_CONFIG	This is type SMTP_CLIENT_MODULE_CONFIG.

	SMTP_POINTERS	Configures the SMTP client to send a message

5.8.21.8.2 smtp_config.h

Simple Mail Transfer Protocol (SMTP) Configuration file

This file contains the SMTP module configuration options

Macros

	Name	Description
	MAX_SMTP_CONNECTIONS	Maximum number of SMTP connections allowed
	SMTP_PORT	Default port to use when unspecified
	SMTP_SERVER_REPLY_TIMEOUT	How long to wait before assuming the connection has been dropped (default 8 seconds)

5.8.22 SNMP TCP/IP Stack Library

5.8.22.1 Introduction

Simple Network Management Protocol (SNMP) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the SNMP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

5.8.22.2 Release Notes

MPLAB Harmony Version: v0.70b **SNMP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.22.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its

licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.22.4 Using the Library

This topic describes the basic architecture of the SNMP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [snmp.h](#)

The interface to the SNMP TCP/IP Stack library is defined in the "[snmp.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the SNMP TCP/IP Stack library should include "tcpip.h".

Library File:

The SNMP TCPIP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.22.4.1 Abstraction Model

This library provides the API of the SNMP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SNMP Software Abstraction Block Diagram

Note: This image was not available at time of release and will be added in a future release of MPLAB Harmony.

5.8.22.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the SNMP module.

Library Interface Section	Description
SNMP Module Functions	Routines to configure this module
SNMPv3 Module Functions	Routines to configure this module
SNMP Application Functions	Routines to configure an application
SNMPv3 Application Functions	Routines to configure an application
SNMP Data Types and Constants	This section provides various definitions describing this API

SNMPv3 Data Types and Constants

This section provides various definitions describing this API

5.8.22.5 Configuring the Library

Macros

	Name	Description
	<u>AUTH_LOCALIZED_PASSWORD_KEY_LEN</u>	SNMPv3 Authentication Localized password key length size
	<u>END_OF_SNMP_READ_COMMUNITIES</u>	??
	<u>END_OF_SNMP_WRITE_COMMUNITIES</u>	??
	<u>MY_DEFAULT_SNMP_IF</u>	For multi-homed hosts, the default SNMP interface
	<u>NOTIFY_COMMUNITY_LEN</u>	Length of Community name array
	<u>OID_MAX_LEN</u>	Change this to match your OID string length.
	<u>PRIV_LOCALIZED_PASSWORD_KEY_LEN</u>	SNMPv3 Privacy Password key length size
	<u>SNMP_BIB_FILE_NAME</u>	Name of the bib file for snmp
	<u>SNMP_COMMUNITY_MAX_LEN</u>	This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after <code>SNMP_COMMUNITY_MAX_LEN</code> for adding '0' NULL character.
	<u>SNMP_MAX_COMMUNITY_SUPPORT</u>	Specifying more strings than <code>SNMP_MAX_COMMUNITY_SUPPORT</code> will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present.
	<u>SNMP_MAX_MSG_SIZE</u>	SNMP MIN and MAX message 484 bytes in size. As per RFC 3411 <code>snmpEngineMaxMessageSize</code> and RFC 1157 (section 4- protocol specification) and implementation supports more than 484 whenever feasible.
	<u>SNMP_MAX_NON_REC_ID_OID</u>	Update the Non record id OID value which is part of <code>CustomSnmpDemo.c</code> file

	SNMP_READ_COMMUNITIES	Default SNMPv2C community names. These can be overridden at run time if alternate strings are present in external EEPROM or Flash (actual strings could be stored by the TCPIP storage service. These strings are case sensitive. An empty string means disabled (not matchable). For application security, these default community names should not be used, but should all be disabled to force the end user to select unique community names. These defaults are provided only to make it easier to start development. Specifying more strings than SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying... more
	SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE	This macro will be used to avoid SNMP OID memory buffer corruption
	SNMP_WRITE_COMMUNITIES	Default SNMPv2C community names. See SNMP_READ_COMMUNITIES for more information.
	SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 authentication localized Key length for memory validation
	SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 privacy key length size for memory validation
	SNMPV3_TRAP_MSG_PROCESS_MODEL_DB	This is macro SNMPV3_TRAP_MSG_PROCESS_MODEL_DB.
	SNMPV3_TRAP_SECURITY_LEVEL_DB	This is macro SNMPV3_TRAP_SECURITY_LEVEL_DB.
	SNMPV3_TRAP_SECURITY_MODEL_TYPE_DB	This is macro SNMPV3_TRAP_SECURITY_MODEL_TYPE_DB.
	SNMPV3_TRAP_USER_SECURITY_NAME_DB	This is macro SNMPV3_TRAP_USER_SECURITY_NAME_DB.
	SNMPV3_USER_AUTH_PASSWD_DB	This is macro SNMPV3_USER_AUTH_PASSWD_DB.
	SNMPV3_USER_AUTH_TYPE_DB	This is macro SNMPV3_USER_AUTH_TYPE_DB.
	SNMPV3_USER_PRIV_PASSWD_DB	This is macro SNMPV3_USER_PRIV_PASSWD_DB.
	SNMPV3_USER_PRIV_TYPE_DB	This is macro SNMPV3_USER_PRIV_TYPE_DB.
	SNMPV3_USER_SECURITY_NAME_DB	This is macro SNMPV3_USER_SECURITY_NAME_DB.
	SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE	User security name length for memory validation
	SNMPV3_USM_MAX_USER	Client configuration options
	TRAP_COMMUNITY_MAX_LEN	Maximum length of community name table
	TRAP_TABLE_SIZE	This table maintains list of interested receivers who should receive notifications when some interesting event occurs.

	USER_SECURITY_NAME_LEN	SNMPv3 User Security Name length

Structures

	Name	Description
	SNMP_NET_CONFIG	SNMP Network Configuration structure typedef

Description

The configuration of the SNMP TCP/IP Stack Library is based on the file tcpip_config.h

This header file contains the configuration selection for the SNMP TCP/IP Stack Library. Based on the selections made, the SNMP TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the SNMP TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.22.5.1 AUTH_LOCALIZED_PASSWORD_KEY_LEN Macro**C**

```
#define AUTH_LOCALIZED_PASSWORD_KEY_LEN (20)
```

Description

SNMPv3 Authentication Localized password key length size

5.8.22.5.2 END_OF_SNMP_READ_COMMUNITIES Macro**C**

```
#define END_OF_SNMP_READ_COMMUNITIES
```

Description

??

5.8.22.5.3 END_OF_SNMP_WRITE_COMMUNITIES Macro**C**

```
#define END_OF_SNMP_WRITE_COMMUNITIES
```

Description

??

5.8.22.5.4 MY_DEFAULT_SNMP_IF Macro**C**

```
#define MY_DEFAULT_SNMP_IF "PIC32INT"
```

Description

For multi-homed hosts, the default SNMP interface

5.8.22.5.5 NOTIFY_COMMUNITY_LEN Macro

C

```
#define NOTIFY_COMMUNITY_LEN (SNMP_COMMUNITY_MAX_LEN)
```

Description

Length of Community name array

5.8.22.5.6 OID_MAX_LEN Macro

C

```
#define OID_MAX_LEN (18)
```

Description

Change this to match your OID string length.

5.8.22.5.7 PRIV_LOCALIZED_PASSWORD_KEY_LEN Macro

C

```
#define PRIV_LOCALIZED_PASSWORD_KEY_LEN (20)
```

Description

SNMPv3 Privacy Password key length size

5.8.22.5.8 SNMP_BIB_FILE_NAME Macro

C

```
#define SNMP_BIB_FILE_NAME "snmp.bib"
```

Description

Name of the bib file for snmp

5.8.22.5.9 SNMP_COMMUNITY_MAX_LEN Macro

C

```
#define SNMP_COMMUNITY_MAX_LEN (8u)
```

Description

This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after SNMP_COMMUNITY_MAX_LEN for adding '0' NULL character.

5.8.22.5.10 SNMP_MAX_COMMUNITY_SUPPORT Macro

C

```
#define SNMP_MAX_COMMUNITY_SUPPORT (3u)
```

Description

Specifying more strings than SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present.

5.8.22.5.11 SNMP_MAX_MSG_SIZE Macro

C

```
#define SNMP_MAX_MSG_SIZE 484
```

Description

SNMP MIN and MAX message 484 bytes in size. As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 (section 4- protocol specification) and implementation supports more than 484 whenever feasible.

5.8.22.5.12 SNMP_MAX_NON_REC_ID_OID Macro

C

```
#define SNMP_MAX_NON_REC_ID_OID 3
```

Description

Update the Non record id OID value which is part of CustomSnmpDemo.c file

5.8.22.5.13 SNMP_READ_COMMUNITIES Macro

C

```
#define SNMP_READ_COMMUNITIES {"public", "read", ""}
```

Description

Default SNMPv2C community names. These can be overridden at run time if alternate strings are present in external EEPROM or Flash (actual strings could be stored by the TCPIP storage service. These strings are case sensitive. An empty string means disabled (not matchable). For application security, these default community names should not be used, but should all be disabled to force the end user to select unique community names. These defaults are provided only to make it easier to start development. Specifying more strings than [SNMP_MAX_COMMUNITY_SUPPORT](#) will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present. A string larger than [SNMP_COMMUNITY_MAX_LEN](#) bytes will be ignored.

5.8.22.5.14 SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE Macro

C

```
#define SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE (8)
```

Description

This macro will be used to avoid SNMP OID memory buffer corruption

5.8.22.5.15 SNMP_WRITE_COMMUNITIES Macro

C

```
#define SNMP_WRITE_COMMUNITIES {"private", "write", "public"}
```

Description

Default SNMPv2C community names. See [SNMP_READ_COMMUNITIES](#) for more information.

5.8.22.5.16 SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro

C

```
#define SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE (AUTH_LOCALIZED_PASSWORD_KEY_LEN+1)
```

Description

SNMPv3 authentication localized Key length for memory validation

5.8.22.5.17 SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro

C

```
#define SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE (PRIV_LOCALIZED_PASSWORD_KEY_LEN+1)
```

Description

SNMPv3 privacy key length size for memory validation

5.8.22.5.18 SNMPV3_TRAP_MSG_PROCESS_MODEL_DB Macro

C

```
#define SNMPV3_TRAP_MSG_PROCESS_MODEL_DB  
{SNMPV3_MSG_PROCESSING_MODEL, SNMPV3_MSG_PROCESSING_MODEL, SNMPV3_MSG_PROCESSING_MODEL}
```

Description

This is macro SNMPV3_TRAP_MSG_PROCESS_MODEL_DB.

5.8.22.5.19 SNMPV3_TRAP_SECURITY_LEVEL_DB Macro

C

```
#define SNMPV3_TRAP_SECURITY_LEVEL_DB {AUTH_PRIV, AUTH_NO_PRIV, NO_AUTH_NO_PRIV}
```

Description

This is macro SNMPV3_TRAP_SECURITY_LEVEL_DB.

5.8.22.5.20 SNMPV3_TRAP_SECURITY_MODEL_TYPE_DB Macro

C

```
#define SNMPV3_TRAP_SECURITY_MODEL_TYPE_DB  
{SNMPV3_USM_SECURITY_MODEL, SNMPV3_USM_SECURITY_MODEL, SNMPV3_USM_SECURITY_MODEL}
```

Description

This is macro SNMPV3_TRAP_SECURITY_MODEL_TYPE_DB.

5.8.22.5.21 SNMPV3_TRAP_USER_SECURITY_NAME_DB Macro

C

```
#define SNMPV3_TRAP_USER_SECURITY_NAME_DB {"microchip", "SnpAdmin", "root"}
```

Description

This is macro SNMPV3_TRAP_USER_SECURITY_NAME_DB.

5.8.22.5.22 SNMPV3_USER_AUTH_PASSWD_DB Macro

C

```
#define SNMPV3_USER_AUTH_PASSWD_DB {"auth12345", "ChandlerUS", ""}
```

Description

This is macro SNMPV3_USER_AUTH_PASSWD_DB.

5.8.22.5.23 SNMPV3_USER_AUTH_TYPE_DB Macro

C

```
#define SNMPV3_USER_AUTH_TYPE_DB {SNMPV3_HMAC_MD5, SNMPV3_HMAC_SHA1, SNMPV3_NO_HMAC_AUTH}
```

Description

This is macro SNMPV3_USER_AUTH_TYPE_DB.

5.8.22.5.24 SNMPV3_USER_PRIV_PASSWD_DB Macro

C

```
#define SNMPV3_USER_PRIV_PASSWD_DB {"priv12345", "", ""}
```

Description

This is macro SNMPV3_USER_PRIV_PASSWD_DB.

5.8.22.5.25 SNMPV3_USER_PRIV_TYPE_DB Macro

C

```
#define SNMPV3_USER_PRIV_TYPE_DB {SNMPV3_AES_PRIV, SNMPV3_NO_PRIV, SNMPV3_NO_PRIV}
```

Description

This is macro SNMPV3_USER_PRIV_TYPE_DB.

5.8.22.5.26 SNMPV3_USER_SECURITY_NAME_DB Macro

C

```
#define SNMPV3_USER_SECURITY_NAME_DB {"microchip", "SnmpAdmin", "root"}
```

Description

This is macro SNMPV3_USER_SECURITY_NAME_DB.

5.8.22.5.27 SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE Macro

C

```
#define SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE (USER_SECURITY_NAME_LEN+1)
```

Description

User security name length for memory validation

5.8.22.5.28 SNMPV3_USM_MAX_USER Macro

C

```
#define SNMPV3_USM_MAX_USER 3
```

Description

Client configuration options

5.8.22.5.29 TRAP_COMMUNITY_MAX_LEN Macro

C

```
#define TRAP_COMMUNITY_MAX_LEN (SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE+1)
```

Description

Maximum length of community name table

5.8.22.5.30 TRAP_TABLE_SIZE Macro

C

```
#define TRAP_TABLE_SIZE (2)
```

Description

This table maintains list of interested receivers who should receive notifications when some interesting event occurs.

5.8.22.5.31 USER_SECURITY_NAME_LEN Macro

C

```
#define USER_SECURITY_NAME_LEN (16)
```

Description

SNMPv3 User Security Name length

5.8.22.5.32 SNMP_NET_CONFIG Structure

C

```
typedef struct {  
    uint8_t readCommunity[SNMP_MAX_COMMUNITY_SUPPORT][SNMP_COMMUNITY_MAX_LEN+1];  
    uint8_t writeCommunity[SNMP_MAX_COMMUNITY_SUPPORT][SNMP_COMMUNITY_MAX_LEN+1];  
    uint32_t SnmpEngineBootRcrd;  
} SNMP_NET_CONFIG;
```

Description

SNMP Network Configuration structure typedef

Members

Members	Description
uint8_t readCommunity[SNMP_MAX_COMMUNITY_SUPPORT][SNMP_COMMUNITY_MAX_LEN+1];	SNMPv2C Read community names SNMP_COMMUNITY_MAX_LEN (8) + 1 null termination byte
uint8_t writeCommunity[SNMP_MAX_COMMUNITY_SUPPORT][SNMP_COMMUNITY_MAX_LEN+1];	SNMPv2C Write community names SNMP_COMMUNITY_MAX_LEN (8) + 1 null termination byte
uint32_t SntpEngineBootRcrd;	SNMP Engine Boot Record location

5.8.22.6 Building the Library

This section list the files that are available in the \src of the SNMP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.22.7 Library Interface

SNMP Application Functions

	Name	Description
≡	SNMPGetExactIndex	To search for exact index node in case of a Sequence variable.
≡	SNMPGetNextIndex	To search for next index node in case of a Sequence variable.
≡	SNMPGetVar	Used to Get/collect OID variable information.
≡	SNMPIsValidSetLen	Validates the set variable data length to data type.
≡	SNMPSendTrap	Prepare, validate remote node which will receive trap and send trap pdu.
≡	SNMPSetVar	This routine Set the mib variable with the requested value.
≡	SNMPValidateCommunity	Validates community name for access control.

SNMP Data Types and Constants

	Name	Description
≡	SNMP_END_OF_VAR	This is macro SNMP_END_OF_VAR .

	SNMP_H	FileName: SNMP.h Copyright © 2012 released Microchip Technology Inc. All rights reserved. Microchip licenses to you the right to use, modify, copy and distribute Software only when embedded on a Microchip microcontroller or digital signal controller that is integrated into your product or third party product (pursuant to the sublicense terms in the accompanying license agreement). You should refer to the license agreement accompanying this Software for additional information regarding your rights and obligations. SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND... more
	SNMP_INDEX_INVALID	This is macro SNMP_INDEX_INVALID.
	SNMP_START_OF_VAR	
	SNMP_V1	
	SNMP_V2C	This is macro SNMP_V2C.
	SNMP_V3	This is macro SNMP_V3.
	GENERIC_TRAP_NOTIFICATION_TYPE	This is type GENERIC_TRAP_NOTIFICATION_TYPE.
	IPV6_SNMP_TRAP_INFO	This is type IPV6_SNMP_TRAP_INFO.
	SNMP_BUFFER_DATA	SNMPv1/v2c
	SNMP_COMMUNITY_TYPE	This is type SNMP_COMMUNITY_TYPE.
	SNMP_ID	This is the SNMP OID variable id. This id is assigned via MIB file. Only dynamic and AgentID variables can contain ID. MIB2BIB utility enforces this rule when BIB was generated. typedef int SNMP_ID;
	SNMP_INDEX	This is type SNMP_INDEX.
	SNMP_MODULE_CONFIG	This is type SNMP_MODULE_CONFIG.
	SNMP_NON_MIB_RECD_INFO	This is type SNMP_NON_MIB_RECD_INFO.
	SNMP_NOTIFY_INFO	// the only if that runs SNMP
	SNMP_PROCESSING_MEM_INFO_PTRS	This is type SNMP_PROCESSING_MEM_INFO_PTRS.
	SNMP_STACK_DCPT_STUB	
	SNMP_TRAP_INFO	This is type SNMP_TRAP_INFO.
	SNMP_TRAP_IP_ADDRESS_TYPE	This is type SNMP_TRAP_IP_ADDRESS_TYPE.
	SNMP_VAL	This is type SNMP_VAL.
	TCPIP_SNMP_DCPT	TCPIP_STACK_USE_IPV6
	TCPIP_SNMP_SM	This is type TCPIP_SNMP_SM.
	VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE	This is type VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE.
	tIPV6_SNMP_TRAP_INFO	This is type IPV6_SNMP_TRAP_INFO.
	tSNMP_TRAP_INFO	This is type SNMP_TRAP_INFO.

SNMP Module Functions

	Name	Description
	TCPIP_SNMP_ClientGetNet	
	TCPIP_SNMP_DataCopyToProcessBuffer	Copies uint8_t data to dynamically allocated memory buffer.
	TCPIP_SNMP_EventNotifyGet	To Get the IPv6 DHCP event notification.
	TCPIP_SNMP_Notify	Creates and Sends TRAP pdu.

	TCPIP_SNMP_NotifyIsReady	Resolves given remoteHost IP address into MAC address.
	TCPIP_SNMP_NotifyPrepare	Collects trap notification info and send ARP to remote host.
	TCPIP_SNMP_PacketProcStubPtrsGet	Get SNMP packet processing memory pointer.
	TCPIP_SNMP_ProcessBufferDataGet	Reads uint8_t data from dynamically allocated memory buffer.
	TCPIP_SNMP_ReadCommunityGet	Get the readCommunity String with Snmp index.
	TCPIP_SNMP_TrapTimeGet	Get SNMP Trap UDP client open socket timeout.
	TCPIP_SNMP_WriteCommunityGet	Get the writeCommunity String with Snmp index.

SNMPv3 Application Functions

	Name	Description
	SNMPv3ComputeHMACIpadOpadForAuthLocalizedKey	Compute HMAC inner and outer pad for authorization localized key.
	SNMPv3USMAuthPrivPswdLocalization	Convert Auth and Priv password to the localized Key using SNMPEngineID.
	TCPIP_SNMPv3_CmprTrapSecNameAndSecLvIWithUSMDb	Routine to find the index of the user name in the user data base table.

SNMPv3 Data Types and Constants

	Name	Description
	SNMPV3_PROCESSING_MEM_INFO_PTRS	SNMPv3 Processing Memory Pointers
	SNMPV3_STACK_DCPT_STUB	SNMPv3 Descriptor Structure Typedef

SNMPv3 Module Functions

	Name	Description
	TCPIP_SNMPv3_Notify	Creates and Sends SNMPv3 TRAP pdu.
	TCPIP_SNMPV3_PacketProcStubPtrsGet	Get SNMPv3 packet processing memory pointer.

Description

This section describes the Application Programming Interface (API) functions of the SNMP TCP/IP Stack.

Refer to each section for a detailed description.

5.8.22.7.1 SNMP Module Functions

5.8.22.7.1.1 TCPIP_SNMP_ClientGetNet Function

C

```
TCPIP_NET_HANDLE TCPIP_SNMP_ClientGetNet();
```

5.8.22.7.1.2 TCPIP_SNMP_DataCopyToProcessBuffer Function

C

```
bool TCPIP_SNMP_DataCopyToProcessBuffer(
    uint8_t val,
    SNMP_BUFFER_DATA * putbuf
);
```

Description

The SNMPv1 and v2c stack implementation uses dynamically allocated memory buffer for processing of request and response packets. This routine copies the uint8_t data to the allocated buffer and updates the offset length counter.

Preconditions

The SNMPv1 and v2c stack has successfully allocated dynamic memory buffer from the Heap

Parameters

Parameters	Description
val	uint8_t value to be written to the buffer
putbuf	pointer to the dynamically allocated buffer to which the 'val' to be written

Return Values

Return Values	Description
true	if successfully write to the buffer
false	failure in writing to the buffer

Remarks

This routine is used by the SNMP stack. If required to be used by the application code, valid pointers should be passed to this routine.

5.8.22.7.1.3 TCPIP_SNMP_EventNotifyGet Function**C**

```
IPV6_EVENT_TYPE TCPIP_SNMP_EventNotifyGet(  
    TCPIP_NET_HANDLE hNet  
);
```

Description

This routine is used to get the DHCP with IPv6 notification if the IPv6 unicast address is updated.

Preconditions

TCPIP_SNMP_Initialize() is called.

Parameters

Parameters	Description
hNet	Interface

Remarks

None.

5.8.22.7.1.4 TCPIP_SNMP_Notify Function**C**

```
bool TCPIP_SNMP_Notify(  
    SNMP_ID var,  
    SNMP_VAL val,  
    SNMP_INDEX index  
);
```

Description

This function creates SNMP V2 Trap PDU and sends it to previously specified remoteHost.

SNMP V1 trap pdu: | PDU-type | enterprise | agent-addr | generic-trap | specific-trap | | time-stamp | varbind-list |

The v1 enterprise is mapped directly to SNMPv2TrapOID.0

SNMP V2 trap pdu: version (0 or 1) | community | SNMP-PDU | pdu-type | request-id | error-status | err-index | varbinds

The first two variables (in varbind-list) of snmpv2 are: sysUpTime.0 and SNMPv2TrapOID.0

Generic Trap OID is used as the varbind for authentication failure.

Preconditions

[TCPIP_SNMP_NotifyIsReady\(\)](#) is already called and returned true.

Parameters

Parameters	Description
var	SNMP var ID that is to be used in notification
val	Value of var. Only value of uint8_t, uint16_t or uint32_t can be sent.
index	Index of var. If this var is a single, index would be 0, or else if this var is a sequence, index could be any value from 0 to 127

Return Values

Return Values	Description
true	if SNMP notification was successful sent. This does not guarantee that remoteHost received it.
false	Notification sent failed.
This would fail under following conditions	1) Given SNMP_BIB_FILE does not exist in file system 2) Given var does not exist. 3) Previously given agentID does not exist
4) Data type of given var is unknown	possible if file system itself was corrupted.

Remarks

This would fail if there were not UDP [socket](#) to open.

5.8.22.7.1.5 TCPIP_SNMP_NotifyIsReady Function

C

```
bool TCPIP_SNMP_NotifyIsReady(  
    IP_MULTI_ADDRESS* remoteHost,  
    SNMP_TRAP_IP_ADDRESS_TYPE eTrapMultiAddressType  
);
```

Description

This function resolves given remoteHost IP address into MAC address using ARP module. If remoteHost is not available, this function would never return true. Application must implement timeout logic to handle "remoteHost not available" situation.

Preconditions

[TCPIP_SNMP_NotifyPrepare\(\)](#) is already called.

Parameters

Parameters	Description
remoteHost	Pointer to remote Host IP address

Return Values

Return Values	Description
true	If remoteHost IP address is resolved and TCPIP_SNMP_Notify may be called.
false	If remoteHost IP address is not resolved.

Remarks

This would fail if there were not UDP [socket](#) to open.

5.8.22.7.1.6 TCPIP_SNMP_NotifyPrepare Function

C

```
void TCPIP_SNMP_NotifyPrepare(  
    IP_MULTI_ADDRESS* remoteHost,  
    char* community,  
    uint8_t communityLen,  
    SNMP_ID agentIDVar,  
    uint8_t notificationCode,  
    uint32_t timestamp  
);
```

Description

This function prepares SNMP module to [send](#) SNMP trap notification to remote host. It sends ARP request to remote host to learn remote host MAC address.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Parameters

Parameters	Description
remoteHost	pointer to remote Host IP address
community	Community string to use to notify
communityLen	Community string length
agentIDVar	System ID to use identify this agent
notificaitonCode	Notification Code to use
timestamp	Notification timestamp in 100th of second.

Returns

None

Remarks

This is first of series of functions to complete SNMP notification.

5.8.22.7.1.7 TCPIP_SNMP_PacketProcStubPtrsGet Function

C

```
void TCPIP_SNMP_PacketProcStubPtrsGet(  
    SNMP_PROCESSING_MEM_INFO_PTRS * dynMemInfoPtr  
);
```

Description

This function is used to get dynamic memory allocation pointer details which is used for SNMP packet processing.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Parameters

Parameters	Description
dynMemInfoPtr	Dynamic memory pointer for packet processing

Remarks

None.

5.8.22.7.1.8 TCPIP_SNMP_ProcessBufferDataGet Function

C

```
uint8_t TCPIP_SNMP_ProcessBufferDataGet(  
    SNMP_BUFFER_DATA getbuf,  
    uint16_t pos  
);
```

Description

The SNMPstack implementation uses dynamically allocated memory buffer for processing of request and response packets. This routine reads the uint8_t data from the allocated buffer at the positions (offset) provided.

Preconditions

The SNMPv1 and v2c stack has successfully allocated dynamic memory buffer from the Heap

Parameters

Parameters	Description
getbuf	Structure from where to read the data byte.
pos	position in the buffer from which the data to be read

Return Values

Return Values	Description
uint8_t	1 byte value read

Remarks

The read position offset is required to be provided every time the routine is called. This API do not increment the buffer read offset automatically, everytime it is called.

5.8.22.7.1.9 TCPIP_SNMP_ReadCommunityGet Function

C

```
uint8_t* TCPIP_SNMP_ReadCommunityGet(  
    int index  
);
```

Description

Get the readCommunity String with Snmp index.

Preconditions

_SNMP_ProcessVariables() is called.

Parameters

Parameters	Description
index	SNMP_MAX_COMMUNITY_SUPPORT .

Return Values

Return Values	Description
uint8_t *	unsigned char community string

Remarks

None.

5.8.22.7.1.10 TCPIP_SNMP_TrapTimeGet Function

C

```
SYS_TICK TCPIP_SNMP_TrapTimeGet();
```

Description

This function returns a SYS_TICK time(snmpTrapTimer) which is used to timeout a SNMP TRAP notification for a HOST. snmpTrapTimer is initialized when there is UDP client [socket](#) open either for a HOST IPv4 or IPv6 address.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Remarks

None.

5.8.22.7.1.11 TCPIP_SNMP_WriteCommunityGet Function

C

```
uint8_t* TCPIP_SNMP_WriteCommunityGet(  
    int index  
);
```

Description

Get the writeCommunity String with Snmp index.

Preconditions

_SNMP_ProcessVariables() is called.

Parameters

Parameters	Description
index	SNMP_MAX_COMMUNITY_SUPPORT .

Return Values

Return Values	Description
uint8_t *	unsigned char community string

Remarks

None.

5.8.22.7.2 SNMPv3 Module Functions

5.8.22.7.2.1 TCPIP_SNMPv3_Notify Function

C

```
bool TCPIP_SNMPv3_Notify(  
    SNMP_ID var,  
    SNMP_VAL val,  
    SNMP_INDEX index,  
    uint8_t targetIndex  
);
```

Description

This function creates SNMPv3 trap PDU and sends it to previously specified remoteHost.

Preconditions

TRAP event is triggered.

Parameters

Parameters	Description
var	SNMP var ID that is to be used in notification
val	Value of var. Only value of uint8_t, uint16_t or uint32_t can be sent.
index	Index of var. If this var is a single, index would be 0, or else if this var is a sequence, index could be any value from 0 to 127 targetIndex -index of the 'Snmpv3TrapConfigData' table's security user name for which the TRAP PDU message header to constructed.

Return Values

Return Values	Description
true	if SNMP notification was successful sent. This does not guarantee that remoteHost recieved it.
false	Notification sent failed.
This would fail under following contions	1) Given SNMP_BIB_FILE does not exist in file system 2) Given var does not exist. 3) Previously given agentID does not exist
4) Data type of given var is unknown	only possible if file system itself was corrupted. SNMPV3_MSG_PRIV_FAIL -encryption of the trap msg failed
SNMPV3_MSG_AUTH_FAIL	HAMC of the trap msg failed

Remarks

None

5.8.22.7.2.2 TCPIP_SNMPV3_PacketProcStubPtrsGet Function**C**

```
void TCPIP_SNMPV3_PacketProcStubPtrsGet(  
    SNMPV3_PROCESSING_MEM_INFO_PTRS * dynMemInfoPtr  
);
```

Description

This function is used to get dynamic memory allocation pointer details which is used for SNMPv3 packet processing.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Parameters

Parameters	Description
dynMemInfoPtr	Dynamic memory pointer for packet processing

Remarks

The source code for this routine is found in snmp.c, not snmpv3.c.

5.8.22.7.3 SNMP Application Functions

5.8.22.7.3.1 SNMPGetExactIndex Function

C

```
bool SNMPGetExactIndex(  
    SNMP_ID var,  
    SNMP_INDEX * index  
);
```

Description

This is a callback function called by SNMP module. SNMP user must implement this function in user application and provide appropriate data when called. This function will only be called for OID variable of type sequence.

Preconditions

None

Parameters

Parameters	Description
var	Variable id as per mib.h (input)
index	Index of variable (input)

Return Values

Return Values	Description
true	If the exact index value exists for given variable at given index.
false	Otherwise.

Remarks

Only sequence index needs to be handled in this function.

5.8.22.7.3.2 SNMPGetNextIndex Function

C

```
bool SNMPGetNextIndex(  
    SNMP_ID var,  
    SNMP_INDEX* index  
);
```

Description

This is a callback function called by SNMP module. SNMP user must implement this function in user application and provide appropriate data when called. This function will only be called for OID variable of type sequence.

Preconditions

None

Parameters

Parameters	Description
var	Variable id whose value is to be returned
index	Next Index of variable that should be transferred

Return Values

Return Values	Description
true	If a next index value exists for given variable at given index and index parameter contains next valid index.

false	Otherwise.

Remarks

Only sequence index needs to be handled in this function.

5.8.22.7.3.3 SNMPGetVar Function

C

```
bool SNMPGetVar(  
    SNMP_ID var,  
    SNMP_INDEX index,  
    uint8_t* ref,  
    SNMP_VAL* val  
);
```

Description

This is a callback function called by SNMP module. SNMP user must implement this function in user application and provide appropriate data when called.

Preconditions

None

Parameters

Parameters	Description
var	Variable id whose value is to be returned
index	Index of variable that should be transferred
ref	Variable reference used to transfer multi-byte data It is always SNMP_START_OF_VAR when very first byte is requested. Otherwise, use this as a reference to keep track of multi-byte transfers.
val	Pointer to up to 4 byte buffer. If var data type is uint8_t, transfer data in val->byte If var data type is uint16_t, transfer data in val->word If var data type is uint32_t, transfer data in val->dword If var data type is IP_ADDRESS, transfer data in val->v[] or val->dword If var data type is COUNTER32, TIME_TICKS or GAUGE32, transfer data in val->dword If var data type is ASCII_STRING or OCTET_STRING transfer data in val->byte using multi-byte transfer mechanism.

Return Values

Return Values	Description
true	If a value exists for given variable at given index.
false	Otherwise.

Remarks

None.

5.8.22.7.3.4 SNMPIsValidSetLen Function

C

```
bool SNMPIsValidSetLen(  
    SNMP_ID var,  
    uint8_t len,  
    uint8_t index  
);
```

Description

This routine is used to validate the dynamic variable data length to the variable data type. It is used when SET request is processed. This is a callback function called by module. User application must implement this function.

Preconditions

TCPIP_SNMP_ProcessSetVar() is called.

Parameters

Parameters	Description
var	Variable id whose value is to be set
len	Length value that is to be validated.

Return Values

Return Values	Description
true	if given var can be set to given len
false	if otherwise.

Remarks

This function will be called for only dynamic variables that are defined as ASCII_STRING and OCTET_STRING (i.e. data length greater than 4 bytes)

5.8.22.7.3.5 SNMPSendTrap Function

C

```
void SNMPSendTrap();
```

Description

This function is used to [send](#) trap notification to previously configured ip address if trap notification is enabled. There are different trap notification code. The current implementation sends trap for authentication failure (4).

Preconditions

If application defined event occurs to [send](#) the trap.

Returns

None.

Remarks

This is a callback function called by the application on certain predefined events. This routine only implemented to [send](#) a authentication failure Notification-type macro with PUSH_BUTTON oid stored in MPFS. If the ARP is not resolved i.e. if [TCPIP_SNMP_NotifyIsReady\(\)](#) returns false, this routine times out in 5 seconds. This routine should be modified according to event occurred and should update corresponding OID and notification type to the trap pdu.

5.8.22.7.3.6 SNMPSetVar Function

C

```
bool SNMPSetVar(  
    SNMP_ID var,  
    SNMP_INDEX index,  
    uint8_t ref,  
    SNMP_VAL val  
);
```

Description

This is a callback function called by module for the snmp SET request. User application must modify this function for the new variables address.

Preconditions

TCPIP_SNMP_ProcessVariables() is called.

Parameters

Parameters	Description
var	Variable id whose value is to be set
ref	Variable reference used to transfer multi-byte data 0 if first byte is set otherwise nonzero value to indicate corresponding byte being set.
val	Up to 4 byte data value. If var data type is uint8_t, variable value is in val->byte If var data type is uint16_t, variable value is in val->word If var data type is uint32_t, variable value is in val->dword. If var data type is IP_ADDRESS, COUNTER32, or GAUGE32, value is in val->dword If var data type is OCTET_STRING, ASCII_STRING value is in val->byte; multi-byte transfer will be performed to transfer remaining bytes of data.

Return Values

Return Values	Description
true	if it is OK to set more byte(s).
false	if otherwise.

Remarks

This function may get called more than once depending on number of bytes in a specific set request for given variable. only dynamic read-write variables needs to be handled.

5.8.22.7.3.7 SNMPValidateCommunity Function**C**

```
uint8_t SNMPValidateCommunity(  
    uint8_t* community  
);
```

Description

This function validates the community name for the mib access to NMS. The snmp community name received in the request pdu is validated for read and write community names. The agent gives an access to the mib variables only if the community matches with the predefined values. This routine also sets a global flag to [send](#) trap if authentication failure occurs.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
community	Pointer to community string as sent by NMS.

Returns

This routine returns the community validation result as READ_COMMUNITY or WRITE_COMMUNITY or INVALID_COMMUNITY

Remarks

This is a callback function called by module. User application must implement this function and verify that community matches

with predefined value. This validation occurs for each NMS request.

5.8.22.7.4 SNMPv3 Application Functions

5.8.22.7.4.1 SNMPv3ComputeHMACIpadOpadForAuthLoclzedKey Function

C

```
void SNMPv3ComputeHMACIpadOpadForAuthLoclzedKey(  
    uint8_t userDBIndex  
);
```

Description

This routine computes HMAC inner and outer pad strings for authorization localized key. RFC - 2104.

Preconditions

TCPIP_SNMPv3_Initialize() and ProcessVariabels() are called.

Parameters

Parameters	Description
userDBIndex	password storage poniter

Remarks

None

5.8.22.7.4.2 SNMPv3USMAuthPrivPswdLocalization Function

C

```
void SNMPv3USMAuthPrivPswdLocalization(  
    uint8_t userDBIndex  
);
```

Description

This routine converts MD5 or SHA1 and AES privacy password key to localized key using snmpEngineID(RFC- 3414 - section 6).

Preconditions

TCPIP_SNMPv3_Initialize() and ProcessVariabels() are called.

Parameters

Parameters	Description
userDBIndex	authentication protocol type

Remarks

None

5.8.22.7.4.3 TCPIP_SNMPv3_CmprTrapSecNameAndSecLvIWithUSMDb Function

C

```
bool TCPIP_SNMPv3_CmprTrapSecNameAndSecLvIWithUSMDb(  
    uint8_t tragetIndex,  
    uint8_t userTrapSecLen,  
    uint8_t * userTrapSecurityName,  
    STD_BASED_SNMPV3_SECURITY_LEVEL securityLevel  
);
```

Description

There are two different data base tables defined with SNMPv3 stack, like 'UserInfoDataBase' and 'Snmpv3TrapConfigData'. This routine is used to validate the trap user security level setting with SET request.

Preconditions

SET operation would be allowed if the USM security conditions and user security name in the request is matched to one of the user security name stored in the usm user database.

Parameters

Parameters	Description
userTrapSecLen	user sec name length in the SET request
userTrapSecurityName	pointer to user sec name in the SET request
securityLevel	trap security level to be SET on the agent

Return Values

Return Values	Description
true	if the trap target user sec level setting is successful
FLASE	If the SET failed due to non matching of the security parameters

Remarks

None.

5.8.22.7.5 SNMP Data Types and Constants

5.8.22.7.5.1 SNMP_END_OF_VAR Macro

C

```
#define SNMP_END_OF_VAR (0xff)
```

Description

This is macro SNMP_END_OF_VAR.

5.8.22.7.5.2 SNMP_H Macro

C

```
#define SNMP_H
```

Description

FileName: SNMP.h Copyright © 2012 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute Software only when embedded on a Microchip microcontroller or digital signal controller that is integrated into your product or third party product (pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

5.8.22.7.5.3 SNMP_INDEX_INVALID Macro

C

```
#define SNMP_INDEX_INVALID (0xff)
```

Description

This is macro SNMP_INDEX_INVALID.

5.8.22.7.5.4 SNMP_START_OF_VAR Macro

C

```
#define SNMP_START_OF_VAR (0)
```

5.8.22.7.5.5 SNMP_V1 Macro

C

```
#define SNMP_V1 (0)
```

5.8.22.7.5.6 SNMP_V2C Macro

C

```
#define SNMP_V2C (1)
```

Description

This is macro SNMP_V2C.

5.8.22.7.5.7 SNMP_V3 Macro

C

```
#define SNMP_V3 (3)
```

Description

This is macro SNMP_V3.

5.8.22.7.5.8 GENERIC_TRAP_NOTIFICATION_TYPE Enumeration

C

```
typedef enum {  
    COLD_START = 0x0,  
    WARM_START = 0x1,  
    LINK_DOWN = 0x2,  
    LINK_UP = 0x3,  
    AUTH_FAILURE = 0x4,  
    EGP_NEBOR_LOSS = 0x5,  
    ENTERPRISE_SPECIFIC = 0x6  
} GENERIC_TRAP_NOTIFICATION_TYPE;
```

Description

This is type GENERIC_TRAP_NOTIFICATION_TYPE.

5.8.22.7.5.9 IPV6_SNMP_TRAP_INFO Structure

C

```
typedef struct tIPV6_SNMP_TRAP_INFO {  
    uint8_t Size;  
    struct {  
        uint8_t communityLen;  
        char community[TRAP_COMMUNITY_MAX_LEN];  
        IPV6_ADDR IPv6Address;  
        struct {  
            unsigned int bEnabled : 1;  
        } Flags;  
    } table[TRAP_TABLE_SIZE];  
} IPV6_SNMP_TRAP_INFO;
```

Description

This is type IPV6_SNMP_TRAP_INFO.

Members

Members	Description
uint8_t communityLen;	Community name length
char community[TRAP_COMMUNITY_MAX_LEN];	Community name array
IPV6_ADDR IPv6Address;	IPv6 address to which trap to be sent
unsigned int bEnabled : 1;	Trap enabled flag

5.8.22.7.5.10 SNMP_BUFFER_DATA Structure

C

```
typedef struct {  
    uint8_t * head;  
    uint16_t length;  
} SNMP_BUFFER_DATA;
```

Description

SNMPv1/v2c

5.8.22.7.5.11 SNMP_COMMUNITY_TYPE Enumeration

C

```
typedef enum {  
    READ_COMMUNITY = 1,  
    WRITE_COMMUNITY = 2,  
    INVALID_COMMUNITY = 3  
} SNMP_COMMUNITY_TYPE;
```

Description

This is type SNMP_COMMUNITY_TYPE.

Members

Members	Description
READ_COMMUNITY = 1	Read only community
WRITE_COMMUNITY = 2	Read write community
INVALID_COMMUNITY = 3	Community invalid

5.8.22.7.5.12 SNMP_ID Type

C

```
typedef uint32_t SNMP_ID;
```

Description

This is the SNMP OID variable id. This id is assigned via MIB file. Only dynamic and AgentID variables can contain ID. MIB2BIB utility enforces this rule when BIB was generated. typedef int SNMP_ID;

5.8.22.7.5.13 SNMP_INDEX Type

C

```
typedef uint8_t SNMP_INDEX;
```

Description

This is type SNMP_INDEX.

5.8.22.7.5.14 SNMP_MODULE_CONFIG Structure

C

```
typedef struct {  
} SNMP_MODULE_CONFIG;
```

Description

This is type SNMP_MODULE_CONFIG.

5.8.22.7.5.15 SNMP_NON_MIB_RECD_INFO Structure

C

```
typedef struct {  
    uint8_t oidstr[16];  
    uint8_t version;  
} SNMP_NON_MIB_RECD_INFO;
```

Description

This is type SNMP_NON_MIB_RECD_INFO.

5.8.22.7.5.16 SNMP_NOTIFY_INFO Structure

C

```
typedef struct {  
    char community[NOTIFY_COMMUNITY_LEN];  
    uint8_t communityLen;  
    SNMP_ID agentIDVar;  
    uint8_t notificationCode;  
    UDP_SOCKET socket;  
    TCPIP_UINT32_VAL timestamp;  
    SNMP_ID trapIDVar;  
    TCPIP_NET_HANDLE snmpTrapInf;  
} SNMP_NOTIFY_INFO;
```

Description

// the only if that runs SNMP

Members

Members	Description
char community[NOTIFY_COMMUNITY_LEN];	Community name array
uint8_t communityLen;	Community name length
SNMP_ID agentIDVar;	Agent id for trap identification
uint8_t notificationCode;	Trap notification code
UDP_SOCKET socket;	Udp socket number
TCPIP_UINT32_VAL timestamp;	Time stamp for trap
SNMP_ID trapIDVar;	SNMPV2 specific trap
TCPIP_NET_HANDLE snmpTrapInf;	interface we use for the SNMP TRAP

5.8.22.7.5.17 SNMP_PROCESSING_MEM_INFO_PTRS Structure**C**

```
typedef struct {
    SNMP_STACK_DCPT_STUB* snmpStkDynMemStubPtr;
    const void* snmpHeapMemHandler;
    TCPIP_SNMP_DCPT* snmpDcptPtr;
} SNMP_PROCESSING_MEM_INFO_PTRS;
```

Description

This is type SNMP_PROCESSING_MEM_INFO_PTRS.

5.8.22.7.5.18 SNMP_STACK_DCPT_STUB Structure**C**

```
typedef struct {
    uint8_t gOIDCorrespondingSnmplibID;
    uint8_t gSetTrapSendFlag;
    bool getZeroInstance;
    uint8_t appendZeroToOID;
    uint8_t gSendTrapFlag;
    uint8_t gGenericTrapNotification;
    uint8_t gSpecificTrapNotification;
    SNMP_NOTIFY_INFO SNMPNotifyInfo;
    SNMP_BUFFER_DATA outPduBufData;
    SNMP_BUFFER_DATA trapPduOutBufData;
    SNMP_NET_CONFIG snmpNetConfig;
} SNMP_STACK_DCPT_STUB;
```

Members

Members	Description
SNMP_NOTIFY_INFO SNMPNotifyInfo;	notify info for trap

5.8.22.7.5.19 SNMP_TRAP_INFO Structure**C**

```
typedef struct tSNMP_TRAP_INFO {
    uint8_t Size;
    struct {
        uint8_t communityLen;
        char community[TRAP_COMMUNITY_MAX_LEN];
        IPV4_ADDR IPAddress;
        struct {
```

```

    unsigned int bEnabled : 1;
    } Flags;
    } table[TRAP_TABLE_SIZE];
} SNMP_TRAP_INFO;

```

Description

This is type SNMP_TRAP_INFO.

Members

Members	Description
uint8_t communityLen;	Community name length
char community[TRAP_COMMUNITY_MAX_LEN];	Community name array
IPV4_ADDR IPAddress;	IP address to which trap to be sent
unsigned int bEnabled : 1;	Trap enabled flag

5.8.22.7.5.20 SNMP_TRAP_IP_ADDRESS_TYPE Enumeration

C

```

typedef enum {
    IPV4_SNMP_TRAP = 1,
    IPV6_SNMP_TRAP
} SNMP_TRAP_IP_ADDRESS_TYPE;

```

Description

This is type SNMP_TRAP_IP_ADDRESS_TYPE.

5.8.22.7.5.21 SNMP_VAL Union

C

```

typedef union {
    uint32_t dword;
    uint16_t word;
    uint8_t byte;
    uint8_t v[sizeof(uint32_t)];
} SNMP_VAL;

```

Description

This is type SNMP_VAL.

Members

Members	Description
uint32_t dword;	double word value
uint16_t word;	word value
uint8_t byte;	byte value
uint8_t v[sizeof(uint32_t)];	byte array

5.8.22.7.5.22 TCPIP_SNMP_DCPT Structure

C

```

typedef struct {
    TCPIP_SNMP_SM sm;
    UDP_SOCKET skt;
    bool readFromSnmpBuf;
    IPV6_HANDLE snmpIPV6Handler;
}

```

```
    IPV6_EVENT_TYPE ipv6EventType;  
} TCPIP_SNMP_DCPT;
```

Description

TCPIP_STACK_USE_IPV6

Members

Members	Description
TCPIP_SNMP_SM sm;	current status
UDP_SOCKET skt;	associated socket

5.8.22.7.5.23 TCPIP_SNMP_SM Enumeration

C

```
typedef enum {  
    SNMP_HOME = 0,  
    SNMP_LISTEN,  
    SNMP_PROCESS,  
    SNMP_PACKET_DISCARD  
} TCPIP_SNMP_SM;
```

Description

This is type TCPIP_SNMP_SM.

5.8.22.7.5.24 VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE Enumeration

C

```
typedef enum {  
    VENDOR_TRAP_DEFAULT = 0x0,  
    BUTTON_PUSH_EVENT = 0x1,  
    POT_READING_MORE_512 = 0x2  
} VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE;
```

Description

This is type VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE.

5.8.22.7.6 SNMPv3 Data Types and Constants

5.8.22.7.6.1 SNMPV3_PROCESSING_MEM_INFO_PTRS Structure

C

```
typedef struct {  
    SNMPV3_STACK_DCPT_STUB * snmpv3StkProcessingDynMemStubPtr;  
    const void* snmpHeapMemHandler;  
} SNMPV3_PROCESSING_MEM_INFO_PTRS;
```

Description

SNMPv3 Processing Memory Pointers

5.8.22.7.6.2 SNMPV3_STACK_DCPT_STUB Structure

C

```
typedef struct {  
    uint16_t UserInfoDataBaseIndx;  
    uint8_t SnmpEngineID[32];  
}
```

```

uint8_t SnmpEngnIDLength;
uint16_t SnmpMsgBufSeekPos;
uint16_t ScopedPduDataPos;
uint32_t SnmpEngineTimeOffset;
uint32_t SnmpEngineBoots;
TCPIP_UINT16_VAL UsmStatsEngineID;
TCPIP_UINT32_VAL AuthoritativeSnmpEngineBoots;
TCPIP_UINT32_VAL AuthoritativeSnmpEngnTime;
TCPIP_UINT32_VAL IncmngSnmpPduMsgID;
TCPIP_UINT32_VAL SnmpEngineTime;
TCPIP_UINT32_VAL SnmpEngnMaxMsgSize;
SNMPV3_REQUEST_WHOLEMSG InPduWholeMsgBuf;
SNMPV3_RESPONSE_WHOLEMSG OUTPduWholeMsgBuf;
SNMPV3_RESPONSE_WHOLEMSG TrapOUTPduWholeMsgBuf;
snmpV3TrapConfigDataBase Snmpv3TrapConfigData[SNMPV3_USM_MAX_USER];
SNMPV3MSGDATA ScopedPduRequestBuf;
SNMPV3MSGDATA ScopedPduRespnsBuf;
SNMPV3MSGDATA PduHeaderBuf;
SNMPV3MSGDATA TrapMsgHeaderBuf;
SNMPV3MSGDATA TrapScopdPduRespnsBuf;
dispatcherProcessPdu incomingPdu;
uint8_t SnmpSecurityLevel;
uint8_t SnmpRespnsSecrtyFlg;
uint8_t SnmpInMsgAuthParmStrng[12+1];
uint8_t SnmpInMsgAuthParamLen;
uint8_t snmpInMsgPrvParamStrng[8+1];
uint8_t SnmpInMsgPrivParamLen;
uint8_t SnmpOutMsgAuthParaStrng[12+1];
uint8_t SnmpOutMsgAuthParamLen;
uint8_t SnmpOutMsgPrvParamStrng[8+1];
uint8_t SnmpOutMsgPrivParamLen;
uint32_t SnmpEngnSecurityModel;
uint32_t SnmpEngnMsgProcessModel;
SecuritySysProcessIncomingMsg SecurtyPrimtivesOfIncmngPdu;
snmpV3EngnUserDataBase UserInfoDataBase[SNMPV3_USM_MAX_USER];
} SNMPV3_STACK_DCPT_STUB;

```

Description

SNMPv3 Descriptor Structure Typedef

Members

Members	Description
uint8_t SnmpEngineID[32];	Reserving 32 bytes for the snmpEngineID as the octet string length can vary from 5 to 32
uint32_t SnmpEngineBoots;	The number of times that the SNMP engine has (re-)initialized itself since snmpEngineID was last configured.
TCPIP_UINT32_VAL SnmpEngineTime;	The number of seconds since the value of the SnmpEngineBoots object last changed
snmpV3TrapConfigDataBase Snmpv3TrapConfigData[SNMPV3_USM_MAX_USER];	snmv3 global database for trap table
uint32_t SnmpEngnSecurityModel;	Maximum range (2 ³¹ -1), RFC3411
uint32_t SnmpEngnMsgProcessModel;	Maximum range (2 ³¹ -1), RFC3411

5.8.22.8 Files

Files

Name	Description
snmp.h	

snmp_config.h	SNMP configuration file
snmpv3.h	This is file snmpv3.h.
snmpv3_config.h	SNMPv3 configuration file

Description

5.8.22.8.1 snmp.h

SNMP Defs for Microchip TCP/IP Stack

Enumerations

	Name	Description
	GENERIC_TRAP_NOTIFICATION_TYPE	This is type GENERIC_TRAP_NOTIFICATION_TYPE.
	SNMP_COMMUNITY_TYPE	This is type SNMP_COMMUNITY_TYPE.
	SNMP_TRAP_IP_ADDRESS_TYPE	This is type SNMP_TRAP_IP_ADDRESS_TYPE.
	TCPIP_SNMP_SM	This is type TCPIP_SNMP_SM.
	VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE	This is type VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE.

Functions

	Name	Description
	SNMPGetExactIndex	To search for exact index node in case of a Sequence variable.
	SNMPGetNextIndex	To search for next index node in case of a Sequence variable.
	SNMPGetVar	Used to Get/collect OID variable information.
	SNMPIsValidSetLen	Validates the set variable data length to data type.
	SNMPSendTrap	Prepare, validate remote node which will receive trap and send trap pdu.
	SNMPSetVar	This routine Set the mib variable with the requested value.
	SNMPValidateCommunity	Validates community name for access control.
	TCPIP_SNMP_ClientGetNet	
	TCPIP_SNMP_DataCopyToProcessBuffer	Copies uint8_t data to dynamically allocated memory buffer.
	TCPIP_SNMP_EventNotifyGet	To Get the IPv6 DHCP event notification.
	TCPIP_SNMP_Notify	Creates and Sends TRAP pdu.
	TCPIP_SNMP_NotifyIsReady	Resolves given remoteHost IP address into MAC address.
	TCPIP_SNMP_NotifyPrepare	Collects trap notification info and send ARP to remote host.
	TCPIP_SNMP_PacketProcStubPtrsGet	Get SNMP packet processing memory pointer.
	TCPIP_SNMP_ProcessBufferDataGet	Reads uint8_t data from dynamically allocated memory buffer.
	TCPIP_SNMP_ReadCommunityGet	Get the readCommunity String with Snmp index.
	TCPIP_SNMP_TrapTimeGet	Get SNMP Trap UDP client open socket timeout.
	TCPIP_SNMP_WriteCommunityGet	Get the writeCommunity String with Snmp index.

Macros

	Name	Description
	SNMP_END_OF_VAR	This is macro SNMP_END_OF_VAR.

	SNMP_H	FileName: SNMP.h Copyright © 2012 released Microchip Technology Inc. All rights reserved. Microchip licenses to you the right to use, modify, copy and distribute Software only when embedded on a Microchip microcontroller or digital signal controller that is integrated into your product or third party product (pursuant to the sublicense terms in the accompanying license agreement). You should refer to the license agreement accompanying this Software for additional information regarding your rights and obligations. SOFTWARE AND DOCUMENTATION ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND... more
	SNMP_INDEX_INVALID	This is macro SNMP_INDEX_INVALID.
	SNMP_START_OF_VAR	
	SNMP_V1	
	SNMP_V2C	This is macro SNMP_V2C.
	SNMP_V3	This is macro SNMP_V3.

Structures

	Name	Description
	tIPV6_SNMP_TRAP_INFO	This is type IPV6_SNMP_TRAP_INFO.
	tSNMP_TRAP_INFO	This is type SNMP_TRAP_INFO.
	IPV6_SNMP_TRAP_INFO	This is type IPV6_SNMP_TRAP_INFO.
	SNMP_BUFFER_DATA	SNMPv1/v2c
	SNMP_MODULE_CONFIG	This is type SNMP_MODULE_CONFIG.
	SNMP_NON_MIB_RECD_INFO	This is type SNMP_NON_MIB_RECD_INFO.
	SNMP_NOTIFY_INFO	// the only if that runs SNMP
	SNMP_PROCESSING_MEM_INFO_PTRS	This is type SNMP_PROCESSING_MEM_INFO_PTRS.
	SNMP_STACK_DCPT_STUB	
	SNMP_TRAP_INFO	This is type SNMP_TRAP_INFO.
	TCPIP_SNMP_DCPT	TCPIP_STACK_USE_IPV6

Types

	Name	Description
	SNMP_ID	This is the SNMP OID variable id. This id is assigned via MIB file. Only dynamic and AgentID variables can contain ID. MIB2BIB utility enforces this rules when BIB was generated. typedef int SNMP_ID;
	SNMP_INDEX	This is type SNMP_INDEX.

Unions

	Name	Description
	SNMP_VAL	This is type SNMP_VAL.

5.8.22.8.2 snmp_config.h

Simple Network Management Protocol (SNMP) Configuration file

This file contains the SNMP module configuration options

Macros

	Name	Description
	END_OF_SNMP_READ_COMMUNITIES	??
	END_OF_SNMP_WRITE_COMMUNITIES	??
	MY_DEFAULT_SNMP_IF	For multi-homed hosts, the default SNMP interface
	NOTIFY_COMMUNITY_LEN	Length of Community name array
	OID_MAX_LEN	Change this to match your OID string length.
	SNMP_BIB_FILE_NAME	Name of the bib file for snmp
	SNMP_COMMUNITY_MAX_LEN	This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after SNMP_COMMUNITY_MAX_LEN for adding '0' NULL character.
	SNMP_MAX_COMMUNITY_SUPPORT	Specifying more strings than SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present.
	SNMP_MAX_MSG_SIZE	SNMP MIN and MAX message 484 bytes in size. As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 (section 4- protocol specification) and implementation supports more than 484 whenever feasible.
	SNMP_MAX_NON_REC_ID_OID	Update the Non record id OID value which is part of CustomSnmpDemo.c file
	SNMP_READ_COMMUNITIES	Default SNMPv2C community names. These can be overridden at run time if alternate strings are present in external EEPROM or Flash (actual strings could be stored by the TCPIP storage service. These strings are case sensitive. An empty string means disabled (not matchable). For application security, these default community names should not be used, but should all be disabled to force the end user to select unique community names. These defaults are provided only to make it easier to start development. Specifying more strings than SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying... more
	SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE	This macro will be used to avoid SNMP OID memory buffer corruption
	SNMP_WRITE_COMMUNITIES	Default SNMPv2C community names. See SNMP_READ_COMMUNITIES for more information.
	TRAP_COMMUNITY_MAX_LEN	Maximum length of community name table
	TRAP_TABLE_SIZE	This table maintains list of interested receivers who should receive notifications when some interesting event occurs.

Structures

	Name	Description
	SNMP_NET_CONFIG	SNMP Network Configuration structure typedef

5.8.22.8.3 snmpv3.h

This is file snmpv3.h.

Functions

	Name	Description
	SNMPv3ComputeHMACIpadOpadForAuthLoclzedKey	Compute HMAC inner and outer pad for authorization localized key.
	SNMPv3USMAuthPrivPswdLocalization	Convert Auth and Priv password to the localized Key using SNMPEngineID.
	TCPIP_SNMPv3_CmprTrapSecNameAndSecLvIWithUSMDb	Routine to find the index of the user name in the user data base table.
	TCPIP_SNMPv3_Notify	Creates and Sends SNMPv3 TRAP pdu.
	TCPIP_SNMPV3_PacketProcStubPtrsGet	Get SNMPv3 packet processing memory pointer.

Structures

	Name	Description
	SNMPV3_PROCESSING_MEM_INFO_PTRS	SNMPv3 Processing Memory Pointers
	SNMPV3_STACK_DCPT_STUB	SNMPv3 Descriptor Structure Typedef

5.8.22.8.4 snmpv3_config.h

Simple Network Management Protocol (SNMPv3) Configuration file

This file contains the SNMPv3 module configuration options

Macros

	Name	Description
	AUTH_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Authentication Localized passwed key lenegth size
	PRIV_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Privacy Pasword key length size
	SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 authentication localized Key length for memory validation
	SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 privacy key length size for memory validation
	SNMPV3_TRAP_MSG_PROCESS_MODEL_DB	This is macro SNMPV3_TRAP_MSG_PROCESS_MODEL_DB.
	SNMPV3_TRAP_SECURITY_LEVEL_DB	This is macro SNMPV3_TRAP_SECURITY_LEVEL_DB.
	SNMPV3_TRAP_SECURITY_MODEL_TYPE_DB	This is macro SNMPV3_TRAP_SECURITY_MODEL_TYPE_DB.
	SNMPV3_TRAP_USER_SECURITY_NAME_DB	This is macro SNMPV3_TRAP_USER_SECURITY_NAME_DB.
	SNMPV3_USER_AUTH_PASSWD_DB	This is macro SNMPV3_USER_AUTH_PASSWD_DB.
	SNMPV3_USER_AUTH_TYPE_DB	This is macro SNMPV3_USER_AUTH_TYPE_DB.
	SNMPV3_USER_PRIV_PASSWD_DB	This is macro SNMPV3_USER_PRIV_PASSWD_DB.
	SNMPV3_USER_PRIV_TYPE_DB	This is macro SNMPV3_USER_PRIV_TYPE_DB.

	SNMPV3_USER_SECURITY_NAME_DB	This is macro SNMPV3_USER_SECURITY_NAME_DB.
	SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE	User security name length for memory validation
	SNMPV3_USM_MAX_USER	Client configuration options
	USER_SECURITY_NAME_LEN	SNMPv3 User Security Name length

5.8.23 SNTP TCP/IP Stack Library

5.8.23.1 Introduction

Simple Network Time Protocol (SNTP) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the SNTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers. It then calculates reference times on any call to `SNTPGetUTCSeconds` (see page 458) using the internal Tick timer module.

The SNTP module is good for providing absolute time stamps. However, it should not be relied upon for measuring time differences (especially small differences). The pool of public time servers is implemented using round-robin DNS, so each update will come from a different server. Differing network delays and the fact that these servers are not verified implies that this time could be non-linear. While it is deemed reliable, it is not guaranteed to be accurate.

The Tick module provides much better accuracy (since it is driven by a hardware clock) and resolution, and should be used for measuring timeouts and other internal requirements.

Developers can change the value of [NTP_SERVER](#) (see page 462) if they wish to always point to a preferred time server, or to specify a region when accessing time servers. The default is to use the global pool.

5.8.23.2 Release Notes

MPLAB Harmony Version: v0.70b **SNTP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.23.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.23.4 Using the Library

This topic describes the basic architecture of the SNTP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [sntp.h](#)

The interface to the SNTP TCP/IP Stack library is defined in the "[sntp.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the SNTP TCP/IP Stack library should include "tcpip.h".

Library File:

The SNTP TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.23.4.1 Abstraction Model

This library provides the API of the SNTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SNTP Software Abstraction Block Diagram

Note: This image was not available at time of release and will be added in a future release of MPLAB Harmony.

5.8.23.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the SNTP module.

Library Interface Section	Description
Functions	Routines for configuring SNTP
Data Types and Constants	This section provides various definitions describing this API

5.8.23.5 Configuring the Library

Macros

	Name	Description
	NTP_DEFAULT_CONNECTION_TYPE	The default connection type to use: IPv4/IPv6
	NTP_DEFAULT_IF	for multi-homed hosts, the default SNTP interface
	NTP_EPOCH	Reference Epoch to use. (default: 01-Jan-1970 00:00:00)
	NTP_FAST_QUERY_INTERVAL	Defines how long to wait to retry an update after a failure, seconds. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.
	NTP_MAX_RETRIES	Number of retries to connect to the server
	NTP_MAX_STRATUM	The maximum acceptable NTP stratum number Should be less than 16 (unsynchronized server)
	NTP_QUERY_INTERVAL	Defines how frequently to resynchronize the date/time, seconds (default: 10 minutes)
	NTP_REPLY_TIMEOUT	Defines how long to wait before assuming the query has failed, seconds
	NTP_SERVER	These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography.
	NTP_SERVER_PORT	Port for contacting NTP servers
	NTP_TIME_STAMP_TMO	elapsed time that qualifies a time stamp as stale normally it should be correlated with NTP_QUERY_INTERVAL
	NTP_VERSION	The default NTP version to use (3 or 4)

Description

The configuration of the SNTP TCP/IP Stack Library is based on the file tcpip_config.h

This header file contains the configuration selection for the SNTP TCP/IP Stack Library. Based on the selections made, the SNTP TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the SNTP TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.23.5.1 NTP_DEFAULT_CONNECTION_TYPE Macro

C

```
#define NTP_DEFAULT_CONNECTION_TYPE (IP_ADDRESS_TYPE_IPV4)
```

Description

The default connection type to use: IPv4/IPv6

5.8.23.5.2 NTP_DEFAULT_IF Macro

C

```
#define NTP_DEFAULT_IF "PIC32INT"
```

Description

for multi-homed hosts, the default SNTP interface

5.8.23.5.3 NTP_EPOCH Macro

C

```
#define NTP_EPOCH (86400ul * (365ul * 70ul + 17ul))
```

Description

Reference Epoch to use. (default: 01-Jan-1970 00:00:00)

5.8.23.5.4 NTP_FAST_QUERY_INTERVAL Macro

C

```
#define NTP_FAST_QUERY_INTERVAL (14ul)
```

Description

Defines how long to wait to retry an update after a failure, seconds. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.

5.8.23.5.5 NTP_MAX_RETRIES Macro

C

```
#define NTP_MAX_RETRIES (3)
```

Description

Number of retries to [connect](#) to the server

5.8.23.5.6 NTP_MAX_STRATUM Macro

C

```
#define NTP_MAX_STRATUM (15)
```

Description

The maximum acceptable NTP stratum number Should be less than 16 (unsynchronized server)

5.8.23.5.7 NTP_QUERY_INTERVAL Macro

C

```
#define NTP_QUERY_INTERVAL (10ul*60ul)
```

Description

Defines how frequently to resynchronize the date/time, seconds (default: 10 minutes)

5.8.23.5.8 NTP_REPLY_TIMEOUT Macro

C

```
#define NTP_REPLY_TIMEOUT (6ul)
```

Description

Defines how long to wait before assuming the query has failed, seconds

5.8.23.5.9 NTP_SERVER Macro

C

```
#define NTP_SERVER "pool.ntp.org"
```

Description

These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography.

5.8.23.5.10 NTP_SERVER_PORT Macro

C

```
#define NTP_SERVER_PORT (123ul)
```

Description

Port for contacting NTP servers

5.8.23.5.11 NTP_TIME_STAMP_TMO Macro

C

```
#define NTP_TIME_STAMP_TMO (11 * 60)
```

Description

elapsed time that qualifies a time stamp as stale normally it should be correlated with [NTP_QUERY_INTERVAL](#)

5.8.23.5.12 NTP_VERSION Macro

C

```
#define NTP_VERSION (4)
```

Description

The default NTP version to use (3 or 4)

5.8.23.6 Building the Library

This section list the files that are available in the \src of the SNTP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.23.7 Library Interface

Data Types and Constants

	Name	Description
	SNTP_MODULE_CONFIG	Placeholder for SNTP Module Configuration
	SNTP_RESULT	Provides a list of possible results for the SNTP module.

Functions

	Name	Description
	TCPIP_SNTP_ConnectionInitiate	Forces a connection to the NTP server
	TCPIP_SNTP_ConnectionParamSet	Sets the current SNTP connection parameters
	TCPIP_SNTP_LastErrorGet	Gets the last error code set in the NTP server
	TCPIP_SNTP_TimeStampGet	Gets the last valid timestamp obtained from an NTP server
	TCPIP_SNTP_UTCSecondsGet	Obtains the current time from the SNTP module.

Description

This section describes the Application Programming Interface (API) functions of the SNTP TCP/IP Stack.

Refer to each section for a detailed description.

5.8.23.7.1 Functions

5.8.23.7.1.1 TCPIP_SNTP_ConnectionInitiate Function

C

```
SNTP_RESULT TCPIP_SNTP_ConnectionInitiate();
```

Description

This function will start a connection to the NTP server

Preconditions

TCP/IP Stack should have been initialized

Returns

SNTP_RES_OK - if the call succeeded SNTP_RES_BUSY error code if the connection could not have been started

Remarks

None

5.8.23.7.1.2 TCPIP_SNTP_ConnectionParamSet Function

C

```
SNTP_RESULT TCPIP_SNTP_ConnectionParamSet(
    TCPIP_NET_HANDLE netH,
    IP_ADDRESS_TYPE ntpConnType
);
```

Description

This function sets the parameters for the next SNTP connections.

Preconditions

TCP/IP Stack should have been initialized

Parameters

Parameters	Description
neth	new interface to use as default SNTP interface if 0, the current interface is not changed
ntpConnType	type of conenction to make: IPv4 or IPv6 if IP_ADDRESS_TYPE_ANY, the current setting is not changed

Returns

SNTP_RES_OK - if the call succeeded [SNTP_RESULT](#) error code otherwise

Remarks

None

5.8.23.7.1.3 TCPIP_SNTP_LastErrorGet Function**C**

```
SNTP_RESULT TCPIP_SNTP_LastErrorGet();
```

Description

This function returns the last NTP error code and clears the current error code

Preconditions

TCP/IP Stack should have been initialized

Returns

the last error code encountered by the NTP module

Remarks

None

5.8.23.7.1.4 TCPIP_SNTP_TimeStampGet Function**C**

```
SNTP_RESULT TCPIP_SNTP_TimeStampGet(  
    uint64_t* pTStamp,  
    SYS_TICK* pLastUpdate  
);
```

Description

This function gets the last valid timestamp obtained from an NTP server

Preconditions

TCP/IP Stack should have been initialized

Parameters

Parameters	Description
pTStamp	pointer to a 64 bit buffer to store the last NTP timestamp could be NULL if the timestamp not needed
pLastUpdate	pointer to store the last time stamp update tick could be NULL if the update time not needed

Returns

SNTP_RES_OK - if the call succeeded SNTP_RES_TSTAMP_STALE error code if there's no recent time stamp

Remarks

None

5.8.23.7.1.5 TCPIP_Sntp_UTCSecondsGet Function

C

```
uint32_t TCPIP_Sntp_UTCSecondsGet();
```

Description

This function obtains the current time as reported by the SNTP module. Use this value for absolute time stamping. The value returned is (by default) the number of seconds since 01-Jan-1970 00:00:00.

Preconditions

TCP/IP Stack should have been initialized

Returns

The number of seconds since the Epoch. (Default 01-Jan-1970 00:00:00)

Remarks

Do not use this function for time difference measurements. The Tick module is more appropriate for those requirements.

5.8.23.7.2 Data Types and Constants

5.8.23.7.2.1 SNTP_MODULE_CONFIG Structure

C

```
typedef struct {  
} SNTP_MODULE_CONFIG;
```

Description

Placeholder for SNTP Module Configuration

5.8.23.7.2.2 SNTP_RESULT Enumeration

C

```
typedef enum {  
    SNTP_RES_OK,  
    SNTP_RES_PROGRESS,  
    SNTP_RES_BUSY = -1,  
    SNTP_RES_TSTAMP_STALE = -2,  
    SNTP_RES_SKT_ERR = -3,  
    SNTP_RES_NTP_SERVER_TMO = -4,  
    SNTP_RES_NTP_VERSION_ERR = -5,  
    SNTP_RES_NTP_TSTAMP_ERR = -6,  
    SNTP_RES_NTP_SYNC_ERR = -7,  
    SNTP_RES_NTP_KOD_ERR = -8,  
    SNTP_RES_NTP_DNS_ERR = -9  
} SNTP_RESULT;
```

Description

SNTP_RESULT Enumeration

Provides a list of possible SNTP results.

Members

Members	Description
SNTP_RES_OK	the operation was successful
SNTP_RES_PROGRESS	an NTP operation is in progress
SNTP_RES_BUSY = -1	module is busy
SNTP_RES_TSTAMP_STALE = -2	timestamp is stale, there's no recent timestamp
SNTP_RES_SKT_ERR = -3	NTP socket could not be opened
SNTP_RES_NTP_SERVER_TMO = -4	NTP server could not be accessed
SNTP_RES_NTP_VERSION_ERR = -5	wrong NTP version received
SNTP_RES_NTP_TSTAMP_ERR = -6	wrong NTP time stamp received
SNTP_RES_NTP_SYNC_ERR = -7	NTP time synchronization error
SNTP_RES_NTP_KOD_ERR = -8	an NTP KissOfDeath code has been received
SNTP_RES_NTP_DNS_ERR = -9	an NTP DNS error

Remarks

None

5.8.23.8 Files

Files

Name	Description
sntp.h	
sntp_config.h	SNTP configuration file

Description

5.8.23.8.1 sntp.h

SNTP Client Module Header

Enumerations

	Name	Description
	SNTP_RESULT	Provides a list of possible results for the SNTP module.

Functions

	Name	Description
	TCPIP_SNTP_ConnectionInitiate	Forces a connection to the NTP server
	TCPIP_SNTP_ConnectionParamSet	Sets the current SNTP connection parameters
	TCPIP_SNTP_LastErrorGet	Gets the last error code set in the NTP server
	TCPIP_SNTP_TimeStampGet	Gets the last valid timestamp obtained from an NTP server
	TCPIP_SNTP_UTCSecondsGet	Obtains the current time from the SNTP module.

Structures

	Name	Description
	<u>SNTP_MODULE_CONFIG</u>	Placeholder for SNTP Module Configuration

5.8.23.8.2 sntp_config.h

Network Time Protocol (SNTP) Configuration file

This file contains the SNTP module configuration options

Macros

	Name	Description
	<u>NTP_DEFAULT_CONNECTION_TYPE</u>	The default connection type to use: IPv4/IPv6
	<u>NTP_DEFAULT_IF</u>	for multi-homed hosts, the default SNTP interface
	<u>NTP_EPOCH</u>	Reference Epoch to use. (default: 01-Jan-1970 00:00:00)
	<u>NTP_FAST_QUERY_INTERVAL</u>	Defines how long to wait to retry an update after a failure, seconds. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.
	<u>NTP_MAX_RETRIES</u>	Number of retries to <u>connect</u> to the server
	<u>NTP_MAX_STRATUM</u>	The maximum acceptable NTP stratum number Should be less than 16 (unsynchronized server)
	<u>NTP_QUERY_INTERVAL</u>	Defines how frequently to resynchronize the date/time, seconds (default: 10 minutes)
	<u>NTP_REPLY_TIMEOUT</u>	Defines how long to wait before assuming the query has failed, seconds
	<u>NTP_SERVER</u>	These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography.
	<u>NTP_SERVER_PORT</u>	Port for contacting NTP servers
	<u>NTP_TIME_STAMP_TMO</u>	elapsed time that qualifies a time stamp as stale normally it should be correlated with <u>NTP_QUERY_INTERVAL</u>
	<u>NTP_VERSION</u>	The default NTP version to use (3 or 4)

5.8.24 SSL TCP/IP Stack Library**5.8.24.1 Introduction**

Secure Sockets Layer (SSL) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the SSL TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The SSL module adds encryption support to the TCP layer by implementing the SSLv3 protocol. This protocol is the standard for secure communications across the Internet, and prevents snooping or tampering of data as it travels across an untrusted network. This implementation of SSL supports the RSA asymmetric encryption protocol and the ARCFOUR symmetric encryption protocol.

To comply with US Export Control restrictions, the encryption portion of the SSL module must be purchased separately from Microchip. The library of Data Encryption Routines (SW300052) is available for a nominal fee from microchipDirect.

Newer versions of the TCP/IP Stack include modifications to the SSL module to enable an expanded range of RSA key lengths. This will allow the TCP/IP Stack to support a wider range of SSL client/servers. Because of these changes, you may need to manually modify your copy of RSA.c or obtain an updated copy from microchipDirect. The following table enumerates which cryptographic add-on is required for which stack version.

5.8.24.2 Release Notes

MPLAB Harmony Version: v0.70b **SSL TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.24.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.24.4 Using the Library

This topic describes the basic architecture of the SSL TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [ssl.h](#)

The interface to the SSL TCP/IP Stack library is defined in the "[ssl.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the SSL TCP/IP Stack library should include "tcpip.h".

Library File:

The SSL TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.24.4.1 Abstraction Model

This library provides the API of the SSL TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SSL Software Abstraction Block Diagram

Note: This image was not available at time of release and will be added in a future release of MPLAB Harmony.

5.8.24.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the SSL module.

Library Interface Section	Description
Functions	Routines for configuring SSL
Data Types and Constants	This section provides various definitions describing this API

5.8.24.4.3 How the Library Works

5.8.24.4.3.1 SSL Client Support

An SSL client can be initiated by first opening a TCP connection, then calling TCPStartSSLSession to initiate the SSL handshake process. The handshake uses the public key from the certificate provided by the server. Key lengths up to 1024 bits are supported on all processors; key lengths up to 2048 bits are supported on PIC32 microcontrollers. The [SSL_RSA_CLIENT_SIZE](#) macro in SSLClientSize.h sets the maximum certificate key length that your client should process.

```
#define SSL_RSA_CLIENT_SIZE (1024u1) // Size of Encryption Buffer (must be larger than key size)
```

Once the handshake has started, call TCPSSLIsHandshaking (see page 468) until it returns FALSE. This will indicate that the handshake has completed and all traffic is now secured using 128-bit ARCFOUR encryption. If the handshake fails for any reason, the TCP connection will automatically be terminated as required by the SSL protocol specification.

For faster performance, the SSL module caches security parameters for the most recently made connections. This allows quick reconnections to the same node without the computational expense of another RSA handshake. By default, the two most recent connections are cached, but this can be modified in TCPIPConfig.h.

SSL client support is already enabled for SMTP. When STACK_USE_SSL_CLIENT is defined, the SMTP module automatically adds a field to SMTPClient (see page 304) called UseSSL. That field controls whether or not the SMTP client module will attempt to make an SSL connection before transmitting any data.

Note that TCP sockets using SSL may require an increase in TX/RX buffer size to support SSL. You can adjust the size of your sockets using the TCP/IP Configuration Utility included with the stack.

5.8.24.4.3.2 SSL Server Support

To initiate an SSL server, first open a TCP [socket](#) for listening using TCPOpen (see page 542). Then call TCPAddSSLListener (see page 468) to [listen](#) (see page 175) for incoming SSL connections on an alternate port. This allows a single [socket](#) to share application-level resources and [listen](#) (see page 175) for connections on two different ports. Connections occurring on the originally opened port will proceed unsecured, while connections on the SSL port will first complete an SSL handshake to secure the data.

If your application will not [accept](#) (see page 169) unsecured traffic, simply open a non-secured [socket](#) on a free port number, then verify that each incoming connection is secured (not on that port) by calling TCPIsSSL (see page 469).

SSL server support is automatically enabled for HTTP2 when STACK_USE_SSL_SERVER is defined. By default, the HTTP2 module will then [listen](#) (see page 175) for unsecured traffic on port 80 and secured connections on port 443.

This SSL server implementation supports key lengths up to 1024 bits on most PIC microcontrollers, and 2048 bits on PIC32 microcontrollers. The [SSL_RSA_KEY_SIZE](#) (see page 471) macro in TCPIPConfig.h sets the server certificate key length.

```
// Bits in SSL RSA key. This parameter is used for SSL sever  
// connections only.  
#define SSL_RSA_KEY_SIZE (512u1)
```

Note that TCP sockets using SSL may require an increase in TX/RX buffer size to support SSL. You can adjust the size of your sockets using the TCP/IP Configuration Utility included with the stack.

5.8.24.4.3.3 SSL Limitations

SSL was designed for desktop PCs with faster processors and significantly more resources than are available on an embedded platform. A few compromises must be made in order to use SSL in a less resource-intensive manner.

The SSL client module does not perform any validation or verification of certificates. Doing so would require many root certificates to be stored locally for verification, which is not feasible for memory-limited parts. This does not compromise security once the connection has been established, but does not provide complete security against man-in-the-middle attacks. (This sort of attack is uncommon and would be difficult to execute.)

Neither the SSL client nor the server can completely verify MACs before processing data. SSL records include a signature to verify that messages were not modified in transit. This Message Authentication (see page 89) Code, or MAC, is inserted after at least every 16kB of traffic. (It usually is inserted much more frequently than that.) Without 16kB of RAM to buffer packets for each [socket](#), incoming data must be handed to the application layer before the MAC can be completely verified. Invalid MACs will still cause the connection to terminate immediately, but by the time this is detected some bad data may have already reached the application. Since the ARCFOUR cipher in use is a stream cipher, it would be difficult to exploit this in any meaningful way. An attacker would not be able to control what data is actually modified or inserted, as doing so without knowledge of the key would yield garbage. However, it is important to understand that incoming data is not completely verified before being passed to the application.

5.8.24.4.4 Generating Server Certificates

The SSL certificates used by the TCP/IP Stack's SSL module are stored in the CustomSSLCert.c source file. The following series of steps describe how to create the structures in CustomSSLCert.c using an SSL certificate.

1. Download and install the OpenSSL library. There are several third-party sites that offer SSL installers (e.g. <http://www.slproweb.com/products/Win32OpenSSL.html>). Note that some distributions may not include all commands specified by the OpenSSL documentation.

2. Open a console and change directory to the **OpenSSL/bin** folder.
3. If you don't have a key and certificate, you can generate them first. The following example console commands will generate a 512-bit key:
 1. Generate the key: **openssl genrsa -out 512bits.key 512**
 2. Generate the Certificate Signing Request (CSR). You will need to add additional information when prompted: **openssl req -new -key 512bits.key -out 512bits.csr**
 3. Generate the X.509 certificate if self-signing (or **send** the CSR to a Certificate Authority for signing): **openssl x509 -req -days 365 -in 512bits.csr -signkey 512bits.key -out 512bits.crt** (note that if the -days option is not specified, the default expiration time is 30 days)
 4. For additional documentation, refer to <http://www.openssl.org/docs/apps/openssl.html>.

4. Parse your key file using the command: **openssl.exe asn1parse -in "[directory containing your key]\512bits.key"**

5. You should see a screen like this:

6. If you are not using an ENCX24J600 family device, then the last 5 integers displayed here are the SSL_P, SSL_Q, SSL_dP, SSL_dQ, and SSL_qInv parameters, respectively. However, they are displayed here in big-endian format, and the Microchip cryptographic library implementation requires parameters in little-endian format, so you will have to enter the parameters into the C arrays in opposite order. For example, the INTEGER at offset 145:

```
145:d=1 h1=2 ;= 33 prim: INTEGER
```

```
:D777566780029FCD610200B66D89507D
```

```
915E3E5BDB6FAB0233B5DFA2E4081DF7
```

will be swapped in the CustomSSLCert.c file:

```
ROM BYTE SSL_P[] = {
```

```
0xF7 , 0x1D, 0x08, 0xE4, 0xA2, 0xDF, 0xB5, 0x33,
```

```
0x02, 0xAB, 0x6F, 0xDB, 0x5B, 0x3E, 0x5E, 0x91,
```

```
0x7D, 0x50, 0x89, 0x6D, 0xB6, 0x00, 0x02, 0x61,
```

```
0xCD, 0x9F, 0x02, 0x80, 0x67, 0x56, 0x77, 0xD7
```

```
};
```

7. If you are using an ENCX24J600 family device, then the second and fourth integers displayed here are the SSL_N and SSL_D parameters, respectively. There is no need to do an endian format change for these parameters. For the example, the expected SSL_N and SSL_D values are shown in the figure below:

8. Parse your X.509 certificate using the command: **openssl.exe asn1parse -in "[directory containing your cert]\512bits.crt" -out cert.bin**

9. Open the cert.bin output file in a hex editor. For example, here is the default certificate information generated from 512bits.crt given in the stack:

10. This information must be copied verbatim into the SSL_CERT [] array. Note that this is binary data (not a large integer) so it does not get endian-swapped like the private key parameters.

```
ROM BYTE SSL_CERT[524] = {
```

```
0x30, 0x82, 0x02, 0x08, 0x30, 0x82, 0x01, 0xb2, 0x02, 0x09, 0x00, 0xa5, 0x6a, 0xea, 0x1a,
```

```
0xa9, 0x52, 0x9d, 0x1e, 0x30, 0x0d, 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01,
```

```
0x05, 0x05, 0x00, 0x30, 0x81, 0x8a, 0x31, 0x0b, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13,
```

```
0x02, 0x55, 0x53, 0x31, 0x10, 0x30, 0x0e, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x07, 0x41, 0x72,
```

```
0x69, 0x7a, 0x6f, 0x6e, 0x61, 0x31, 0x11, 0x30, 0x0f, 0x06, 0x03, 0x55, 0x04, 0x07, 0x13, 0x08,
```

```
0x43, 0x68, 0x61, 0x6e, 0x64, 0x6c, 0x65, 0x72, 0x31, 0x23, 0x30, 0x21, 0x06, 0x03, 0x55, 0x04,
```

```

0x0a,
0x13, 0x1a, 0x4d, 0x69, 0x63, 0x72, 0x6f, 0x63, 0x68, 0x69, 0x70, 0x20, 0x54, 0x65, 0x63,
0x68,
0x6e, 0x6f, 0x6c, 0x6f, 0x67, 0x79, 0x2c, 0x20, 0x49, 0x6e, 0x63, 0x2e, 0x31, 0x1d, 0x30,
0x1b,
0x06, 0x03, 0x55, 0x04, 0x0b, 0x13, 0x14, 0x53, 0x53, 0x4c, 0x20, 0x44, 0x65, 0x6d, 0x6f,
0x20,
0x43, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x65, 0x31, 0x12, 0x30, 0x10,
0x06,
0x03, 0x55, 0x04, 0x03, 0x13, 0x09, 0x6d, 0x63, 0x68, 0x70, 0x62, 0x6f, 0x61, 0x72, 0x64,
0x30,
0x1e, 0x17, 0x0d, 0x30, 0x37, 0x31, 0x30, 0x30, 0x39, 0x31, 0x38, 0x33, 0x37, 0x32, 0x37,
0x5a,
0x17, 0x0d, 0x31, 0x37, 0x31, 0x30, 0x30, 0x36, 0x31, 0x38, 0x33, 0x37, 0x32, 0x37, 0x5a,
0x30,
0x81, 0x8a, 0x31, 0x0b, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x55, 0x53,
0x31,
0x10, 0x30, 0x0e, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x07, 0x41, 0x72, 0x69, 0x7a, 0x6f,
0x6e,
0x61, 0x31, 0x11, 0x30, 0x0f, 0x06, 0x03, 0x55, 0x04, 0x07, 0x13, 0x08, 0x43, 0x68, 0x61,
0x6e,
0x64, 0x6c, 0x65, 0x72, 0x31, 0x23, 0x30, 0x21, 0x06, 0x03, 0x55, 0x04, 0x0a, 0x13, 0x1a,
0x4d,
0x69, 0x63, 0x72, 0x6f, 0x63, 0x68, 0x69, 0x70, 0x20, 0x54, 0x65, 0x63, 0x68, 0x6e, 0x6f,
0x6c,
0x6f, 0x67, 0x79, 0x2c, 0x20, 0x49, 0x6e, 0x63, 0x2e, 0x31, 0x1d, 0x30, 0x1b, 0x06, 0x03,
0x55,
0x04, 0x0b, 0x13, 0x14, 0x53, 0x53, 0x4c, 0x20, 0x44, 0x65, 0x6d, 0x6f, 0x20, 0x43, 0x65,
0x72,
0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x65, 0x31, 0x12, 0x30, 0x10, 0x06, 0x03, 0x55,
0x04,
0x03, 0x13, 0x09, 0x6d, 0x63, 0x68, 0x70, 0x62, 0x6f, 0x61, 0x72, 0x64, 0x30, 0x5c, 0x30,
0x0d,
0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x01, 0x05, 0x00, 0x03, 0x4b,
0x00,
0x30, 0x48, 0x02, 0x41, 0x00, 0xaa, 0x96, 0xca, 0x97, 0xea, 0x27, 0xb0, 0xd7, 0xe9, 0x21,
0xd0,
0x40, 0xd4, 0x2c, 0x09, 0x5a, 0x2e, 0x3a, 0xe4, 0x12, 0x64, 0x2d, 0x4b, 0x1b, 0x92, 0xdf,
0x79,
0x68, 0x4e, 0x3c, 0x51, 0xf4, 0x43, 0x48, 0x0d, 0xf2, 0xc8, 0x50, 0x9b, 0x6e, 0xe5, 0xea,
0xfe,
0xef, 0xd9, 0x10, 0x41, 0x08, 0x14, 0xf9, 0x85, 0x49, 0xfc, 0x50, 0xd3, 0x57, 0x34, 0xdc,
0x3a,
0x0d, 0x79, 0xf8, 0xd3, 0x99, 0x02, 0x03, 0x01, 0x00, 0x01, 0x30, 0x0d, 0x06, 0x09, 0x2a,
0x86,
0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x05, 0x05, 0x00, 0x03, 0x41, 0x00, 0x18, 0x18, 0xfe,
0x8b,
0x2d, 0x0d, 0xf7, 0x0d, 0x65, 0x9d, 0x29, 0xec, 0xb3, 0x51, 0x6e, 0x3b, 0x93, 0xbb, 0x40,
0x1a,
0x0b, 0x34, 0x07, 0x63, 0x5e, 0x6a, 0x1c, 0x74, 0x59, 0xd4, 0x54, 0xd2, 0x1b, 0xf3, 0x31,
0xb7,
0x57, 0x4b, 0xa5, 0xe6, 0xe2, 0x35, 0xf7, 0xb3, 0x6a, 0x15, 0x6e, 0x3c, 0x93, 0x85, 0xb2,
0xca,
0xf5, 0x35, 0x00, 0xf4, 0x49, 0xe7, 0x00, 0x8a, 0x00, 0xd8, 0xe8, 0xcf
};

```

11. Update the SSL_CERT_LEN variable to contain the correct value.

5.8.24.5 Configuring the Library

Macros

	Name	Description
	<u>SSL_MAX_BUFFERS</u>	Max # of SSL buffers (2 per <u>socket</u>)

	SSL_MAX_CONNECTIONS	Maximum number of simultaneous connections via SSL
	SSL_MAX_HASHES	Max # of SSL hashes (2 per socket , plus 1 to avoid deadlock)
	SSL_MAX_SESSIONS	Max number of cached SSL sessions
	SSL_MIN_SESSION_LIFETIME	Minimum lifetime for SSL Sessions Sessions cannot be reallocated until this much time (seconds) has elapsed
	SSL_MULTIPLE_INTERFACES	SSL on all interfaces in a multi-homed host set to 0 to work only on one interface
	SSL_RSA_CLIENT_SIZE	Size of encryption buffer, must be at least as big as the key size
	SSL_RSA_KEY_SIZE	Bits in SSL RSA key. This parameter is used for SSL sever connections only. The only valid value is 512 bits (768 and 1024 bits do not work at this time). Note, however, that SSL client operations do currently work up to 1024 bit RSA key length.
	SSL_RSA_LIFETIME_EXTENSION	Lifetime extension for RSA operations Sessions lifetime is extended by this amount (seconds) when an RSA calculation is made
	SSL_VERSION	Moved from MicrochipIncludeTCPIP StackSSL.h
	SSL_VERSION_HI	SSL version number (high byte)
	SSL_VERSION_LO	SSL version number (low byte)

Description

The configuration of the SSL TCP/IP Stack Library is based on the file `tcPIP_config.h`

This header file contains the configuration selection for the SSL TCP/IP Stack Library. Based on the selections made, the SSL TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the TCP/IP Stack SSL Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.24.5.1 SSL_MAX_BUFFERS Macro**C**

```
#define SSL_MAX_BUFFERS (4)
```

Description

Max # of SSL buffers (2 per [socket](#))

5.8.24.5.2 SSL_MAX_CONNECTIONS Macro**C**

```
#define SSL_MAX_CONNECTIONS (2)
```

Description

Maximum number of simultaneous connections via SSL

5.8.24.5.3 SSL_MAX_HASHES Macro**C**

```
#define SSL_MAX_HASHES (5)
```

Description

Max # of SSL hashes (2 per [socket](#), plus 1 to avoid deadlock)

5.8.24.5.4 SSL_MAX_SESSIONS Macro

C

```
#define SSL_MAX_SESSIONS (2)
```

Description

Max number of cached SSL sessions

5.8.24.5.5 SSL_MIN_SESSION_LIFETIME Macro

C

```
#define SSL_MIN_SESSION_LIFETIME (1ul)
```

Description

Minimum lifetime for SSL Sessions Sessions cannot be reallocated until this much time (seconds) has elapsed

5.8.24.5.6 SSL_MULTIPLE_INTERFACES Macro

C

```
#define SSL_MULTIPLE_INTERFACES 1
```

Description

SSL on all interfaces in a multi-homed host set to 0 to work only on one interface

5.8.24.5.7 SSL_RSA_CLIENT_SIZE Macro

C

```
#define SSL_RSA_CLIENT_SIZE (1024ul)
```

Description

Size of encryption buffer, must be at least as big as the key size

5.8.24.5.8 SSL_RSA_KEY_SIZE Macro

C

```
#define SSL_RSA_KEY_SIZE (1024ul)
```

Description

Bits in SSL RSA key. This parameter is used for SSL sever connections only. The only valid value is 512 bits (768 and 1024 bits do not work at this time). Note, however, that SSL client operations do currently work up to 1024 bit RSA key length.

5.8.24.5.9 SSL_RSA_LIFETIME_EXTENSION Macro

C

```
#define SSL_RSA_LIFETIME_EXTENSION (8ul)
```

Description

Lifetime extension for RSA operations Sessions lifetime is extended by this amount (seconds) when an RSA calculation is made

5.8.24.5.10 SSL_VERSION Macro

C

```
#define SSL_VERSION (0x0300u) // Moved from Microchip\Include\TCPIP Stack\SSL.h
```

Description

Moved from Microchip\Include\TCPIP Stack\SSL.h

5.8.24.5.11 SSL_VERSION_HI Macro

C

```
#define SSL_VERSION_HI (0x03u)
```

Description

SSL version number (high byte)

5.8.24.5.12 SSL_VERSION_LO Macro

C

```
#define SSL_VERSION_LO (0x00u)
```

Description

SSL version number (low byte)

5.8.24.6 Building the Library

This section list the files that are available in the \src of the SSL driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.24.7 Library Interface

Data Types and Constants

	Name	Description
	SSL_MODULE_CONFIG	SSL layer configuration/initialization

Functions

	Name	Description
	TCPIP_TCP_SocketIsSecuredBySSL	Determines if a TCP connection is secured with SSL.
	TCPIP_TCPSSL_ClientBegin	Begins an SSL client session.
	TCPIP_TCPSSL_ClientStart	Begins an SSL client session.
	TCPIP_TCPSSL_HandshakeClear	Clears the SSL handshake flag.
	TCPIP_TCPSSL_ListenerAdd	Listens for SSL connection on a specific port.
	TCPIP_TCPSSL_ServerStart	Begins an SSL server session.
	TCPIP_TCPSSL_StillHandshaking	Determines if an SSL session is still handshaking.

Description

This section describes the Application Programming Interface (API) functions of the SSL TCP/IP Stack.

Refer to each section for a detailed description.

5.8.24.7.1 Functions

5.8.24.7.1.1 TCPIP_TCP_SocketIsSecuredBySSL Function

C

```
bool TCPIP_TCP_SocketIsSecuredBySSL(  
    TCP_SOCKET hTCP  
);
```

Description

Call this function to determine whether or not a TCP connection is secured with SSL.

Preconditions

TCP is initialized and hTCP is connected.

Parameters

Parameters	Description
hTCP	TCP connection to check

Return Values

Return Values	Description
true	Connection is secured via SSL
false	Connection is not secured

5.8.24.7.1.2 TCPIP_TCPSSL_ClientBegin Function

C

```
bool TCPIP_TCPSSL_ClientBegin(  
    TCP_SOCKET hTCP,  
    uint8_t* host,  
    void * buffer,  
    uint8_t suppDataType  
);
```

Description

This function escalates the current connection to an SSL secured connection by initiating an SSL client handshake.

Preconditions

TCP is initialized and hTCP is already connected.

Parameters

Parameters	Description
hTCP	TCP connection to secure
host	Expected host name on certificate (currently ignored)
buffer	Buffer for supplementary data return
suppDataType	Type of supplementary data to copy

Return Values

Return Values	Description
true	an SSL connection was initiated
false	Insufficient SSL resources (stubs) were available

Remarks

The host parameter is currently ignored and is not validated.

5.8.24.7.1.3 TCPIP_TCPSSL_ClientStart Function**C**

```
bool TCPIP_TCPSSL_ClientStart(  
    TCP_SOCKET hTCP,  
    uint8_t* host  
);
```

Description

This function escalates the current connection to an SSL secured connection by initiating an SSL client handshake.

Preconditions

TCP is initialized and hTCP is already connected.

Parameters

Parameters	Description
hTCP	TCP connection to secure
host	Expected host name on certificate (currently ignored)

Return Values

Return Values	Description
true	an SSL connection was initiated
false	Insufficient SSL resources (stubs) were available

Remarks

The host parameter is currently ignored and is not validated.

5.8.24.7.1.4 TCPIP_TCPSSL_HandshakeClear Function**C**

```
void TCPIP_TCPSSL_HandshakeClear(  
    TCP_SOCKET hTCP  
);
```

Description

This function clears the flag indicating that an SSL handshake is complete.

Preconditions

TCP is initialized and hTCP is connected.

Parameters

Parameters	Description
hTCP	TCP connection to set

Returns

None

Remarks

This function should never be called by an application. It is used only by the SSL module itself.

5.8.24.7.1.5 TCPIP_TCPSSL_ListenerAdd Function**C**

```
bool TCPIP_TCPSSL_ListenerAdd(  
    TCP_SOCKET hTCP,  
    uint16_t port  
);
```

Description

This function adds an additional listening port to a TCP connection. Connections made on this alternate port will be secured via SSL.

Preconditions

TCP is initialized and hTCP is listening.

Parameters

Parameters	Description
hTCP	TCP connection to secure
port	SSL port to listen on

Return Values

Return Values	Description
true	SSL port was added.
false	The socket was not a listening socket .

5.8.24.7.1.6 TCPIP_TCPSSL_ServerStart Function**C**

```
bool TCPIP_TCPSSL_ServerStart(  
    TCP_SOCKET hTCP  
);
```

Description

This function sets up an SSL server session when a new connection is established on an SSL port.

Preconditions

TCP is initialized and hTCP is already connected.

Parameters

Parameters	Description
hTCP	TCP connection to secure

Return Values

Return Values	Description
true	an SSL connection was initiated
false	Insufficient SSL resources (stubs) were available

5.8.24.7.1.7 TCPIP_TCPSSL_StillHandshaking Function

C

```
bool TCPIP_TCPSSL_StillHandshaking(  
    TCP_SOCKET hTCP  
);
```

Description

Call this function after calling [TCPIP_TCPSSL_ClientStart](#) until false is returned. Then your application may continue with its normal data transfer (which is now secured).

Preconditions

TCP is initialized and hTCP is connected.

Parameters

Parameters	Description
hTCP	TCP connection to check

Return Values

Return Values	Description
true	SSL handshake is still progressing
false	SSL handshake has completed

5.8.24.7.2 Data Types and Constants

5.8.24.7.2.1 SSL_MODULE_CONFIG Structure

C

```
typedef struct {  
} SSL_MODULE_CONFIG;
```

Description

SSL layer configuration/initialization

5.8.24.8 Files

Files

Name	Description
ssl.h	
ssl_config.h	SSL configuration file

Description

5.8.24.8.1 ssl.h

SSLv3 Module Headers

Functions

	Name	Description
	TCPIP_TCP_SocketIsSecuredBySSL	Determines if a TCP connection is secured with SSL.
	TCPIP_TCPSSL_ClientBegin	Begins an SSL client session.
	TCPIP_TCPSSL_ClientStart	Begins an SSL client session.
	TCPIP_TCPSSL_HandshakeClear	Clears the SSL handshake flag.
	TCPIP_TCPSSL_ListenerAdd	Listens for SSL connection on a specific port.
	TCPIP_TCPSSL_ServerStart	Begins an SSL server session.
	TCPIP_TCPSSL_StillHandshaking	Determines if an SSL session is still handshaking.

Structures

	Name	Description
	SSL_MODULE_CONFIG	SSL layer configuration/initialization

5.8.24.8.2 ssl_config.h

Secure Sockets Layer (SSL) Configuration file

This file contains the SSL module configuration options

Macros

	Name	Description
	SSL_MAX_BUFFERS	Max # of SSL buffers (2 per socket)
	SSL_MAX_CONNECTIONS	Maximum number of simultaneous connections via SSL
	SSL_MAX_HASHES	Max # of SSL hashes (2 per socket , plus 1 to avoid deadlock)
	SSL_MAX_SESSIONS	Max number of cached SSL sessions
	SSL_MIN_SESSION_LIFETIME	Minimum lifetime for SSL Sessions Sessions cannot be reallocated until this much time (seconds) has elapsed
	SSL_MULTIPLE_INTERFACES	SSL on all interfaces in a multi-homed host set to 0 to work only on one interface
	SSL_RSA_CLIENT_SIZE	Size of encryption buffer, must be at least as big as the key size
	SSL_RSA_KEY_SIZE	Bits in SSL RSA key. This parameter is used for SSL sever connections only. The only valid value is 512 bits (768 and 1024 bits do not work at this time). Note, however, that SSL client operations do currently work up to 1024 bit RSA key length.
	SSL_RSA_LIFETIME_EXTENSION	Lifetime extension for RSA operations Sessions lifetime is extended by this amount (seconds) when an RSA calculation is made
	SSL_VERSION	Moved from MicrochipIncludeTCPIP StackSSL.h
	SSL_VERSION_HI	SSL version number (high byte)
	SSL_VERSION_LO	SSL version number (low byte)

5.8.25 TCP TCP/IP Stack Library

5.8.25.1 Introduction

Transmission Control Protocol (TCP) TCP/IP Stack Library for Microchip Microcontrollers

This library provides the API of the TCP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

TCP is a standard transport layer protocol described in RFC 793. It provides reliable stream-based connections over unreliable networks, and forms the foundation for HTTP, SMTP, and many other protocol standards.

5.8.25.2 Release Notes

MPLAB Harmony Version: v0.70b **TCP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.25.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.25.4 Using the Library

This topic describes the basic architecture of the TCP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [tcp.h](#)

The interface to the TCP TCP/IP Stack library is defined in the "[tcp.h](#)" header file. This file is included by the "tcPIP.h" file. Any C language source (.c) file that uses the TCP TCP/IP Stack library should include "tcPIP.h".

Library File:

The TCP TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

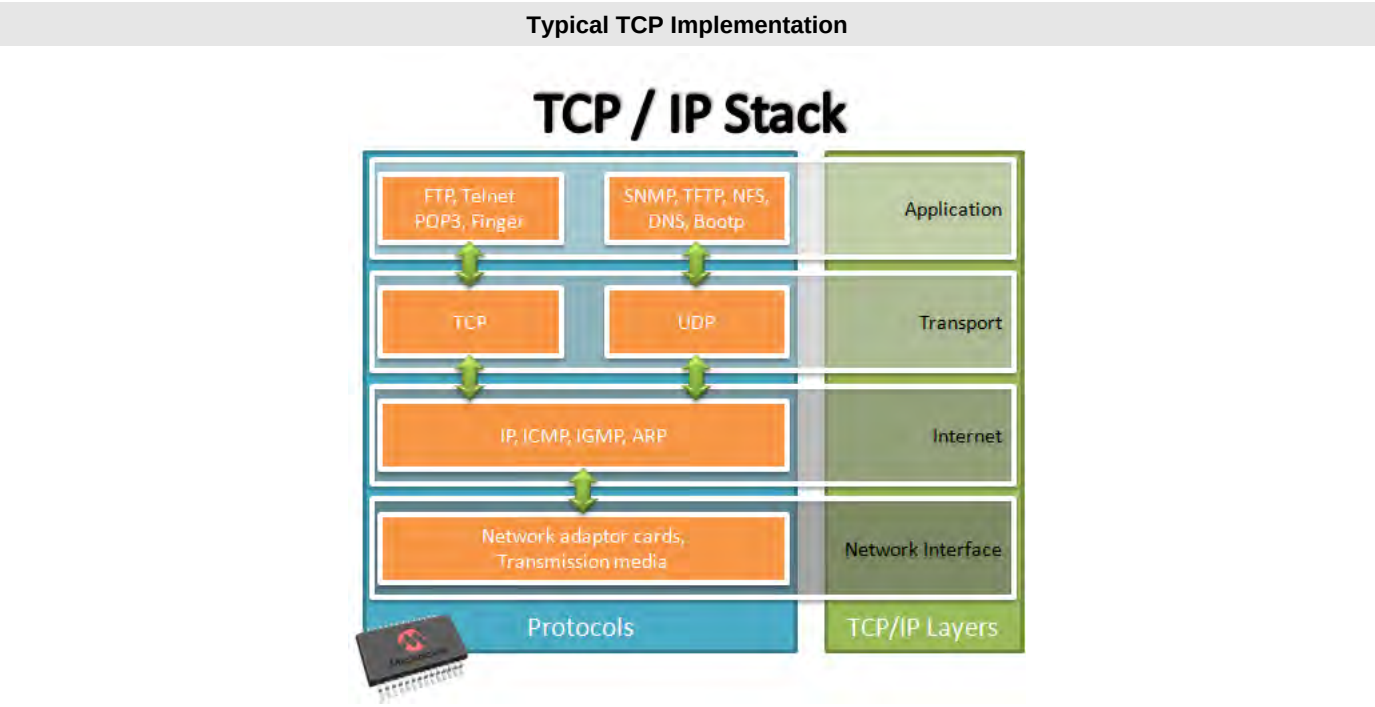
5.8.25.4.1 Abstraction Model

This library provides the API of the TCP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

TCP Software Abstraction Block Diagram

This module provides software abstraction of the TCP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.



5.8.25.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the TCP module.

Library Interface Section	Description
Socket Management Functions	Routines for Managing TCP Sockets
Transmit Data Transfer Functions	Routines for Managing Outgoing Data Transmissions

Receive Data Transfer Functions	Routines for Managing Incoming Data Transmissions
Data Types and Constants	This section provides various definitions describing This API

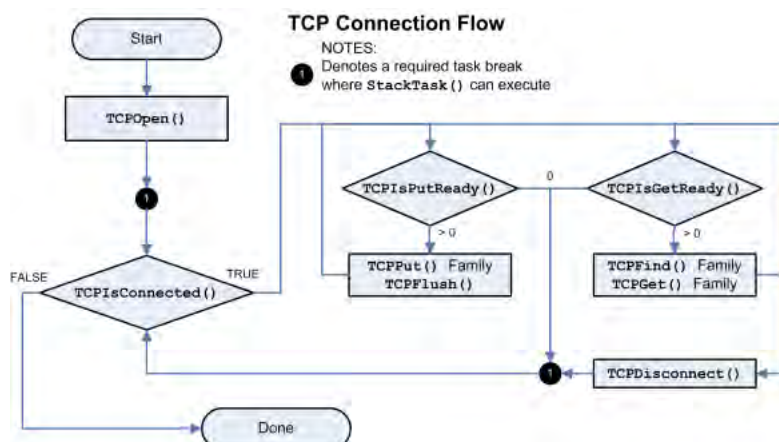
5.8.25.4.3 How the Library Works

Connections made over TCP guarantee data transfer at the expense of throughput. Connections are made through a three-way handshake process, ensuring a one-to-one connection. Remote nodes advertise how much data they are ready to receive, and all data transmitted must be acknowledged. If a remote node fails to acknowledge the receipt of data, it is automatically retransmitted. This ensures that network errors such as lost, corrupted, or out-of-order packets are automatically corrected.

To accomplish this, TCP must operate in a buffer. Once the transmit buffer is full, no more data can be sent until the remote node has acknowledged receipt. For the Microchip TCP/IP Stack, the application must return to the main stack loop in order for this to happen. Likewise, the remote node cannot transmit more data until the local device has acknowledged receipt and that space is available in the buffer. When a local application needs to read more data, it must return to the main stack loop and wait for a new packet to arrive.

5.8.25.4.3.1 Core Functionality

The TCP flow diagram below provides an overview for the use of the TCP module:



Sockets are opened using `TCPOpen`. This function can either open a listening [socket](#) to wait for client connections, or can make a client connection to the remote node. The remote node can be specified by a host name string to be resolved in DNS, an IP address, or a `NODE_INFO` struct containing previously resolved IP and MAC address information.

Once connected, applications can read and write data. On each entry, the application must verify that the [socket](#) is still connected. For most applications a call to [TCPIP_TCP_IsConnected](#) will be sufficient, but [TCPIP_TCP_WasReset](#) may also be used for listening sockets that may turn over quickly.

To write data, call [TCPIP_TCP_PutIsReady](#) to check how much space is available. Then, call any of the [TCPIP_TCP_Put](#) family of functions to write data as space is available. Once complete, call [TCPIP_TCP_Flush](#) to transmit data immediately. Alternately, return to the main stack loop. Data will be transmitted when either a) half of the transmit buffer becomes full or b) a delay time has passed.

To read data, call [TCPIP_TCP_GetIsReady](#) to determine how many bytes are ready to be retrieved. Then use the [TCPIP_TCP_Get](#) family of functions to read data from the [socket](#), and/or the [TCPIP_TCP_Find](#) family of functions to locate data in the buffer. When no more data remains, return to the main stack loop to wait for more data to arrive.

If the application needs to close the connection, call [TCPIP_TCP_Disconnect](#), then return to the main stack loop and wait for the

remote node to acknowledge the disconnection. Client sockets will return to the idle state, while listening sockets will wait for a new connection. For more information, refer to the GenericTCPClient or GenericTCPServer examples, or read the associated RFC.

5.8.25.5 Configuring the Library

Macros

	Name	Description
	<u>LOCAL_TCP_PORT_END_NUMBER</u>	Last port number for randomized local port number selection
	<u>LOCAL_TCP_PORT_START_NUMBER</u>	First port number for randomized local port number selection Use the dynamic port range defined by IANA consists of the 49152-65535 range and is meant for the selection of ephemeral ports (RFC 6056). Adjust to your needs but stay within the IANA range
	<u>TCP_AUTO_TRANSMIT_TIMEOUT_VAL</u>	Timeout before automatically transmitting unflushed data, ms; Default 40 ms
	<u>TCP_CLOSE_WAIT_TIMEOUT</u>	Timeout for the CLOSE_WAIT state, ms
	<u>TCP_DELAYED_ACK_TIMEOUT</u>	Timeout for delayed-acknowledgement algorithm, ms
	<u>TCP_FIN_WAIT_2_TIMEOUT</u>	Timeout for FIN WAIT 2 state, ms
	<u>TCP_KEEP_ALIVE_TIMEOUT</u>	Timeout for keep-alive messages when no traffic is sent, ms
	<u>TCP_MAX_RETRIES</u>	Maximum number of retransmission attempts
	<u>TCP_MAX_SEG_SIZE_RX_LOCAL</u>	TCP Maximum Segment Size for RX (MSS). This value is advertised during TCP connection establishment and the remote node should obey it. The value has to be set in such a way to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem). On Ethernet networks the standard value is 1460. On dial-up links, etc. the default values should be 536. Adjust these values according to your network. The stack will use the local... <u>more</u>
	<u>TCP_MAX_SEG_SIZE_RX_NON_LOCAL</u>	Max segment size, non local
	<u>TCP_MAX_SEG_SIZE_TX</u>	TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unhandlably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that we don't cause any TX buffer overflows. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.
	<u>TCP_MAX_SOCKETS</u>	The maximum number of sockets to create in the stack. When defining TCP_MAX_SOCKETS take into account the number of interfaces the stack is supporting
	<u>TCP_MAX_SYN_RETRIES</u>	Smaller than all other retries to reduce SYN flood DoS duration
	<u>TCP_MAX_UNACKED_KEEP_ALIVES</u>	Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection
	<u>TCP_SOCKET_DEFAULT_RX_SIZE</u>	default <u>socket</u> Rx buffer size
	<u>TCP_SOCKET_DEFAULT_TX_SIZE</u>	default <u>socket</u> Tx buffer size
	<u>TCP_START_TIMEOUT_VAL</u>	TCP Timeout and retransmit numbers All timeouts in milliseconds Timeout to retransmit unacked data, ms
	<u>TCP_TASK_TICK_RATE</u>	5 ms default rate
	<u>TCP_WINDOW_UPDATE_TIMEOUT_VAL</u>	Timeout before automatically transmitting a window update due to a <u>TCPIP_TCP_Get()</u> or <u>TCPIP_TCP_ArrayGet()</u> function call, ms

Description

The configuration of the TCP/IP TCP is based on the file `sys_config.h`

This header file contains the configuration selection for the TCP/IP TCP. Based on the selections made, the TCP/IP TCP will support or not support selected features. These configuration settings will apply to all instances of the TCP/IP TCP.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.25.5.1 LOCAL_TCP_PORT_END_NUMBER Macro

C

```
#define LOCAL_TCP_PORT_END_NUMBER (65535)
```

Description

Last port number for randomized local port number selection

5.8.25.5.2 LOCAL_TCP_PORT_START_NUMBER Macro

C

```
#define LOCAL_TCP_PORT_START_NUMBER (49152)
```

Description

First port number for randomized local port number selection Use the dynamic port range defined by IANA consists of the 49152-65535 range and is meant for the selection of ephemeral ports (RFC 6056). Adjust to your needs but stay within the IANA range

5.8.25.5.3 TCP_AUTO_TRANSMIT_TIMEOUT_VAL Macro

C

```
#define TCP_AUTO_TRANSMIT_TIMEOUT_VAL (40u1)
```

Description

Timeout before automatically transmitting unflushed data, ms; Default 40 ms

5.8.25.5.4 TCP_CLOSE_WAIT_TIMEOUT Macro

C

```
#define TCP_CLOSE_WAIT_TIMEOUT (200u1)
```

Description

Timeout for the CLOSE_WAIT state, ms

5.8.25.5.5 TCP_DELAYED_ACK_TIMEOUT Macro

C

```
#define TCP_DELAYED_ACK_TIMEOUT (100u1)
```

Description

Timeout for delayed-acknowledgement algorithm, ms

5.8.25.5.6 TCP_FIN_WAIT_2_TIMEOUT Macro

C

```
#define TCP_FIN_WAIT_2_TIMEOUT (5000u1)
```

Description

Timeout for FIN WAIT 2 state, ms

5.8.25.5.7 TCP_KEEP_ALIVE_TIMEOUT Macro

C

```
#define TCP_KEEP_ALIVE_TIMEOUT (10000u1)
```

Description

Timeout for keep-alive messages when no traffic is sent, ms

5.8.25.5.8 TCP_MAX_RETRIES Macro

C

```
#define TCP_MAX_RETRIES (5u)
```

Description

Maximum number of retransmission attempts

5.8.25.5.9 TCP_MAX_SEG_SIZE_RX_LOCAL Macro

C

```
#define TCP_MAX_SEG_SIZE_RX_LOCAL (1460)
```

Description

TCP Maximum Segment Size for RX (MSS). This value is advertised during TCP connection establishment and the remote node should obey it. The value has to be set in such a way to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem). On Ethernet networks the standard value is 1460. On dial-up links, etc. the default values should be 536. Adjust these values according to your network. The stack will use the local value for local destination networks and the default value for nonlocal networks.

5.8.25.5.10 TCP_MAX_SEG_SIZE_RX_NON_LOCAL Macro

C

```
#define TCP_MAX_SEG_SIZE_RX_NON_LOCAL (536) // Max segment size, non local
```

Description

Max segment size, non local

5.8.25.5.11 TCP_MAX_SEG_SIZE_TX Macro

C

```
#define TCP_MAX_SEG_SIZE_TX (1460u)
```

Description

TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unhandlably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that we don't cause any TX buffer overflows. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.

5.8.25.5.12 TCP_MAX_SOCKETS Macro

C

```
#define TCP_MAX_SOCKETS (10)
```

Description

The maximum number of sockets to create in the stack. When defining TCP_MAX_SOCKETS take into account the number of interfaces the stack is supporting

5.8.25.5.13 TCP_MAX_SYN_RETRIES Macro

C

```
#define TCP_MAX_SYN_RETRIES (2u)
```

Description

Smaller than all other retries to reduce SYN flood DoS duration

5.8.25.5.14 TCP_MAX_UNACKED_KEEP_ALIVES Macro

C

```
#define TCP_MAX_UNACKED_KEEP_ALIVES (6u)
```

Description

Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection

5.8.25.5.15 TCP_SOCKET_DEFAULT_RX_SIZE Macro

C

```
#define TCP_SOCKET_DEFAULT_RX_SIZE 512
```

Description

default [socket](#) Rx buffer size

5.8.25.5.16 TCP_SOCKET_DEFAULT_TX_SIZE Macro

C

```
#define TCP_SOCKET_DEFAULT_TX_SIZE 512
```

Description

default [socket](#) Tx buffer size

5.8.25.5.17 TCP_START_TIMEOUT_VAL Macro

C

```
#define TCP_START_TIMEOUT_VAL (1000uL)
```

Description

TCP Timeout and retransmit numbers All timeouts in milliseconds Timeout to retransmit unacked data, ms

5.8.25.5.18 TCP_TASK_TICK_RATE Macro

C

```
#define TCP_TASK_TICK_RATE (5)           // 5 ms default rate
```

Description

5 ms default rate

5.8.25.5.19 TCP_WINDOW_UPDATE_TIMEOUT_VAL Macro

C

```
#define TCP_WINDOW_UPDATE_TIMEOUT_VAL (200uL)
```

Description

Timeout before automatically transmitting a window update due to a [TCPIP_TCP_Get\(\)](#) or [TCPIP_TCP_ArrayGet\(\)](#) function call, ms

5.8.25.6 Building the Library

This section list the files that are available in the \src of the TCP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- tcp.c

5.8.25.7 Library Interface

Data Types and Constants

	Name	Description
	TCPIP_TCP_FifoRxFullGet	TCP Get Rx First In, First Out Full
	TCPIP_TCP_FifoTxFreeGet	TCP Get TX First In, First Out Free
	TCP_ADJUST_FLAGS	TCP Adjust Falgs

	TCP_MODULE_CONFIG	TCP layer configuration/initialization
	TCP_OPTION_LINGER_DATA	Linger options
	TCP_PORT	TCP Port
	TCP_SOCKET	TCP Socket
	TCP_SOCKET_INFO	TCP Socket Information
	TCP_SOCKET_OPTION	TCP Socket Option
	INVALID_SOCKET	The socket is invalid or could not be opened

Receive Data Transfer Functions

	Name	Description
	TCPIP_TCP_ArrayFind	Searches for a string in the TCP RX buffer.
	TCPIP_TCP_ArrayGet	Reads an array of data bytes from a TCP socket 's receive FIFO. The data is removed from the FIFO in the process.
	TCPIP_TCP_ArrayPeek	Reads a specified number of data bytes from the TCP RX FIFO without removing them from the buffer.
	TCPIP_TCP_Discard	Discards any pending data in the TCP RX FIFO.
	TCPIP_TCP_FifoSizeAdjust	Adjusts the relative sizes of the RX and TX buffers.
	TCPIP_TCP_Find	Searches for a byte in the TCP RX buffer.
	TCPIP_TCP_Get	Retrieves a single byte to a TCP socket .
	TCPIP_TCP_GetIsReady	Determines how many bytes can be read from the TCP RX buffer.
	TCPIP_TCP_Peek	Peaks at one byte in the TCP RX FIFO without removing it from the buffer.
	TCPIP_TCP_FifoRxFreeGet	Determines how many bytes are free in the RX FIFO.

Socket Management Functions

	Name	Description
	TCPIP_TCP_Bind	Bind a socket to a local address. This function is meant for unconnected server and client sockets. It is similar to TCPIP_TCP_SocketNetSet() that assigns a specific source interface for a socket . If localPort is 0 the stack will assign a unique local port.
	TCPIP_TCP_ClientOpen	Opens a TCP socket as a client.
	TCPIP_TCP_Close	Disconnects an open socket and destroys the socket handle, releasing the associated resources.
	TCPIP_TCP_Connect	Connects a client socket .
	TCPIP_TCP_Disconnect	Disconnects an open socket .
	TCPIP_TCP_IsConnected	Determines if a socket has an established connection.
	TCPIP_TCP_OptionsGet	Allows getting the options for a socket like: current Rx/Tx buffer size, etc.
	TCPIP_TCP_OptionsSet	Allows setting options to a socket like adjust Rx/Tx buffer size, etc.
	TCPIP_TCP_RemoteBind	Bind a socket to a remote address. This function is meant for unconnected server and client sockets.
	TCPIP_TCP_ServerOpen	Opens a TCP socket as a server.
	TCPIP_TCP_SocketInfoGet	Obtains information about a currently open socket .
	TCPIP_TCP_SocketNetGet	Gets the MAC interface of an TCP socket .
	TCPIP_TCP_WasReset	Self-clearing semaphore indicating socket reset.
	TCPIP_TCP_Abort	Aborts a connection.
	TCPIP_TCP_SocketNetSet	Sets the interface for an TCP socket .

Transmit Data Transfer Functions

	Name	Description
≡	TCPIP_TCP_ArrayPut	Writes an array from RAM to a TCP socket .
≡	TCPIP_TCP_FifoTxFullGet	Determines how many bytes are pending in the TCP TX FIFO.
≡	TCPIP_TCP_Flush	Immediately transmits all pending TX data.
≡	TCPIP_TCP_Put	Writes a single byte to a TCP socket .
≡	TCPIP_TCP_PutIsReady	Determines how much free space is available in the TCP TX buffer.
≡	TCPIP_TCP_StringPut	Writes a null-terminated string from RAM to a TCP socket . The null-terminator is not copied to the socket .

Description

This section describes the Application Programming Interface (API) functions of the TCP TCP/IP Stack Library.

Refer to each section for a detailed description.

5.8.25.7.1 Socket Management Functions**5.8.25.7.1.1 TCPIP_TCP_Bind Function****C**

```
bool TCPIP_TCP_Bind(
    TCP_SOCKET hTCP,
    IP_ADDRESS_TYPE addType,
    TCP_PORT localPort,
    IP_MULTI_ADDRESS* localAddress
);
```

Description

Sockets don't need specific binding, it is done automatically by the stack. However, specific binding can be requested using these functions. Works for both client and server sockets.

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	Socket to bind
addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
localPort	Local port to use. If 0, the stack will assign a unique local port.
localAddress	The local address to bind to. Could be NULL if the local address does not need to be changed.

Returns

True of success, false otherwise.

Remarks

The call should fail if the [socket](#) is already connected (both server and client sockets). However, this is not currently implemented. It is the user's responsibility to call this function only for sockets that are not connected. Changing the [socket](#) parameters while the [socket](#) is connected will result in connection loss/unpredictable behavior.

5.8.25.7.1.2 TCPIP_TCP_ClientOpen Function

C

```
TCP_SOCKET TCPIP_TCP_ClientOpen(
    IP_ADDRESS_TYPE addType,
    TCP_PORT remotePort,
    IP_MULTI_ADDRESS* remoteAddress
);
```

Description

Provides a unified method for opening TCP client sockets.

Sockets are dynamically allocated at stack initialization, and can be claimed with this function and freed using [TCPIP_TCP_Abort](#) or [TCPIP_TCP_Close](#).

Preconditions

TCP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
TCP_PORT remotePort	TCP port to connect to. The local port for client sockets will be automatically picked by the TCP module.
IP_MULTI_ADDRESS* remoteAddress	The remote address to be used. If 0 then the address is unspecified

Returns

[INVALID_SOCKET](#) - No sockets of the specified type were available to be opened. Otherwise - A [TCP_SOCKET](#) handle. Save this handle and use it when calling all other TCP APIs.

Remarks

IP_ADDRESS_TYPE_ANY is not supported (not a valid type for client open)!

If the remoteAddress != 0 (and the address pointed by remoteAddress != 0) then the [socket](#) will immediately initiate a connection to the remote host

If the remoteAddress is unspecified, then no connection is initiated. Client [socket](#) parameters can be set using [TCPIP_TCP_Bind\(\)](#), [TCPIP_TCP_RemoteBind\(\)](#), etc. calls and then connection initiated by calling [TCPIP_TCP_Connect\(\)](#).

5.8.25.7.1.3 TCPIP_TCP_Close Function

C

```
void TCPIP_TCP_Close(
    TCP_SOCKET hTCP
);
```

Description

- If the graceful option is set for the [socket](#) (default):
 - a [TCPIP_TCP_Disconnect](#) will be tried. If the linger option is set (default) the [TCPIP_TCP_Disconnect](#) will try to [send](#) any queued TX data before issuing FIN. If the FIN [send](#) operation fails or the [socket](#) is not connected the abort is generated.
- If the graceful option is not set, or the previous step could not [send](#) the FIN: A TCPAbort() is called, sending a RST to the remote node. Communication is closed, the [socket](#) is no longer valid and the associated resources are freed.

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	Handle to the socket to disconnect and close.

Returns

None

5.8.25.7.1.4 TCPIP_TCP_Connect Function**C**

```
bool TCPIP_TCP_Connect(  
    TCP_SOCKET hTCP  
);
```

Description

This function will try to initiate a connection on a client [socket](#) that's not connected yet. The client [socket](#) should have been created with a call to [TCPIP_TCP_ClientOpen](#) having the remoteAddress set to 0;

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ClientOpen\(\)](#) with an unspecified address. hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	Handle to the client socket to connect .

Returns

true is the call succeeded false otherwise

Remarks

the call will fail if the client [socket](#) has no remote host specified. Use [TCPIP_TCP_RemoteBind\(\)](#) to specify a remote host address for the client [socket](#).

5.8.25.7.1.5 TCPIP_TCP_Disconnect Function**C**

```
bool TCPIP_TCP_Disconnect(  
    TCP_SOCKET hTCP  
);
```

Description

This function closes the TX side of a connection by sending a FIN (if currently connected) to the remote node of the connection.

If the [socket](#) has the linger option set (default), the queued TX data transmission will be attempted before sending the FIN. If the linger option is off, the queued TX data will be discarded.

Please note that this call may fail in which case it can be re-issued.

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	Handle of the socket to disconnect.

Returns

true if the call succeeded false otherwise; that means that the notification could not be sent to the remote host. the call can be re-issued at a later time if desired

Remarks

If the [socket](#) is using SSL, a CLOSE_NOTIFY record will be transmitted first to allow the SSL session to be resumed at a later time.

5.8.25.7.1.6 TCPIP_TCP_IsConnected Function**C**

```
bool TCPIP_TCP_IsConnected(  
    TCP_SOCKET hTCP  
);
```

Description

This function determines if a [socket](#) has an established connection to a remote node. Call this function after calling [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#) to determine when the connection is set up and ready for use. This function was historically used to check for disconnections, but [TCPIP_TCP_WasReset](#) is now a more appropriate solution.

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	The socket to check.

Return Values

Return Values	Description
true	The socket has an established connection to a remote node.
false	The socket is not currently connected.

Remarks

A [socket](#) is said to be connected only if it is in the TCP_ESTABLISHED state. Sockets in the process of opening or closing will return false.

5.8.25.7.1.7 TCPIP_TCP_OptionsGet Function**C**

```
bool TCPIP_TCP_OptionsGet(  
    TCP_SOCKET hTCP,  
    TCP_SOCKET_OPTION option,  
    void* optParam  
);
```

Description

Various options can be get at the [socket](#) level. This function provides compatibility with BSD implementations.

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	socket to get options for
option	specific option to get
optParam	pointer to an area that will receive the option value; this is option dependent the size of the area has to be large enough to accomodate the specific option TCP_OPTION_LINGER -- pointer to a TCP_OPTION_LINGER_DATA structure TCP_OPTION_KEEPPALIVE -- not supported yet TCP_OPTION_RX_BUFF -- size of the new RX buffer TCP_OPTION_TX_BUFF -- size of the new TX buffer TCP_OPTION_RX_TMO -- not supported yet TCP_OPTION_TX_TMO -- not supported yet TCP_OPTION_NODELAY -- pointer to boolean to return current NO DELAY status TCP_OPTION_EXCLUSIVE_ADDRESS -- not supported yet

Returns

true if success false otherwise

5.8.25.7.1.8 TCPIP_TCP_OptionsSet Function**C**

```
bool TCPIP_TCP_OptionsSet(
    TCP_SOCKET hTCP,
    TCP_SOCKET_OPTION option,
    void* optParam
);
```

Description

Various options can be set at the [socket](#) level. This function provides compatibility with BSD implementations.

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	socket to set options for
option	specific option to be set
optParam	the option value; this is option dependent TCP_OPTION_LINGER -- pointer to a TCP_OPTION_LINGER_DATA structure TCP_OPTION_KEEPPALIVE -- not supported yet TCP_OPTION_RX_BUFF -- size of the new RX buffer TCP_OPTION_TX_BUFF -- size of the new TX buffer TCP_OPTION_RX_TMO -- not supported yet TCP_OPTION_TX_TMO -- not supported yet TCP_OPTION_NODELAY -- boolean to enable/disable the NO DELAY functionality TCP_OPTION_EXCLUSIVE_ADDRESS -- not supported yet

Returns

true if success false otherwise

5.8.25.7.1.9 TCPIP_TCP_RemoteBind Function**C**

```
bool TCPIP_TCP_RemoteBind(
    TCP_SOCKET hTCP,
```

```
    IP_ADDRESS_TYPE addType,  
    TCP_PORT remotePort,  
    IP_MULTI_ADDRESS* remoteAddress  
);
```

Description

Sockets don't need specific remote binding, they should [accept](#) connections on any incoming interface. Thus the binding is done automatically by the stack. However, specific remote binding can be requested using these functions. For a server [socket](#) it can be used to restrict accepting connections from a specific remote host. For a client [socket](#) it will just change the default binding done when the [socket](#) was opened.

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	Socket to bind
addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
remotePort	remote port to use Could be 0 if the remote port does not need to be changed
remoteAddress	The remote address to bind to. Could be NULL if the remote address does not need to be changed

Returns

True of success false otherwise

Remarks

The [socket](#) remote port is changed only if a non-zero remotePort value is passed. The [socket](#) remote host address is changed only if a non-zero remoteAddress value is passed.

The call should fail if the [socket](#) is already connected (both server and client sockets). However this is not currently implemented. It is the user's responsibility to call this function only for sockets that are not connected. Changing the [socket](#) parameters while the [socket](#) is connected will result in connection loss/unpredictable behavior.

5.8.25.7.1.10 TCPIP_TCP_ServerOpen Function

C

```
TCP_SOCKET TCPIP_TCP_ServerOpen(  
    IP_ADDRESS_TYPE addType,  
    TCP_PORT localPort,  
    IP_MULTI_ADDRESS* localAddress  
);
```

Description

Provides a unified method for opening TCP server sockets.

Sockets are statically/dynamically allocated on boot, and can be claimed with this function and freed using [TCPIP_TCP_Close](#).

Preconditions

TCP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
TCP_PORT localPort	TCP port to listen on for connections.

IP_MULTI_ADDRESS* localAddress	Local address to use.

Returns

[INVALID_SOCKET](#) - No sockets of the specified type were available to be opened. Otherwise - A [TCP_SOCKET](#) handle. Save this handle and use it when calling all other TCP APIs.

5.8.25.7.1.11 TCPIP_TCP_SocketInfoGet Function**C**

```
bool TCPIP_TCP_SocketInfoGet(  
    TCP_SOCKET hTCP,  
    TCP_SOCKET_INFO* pInfo  
);
```

Description

Fills the provided [TCP_SOCKET_INFO](#) structure associated with this [socket](#). This contains the NODE_INFO structure with IP and MAC address (or gateway MAC) and the remote port.

Preconditions

TCP is initialized and the [socket](#) is connected.

Parameters

Parameters	Description
hTCP	The socket to check.

Returns

true if the call succeeded false if no such [socket](#) or the [socket](#) is not opened

5.8.25.7.1.12 TCPIP_TCP_SocketNetGet Function**C**

```
TCPIP_NET_HANDLE TCPIP_TCP_SocketNetGet(  
    TCP_SOCKET hTCP  
);
```

Description

This function returns the interface handle of an TCP [socket](#)

Preconditions

TCP [socket](#) should have been opened with `_TCP_Open()`. hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	The TCP socket

Returns

the handle of the local interface this [socket](#) is bound to.

5.8.25.7.1.13 TCPIP_TCP_WasReset Function**C**

```
bool TCPIP_TCP_WasReset(  
    TCP_SOCKET hTCP
```



```
);
```

Description

This function is a self-clearing semaphore indicating whether or not a [socket](#) has been disconnected since the previous call. This function works for all possible disconnections: a call to [TCPIP_TCP_Disconnect](#), a FIN from the remote node, or an acknowledgement timeout caused by the loss of a network link. It also returns true after the first call to [TCPIP_TCP_Initialize](#). Applications should use this function to reset their state machines.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Return Values

Return Values	Description
true	The socket has been disconnected since the previous call.
false	The socket has not been disconnected since the previous call.

5.8.25.7.1.14 TCPIP_TCP_Abort Function

C

```
void TCPIP_TCP_Abort(  
    TCP_SOCKET hTCP,  
    bool killSocket  
);
```

Description

This function aborts a connection to a remote node by sending a RST (if currently connected). Any pending TX/RX data is discarded.

A client [socket](#) will always be closed and the associated resources released. The [socket](#) cannot be used again after this call.

A server [socket](#) will abort the current connection:

- if killSocket == false the [socket](#) will remain listening
- if killSocket == true the [socket](#) will be closed and all associated resources released. The [socket](#) cannot be used again after this call.

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	Handle to the socket to disconnect.
killSocket	if true, it kills a server socket

Returns

None

5.8.25.7.1.15 TCPIP_TCP_SocketNetSet Function

C

```
bool TCPIP_TCP_SocketNetSet(  
    TCP_SOCKET hTCP,  
    TCPIP_NET_HANDLE hNet  
);
```

Description

This function sets the network interface for an TCP [socket](#)

Preconditions

TCP [socket](#) should have been opened with `_TCP_Open()`. hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	The TCP socket
hNet	interface handle.

Returns

true if success false otherwise.

Remarks

None

5.8.25.7.2 Transmit Data Transfer Functions

5.8.25.7.2.1 TCPIP_TCP_ArrayPut Function

C

```
uint16_t TCPIP_TCP_ArrayPut(  
    TCP_SOCKET hTCP,  
    const uint8_t* Data,  
    uint16_t Len  
);
```

Description

Writes an array from RAM to a TCP [socket](#).

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the array to be written.
len	Number of bytes to be written.

Returns

The number of bytes written to the [socket](#). If less than len, the buffer became full or the [socket](#) is not connected.

5.8.25.7.2.2 TCPIP_TCP_FifoTxFullGet Function

C

```
uint16_t TCPIP_TCP_FifoTxFullGet(  
    TCP_SOCKET hTCP  
);
```

Description

Determines how many bytes are pending in the TCP TX FIFO.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Returns

Number of bytes pending to be flushed in the TCP TX FIFO.

5.8.25.7.2.3 TCPIP_TCP_Flush Function

C

```
bool TCPIP_TCP_Flush(  
    TCP_SOCKET hTCP  
);
```

Description

This function immediately transmits all pending TX data with a PSH flag. If this function is not called, data will automatically be sent when either a) the TX buffer is half full or b) the [TCP_AUTO_TRANSMIT_TIMEOUT_VAL](#) (default: 40ms) has elapsed.

Preconditions

TCP is initialized and the [socket](#) is connected.

Parameters

Parameters	Description
hTCP	The socket whose data is to be transmitted.

Returns

None

Remarks

SSL application data is automatically flushed, so this function has no effect for SSL sockets.

5.8.25.7.2.4 TCPIP_TCP_Put Function

C

```
uint16_t TCPIP_TCP_Put(  
    TCP_SOCKET hTCP,  
    uint8_t byte  
);
```

Description

Writes a single byte to a TCP [socket](#).

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	The socket to which data is to be written.
byte	The byte to write.

Return Values

Return Values	Description
0	The byte was written to the transmit buffer.
1	The transmit buffer was full, or the socket is not connected.

Remarks

Note that the following function is inefficient. A buffered approach ([TCPIP_TCP_ArrayPut](#)) should be preferred

5.8.25.7.2.5 TCPIP_TCP_PutIsReady Function**C**

```
uint16_t TCPIP_TCP_PutIsReady(  
    TCP_SOCKET hTCP  
);
```

Description

Call this function to determine how many bytes can be written to the TCP TX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to transmit more data.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Returns

The number of bytes available to be written in the TCP TX buffer.

5.8.25.7.2.6 TCPIP_TCP_StringPut Function**C**

```
const uint8_t* TCPIP_TCP_StringPut(  
    TCP_SOCKET hTCP,  
    const uint8_t* Data  
);
```

Description

Writes a null-terminated string from RAM to a TCP [socket](#). The null-terminator is not copied to the [socket](#).

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the string to be written.

Returns

Pointer to the byte following the last byte written to the [socket](#). If this pointer does not dereference to a NUL byte, the buffer became full or the [socket](#) is not connected.

Remarks

The return value of this function differs from that of [TCPIP_TCP_ArrayPut](#). To write long strings in a single state, initialize the *data pointer to the first byte, then call this function repeatedly (breaking to the main stack loop after each call) until the return value dereferences to a NUL byte. Save the return value as the new starting *data pointer otherwise.

5.8.25.7.3 Receive Data Transfer Functions

5.8.25.7.3.1 TCPIP_TCP_ArrayFind Function

C

```
uint16_t TCPIP_TCP_ArrayFind(  
    TCP_SOCKET hTCP,  
    const uint8_t* cFindArray,  
    uint16_t wLen,  
    uint16_t wStart,  
    uint16_t wSearchLen,  
    bool bTextCompare  
);
```

Description

This function finds the first occurrence of an array of bytes in the TCP RX buffer. It can be used by an application to abstract searches out of their own application code. For increased efficiency, the function is capable of limiting the scope of search to a specific range of bytes. It can also perform a case-insensitive search if required.

For example, if the buffer contains "I love PIC MCUs!" and the search array is "love" with a length of 4, a value of 2 will be returned.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to search within.
cFindArray	The array of bytes to find in the buffer.
wLen	Length of cFindArray.
wStart	Zero-indexed starting position within the buffer.
wSearchLen	Length from wStart to search in the buffer.
bTextCompare	true for case-insensitive text search, false for binary search

Return Values

Return Values	Description
0xFFFF	Search array not found

Otherwise	Zero-indexed position of the first occurrence

Remarks

Since this function usually must transfer data from external storage to internal RAM for comparison, its performance degrades significantly when the buffer is full and the array is not found. For better performance, try to search for characters that are expected to exist or limit the scope of the search as much as possible. The HTTP module, for example, uses this function to parse headers. However, it searches for newlines, then the separating colon, then reads the header name to RAM for final comparison. This has proven to be significantly faster than searching for full header name strings outright.

5.8.25.7.3.2 TCPIP_TCP_ArrayGet Function**C**

```
uint16_t TCPIP_TCP_ArrayGet(
    TCP_SOCKET hTCP,
    uint8_t* buffer,
    uint16_t count
);
```

Description

Reads an array of data bytes from a TCP [socket](#)'s receive FIFO. The data is removed from the FIFO in the process.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket from which data is to be read.
buffer	Pointer to the array to store data that was read.
len	Number of bytes to be read.

Returns

The number of bytes read from the [socket](#). If less than len, the RX FIFO buffer became empty or the [socket](#) is not connected.

5.8.25.7.3.3 TCPIP_TCP_ArrayPeek Function**C**

```
uint16_t TCPIP_TCP_ArrayPeek(
    TCP_SOCKET hTCP,
    uint8_t * vBuffer,
    uint16_t wLen,
    uint16_t wStart
);
```

Description

Reads a specified number of data bytes from the TCP RX FIFO without removing them from the buffer. No TCP control actions are taken as a result of this function (ex: no window update is sent to the remote node).

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to peak from (read without removing from stream).

vBuffer	Destination to write the peeked data bytes.
wLen	Length of bytes to peak from the RX FIFO and copy to vBuffer.
wStart	Zero-indexed starting position within the FIFO to start peeking from.

Remarks

None

5.8.25.7.3.4 TCPIP_TCP_Discard Function**C**

```
uint16_t TCPIP_TCP_Discard(
    TCP_SOCKET hTCP
);
```

Description

Discards any pending data in the TCP RX FIFO.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket whose RX FIFO is to be cleared.

Returns

Number of bytes that have been discarded from the RX buffer.

5.8.25.7.3.5 TCPIP_TCP_FifoSizeAdjust Function**C**

```
bool TCPIP_TCP_FifoSizeAdjust(
    TCP_SOCKET hTCP,
    uint16_t wMinRXSize,
    uint16_t wMinTXSize,
    TCP_ADJUST_FLAGS vFlags
);
```

Description

This function can be used to simultaneously adjust the sizes of the RX and TX FIFOs.

Adjusting the size of the TX/RX FIFO on the fly can allow for optimal transmission speed for one-sided application protocols. For example, HTTP typically begins by receiving large amounts of data from the client, then switches to serving large amounts of data back. Adjusting the FIFO at these points can increase performance in systems that have limited resources. Once the FIFOs are adjusted, a window update is sent.

The [TCP_ADJUST_FLAGS](#) control the distribution of the remaining available space between the TX and RX FIFOs. If neither or both of TCP_ADJUST_GIVE_REST_TO_TX and TCP_ADJUST_GIVE_REST_TO_RX are set, the function distributes the remaining space (if any) equally. If the new requested FIFOs space is greater than the old existing FIFOs space the TCP_ADJUST_GIVE_REST_TO_TX and TCP_ADJUST_GIVE_REST_TO_RX are ignored.

TCP_ADJUST_PRESERVE_RX and TCP_ADJUST_PRESERVE_TX request the preserving of the existing data. Existing data can be preserved as long as the old data in the buffer does not exceed the capacity of the new buffer.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to be adjusted
wMinRXSize	Minimum number of bytes for the RX FIFO
wMinTXSize	Minimum number of bytes for the TX FIFO
vFlags	If TCP_ADJUST_TX_ONLY or TCP_ADJUST_RX_ONLY are not set, then the TX and RX buffers are evaluated together and any

5.8.25.7.3.6 TCPIP_TCP_Find Function**C**

```
uint16_t TCPIP_TCP_Find(  
    TCP_SOCKET hTCP,  
    uint8_t cFind,  
    uint16_t wStart,  
    uint16_t wSearchLen,  
    bool bTextCompare  
);
```

Description

This function finds the first occurrence of a byte in the TCP RX buffer. It can be used by an application to abstract searches out of their own application code. For increased efficiency, the function is capable of limiting the scope of search to a specific range of bytes. It can also perform a case-insensitive search if required.

For example, if the buffer contains "I love PIC MCUs!" and the cFind byte is ' ', a value of 1 will be returned.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to search within.
cFind	The byte to find in the buffer.
wStart	Zero-indexed starting position within the buffer.
wSearchLen	Length from wStart to search in the buffer.
bTextCompare	true for case-insensitive text search, false for binary search

Return Values

Return Values	Description
0xFFFF	Search array not found
Otherwise	Zero-indexed position of the first occurrence

Remarks

Since this function usually must transfer data from external storage to internal RAM for comparison, its performance degrades significantly when the buffer is full and the array is not found. For better performance, try to search for characters that are expected to exist or limit the scope of the search as much as possible. The HTTP module, for example, uses this function to parse headers. However, it searches for newlines, then the separating colon, then reads the header name to RAM for final comparison. This has proven to be significantly faster than searching for full header name strings outright.

5.8.25.7.3.7 TCPIP_TCP_Get Function

C

```
uint16_t TCPIP_TCP_Get(  
    TCP_SOCKET hTCP,  
    uint8_t* byte  
);
```

Description

Retrieves a single byte to a TCP [socket](#).

Preconditions

TCP [socket](#) should have been opened with [TCPIP_TCP_ServerOpen\(\)](#)/[TCPIP_TCP_ClientOpen\(\)](#). hTCP - valid [socket](#)

Parameters

Parameters	Description
hTCP	The socket from which to read.
byte	Pointer to location in which the read byte should be stored.

Return Values

Return Values	Description
1	A byte was read from the buffer.
0	The buffer was empty, or the socket is not connected.

Remarks

Note that the following function is inefficient. A buffered approach ([TCPIP_TCP_ArrayGet](#)) should be preferred

5.8.25.7.3.8 TCPIP_TCP_GetIsReady Function

C

```
uint16_t TCPIP_TCP_GetIsReady(  
    TCP_SOCKET hTCP  
);
```

Description

Call this function to determine how many bytes can be read from the TCP RX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to wait for more data to arrive.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Returns

The number of bytes available to be read from the TCP RX buffer.

5.8.25.7.3.9 TCPIP_TCP_Peek Function

C

```
uint8_t TCPIP_TCP_Peek(  
    TCP_SOCKET hTCP,
```

```
    uint16_t wStart  
);
```

Description

Peeks at one byte in the TCP RX FIFO without removing it from the buffer.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to peak from (read without removing from stream).
wStart	Zero-indexed starting position within the FIFO to peek from.

Remarks

Use the [TCPIP_TCP_ArrayPeek\(\)](#) function to read more than one byte. It will perform better than calling [TCPIP_TCP_Peek\(\)](#) in a loop.

5.8.25.7.3.10 TCPIP_TCP_FifoRxFreeGet Function

C

```
uint16_t TCPIP_TCP_FifoRxFreeGet(  
    TCP_SOCKET hTCP  
);
```

Description

Determines how many bytes are free in the RX FIFO.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Returns

The number of bytes free in the TCP RX FIFO. If zero, no additional data can be received until the application removes some data using one of the [TCPIP_TCP_Get](#) family functions.

5.8.25.7.4 Data Types and Constants

5.8.25.7.4.1 TCPIP_TCP_FifoRxFullGet Macro

C

```
#define TCPIP_TCP_FifoRxFullGet(a) TCPIP_TCP_GetIsReady(a)
```

Description

Macro: [TCPIP_TCP_FifoRxFullGet](#)

Alias to [TCPIP_TCP_GetIsReady](#) provided for API completeness

5.8.25.7.4.2 TCPIP_TCP_FifoTxFreeGet Macro

C

```
#define TCPIP_TCP_FifoTxFreeGet(a) TCPIP_TCP_PutIsReady(a)
```

Description

Macro: TCPIP_TCP_FifoTxFreeGet([TCP_SOCKET](#) hTCP)

Alias to [TCPIP_TCP_PutIsReady](#) provided for API completeness

Parameters

Parameters	Description
hTCP	The socket to check.

Returns

The number of bytes available to be written in the TCP TX buffer.

5.8.25.7.4.3 TCP_ADJUST_FLAGS Enumeration

C

```
typedef enum {
    TCP_ADJUST_GIVE_REST_TO_RX = 0x01,
    TCP_ADJUST_GIVE_REST_TO_TX = 0x02,
    TCP_ADJUST_PRESERVE_RX = 0x04,
    TCP_ADJUST_PRESERVE_TX = 0x08,
    TCP_ADJUST_TX_ONLY = 0x10,
    TCP_ADJUST_RX_ONLY = 0x20
} TCP_ADJUST_FLAGS;
```

Description

Enumeration: TCP_ADJUST_FLAGS

Adjust [socket](#) buffer sizes.

Members

Members	Description
TCP_ADJUST_GIVE_REST_TO_RX = 0x01	Resize flag: extra bytes go to RX
TCP_ADJUST_GIVE_REST_TO_TX = 0x02	Resize flag: extra bytes go to TX
TCP_ADJUST_PRESERVE_RX = 0x04	Resize flag: attempt to preserve RX buffer
TCP_ADJUST_PRESERVE_TX = 0x08	Resize flag: attempt to preserve TX buffer
TCP_ADJUST_TX_ONLY = 0x10	Resize flag: adjust the TX buffer only
TCP_ADJUST_RX_ONLY = 0x20	Resize flag: adjust the RX buffer only

5.8.25.7.4.4 TCP_MODULE_CONFIG Structure

C

```
typedef struct {
    int nSockets;
    uint16_t sktTxBuffSize;
    uint16_t sktRxBuffSize;
} TCP_MODULE_CONFIG;
```

Description

Structure: TCP_MODULE_CONFIG

TCP layer configuration/initialization

Members

Members	Description
int nSockets;	number of sockets to be created
uint16_t sktTxBuffSize;	size of the socket tx buffer
uint16_t sktRxBuffSize;	size of the socket rx buffer

5.8.25.7.4.5 TCP_OPTION_LINGER_DATA Structure

C

```
typedef struct {  
    bool lingerEnable;  
    bool gracefulEnable;  
    uint16_t lingerTmo;  
} TCP_OPTION_LINGER_DATA;
```

Description

Structure: TCP_OPTION_LINGER_DATA

Linger options

Members

Members	Description
bool lingerEnable;	enable/disable linger; enabled by default for any socket
bool gracefulEnable;	enable/disable graceful close; enabled by default for any socket
uint16_t lingerTmo;	linger timeout in seconds (when enabled) this option is not supported yet

5.8.25.7.4.6 TCP_PORT Type

C

```
typedef uint16_t TCP_PORT;
```

Description

Type: TCP_PORT

TCP Port Number identifier

5.8.25.7.4.7 TCP_SOCKET Type

C

```
typedef int16_t TCP_SOCKET;
```

Description

Type: TCP_SOCKET

A TCP_SOCKET is stored as a single uint8_t

5.8.25.7.4.8 TCP_SOCKET_INFO Structure

C

```
typedef struct {
    IP_ADDRESS_TYPE  addressType;
    IP_MULTI_ADDRESS remoteIPAddress;
    IP_MULTI_ADDRESS localIPAddress;
    TCP_PORT  remotePort;
    TCP_PORT  localPort;
} TCP_SOCKET_INFO;
```

Description

Structure: TCP_SOCKET_INFO

Information about a [socket](#)

Members

Members	Description
TCP_PORT remotePort;	Port number associated with remote node
TCP_PORT localPort;	local port number

5.8.25.7.4.9 TCP_SOCKET_OPTION Enumeration

C

```
typedef enum {
    TCP_OPTION_LINGER,
    TCP_OPTION_KEEPPALIVE,
    TCP_OPTION_RX_BUFF,
    TCP_OPTION_TX_BUFF,
    TCP_OPTION_RX_TMO,
    TCP_OPTION_TX_TMO,
    TCP_OPTION_NODELAY,
    TCP_OPTION_EXCLUSIVE_ADDRESS
} TCP_SOCKET_OPTION;
```

Description

Enumeration: TCP_SOCKET_OPTION

Socket options

Members

Members	Description
TCP_OPTION_LINGER	The LINGER option controls the action taken when unsent data is queued on a socket and the socket is closed. The linger option can be turned on/off and the timeout can be specified.
TCP_OPTION_KEEPPALIVE	enable the use of keep-alive packets on TCP connections The option can be turned on/off and the timeout can be specified
TCP_OPTION_RX_BUFF	request different RX buffer size. Has to call TCPIP_TCP_OptionsGet to see the exact space allocated
TCP_OPTION_TX_BUFF	request different TX buffer size. Has to call TCPIP_TCP_OptionsGet to see the exact space allocated
TCP_OPTION_RX_TMO	specifies the RX timeout. If no data arrives in the specified timeout the socket is closed
TCP_OPTION_TX_TMO	specifies the TX timeout. If no data can be sent in the specified timeout the socket is closed
TCP_OPTION_NODELAY	enables the NO DELAY/Nagle algorithm functionality;

TCP_OPTION_EXCLUSIVE_ADDRESS	enables a socket to be bound for exclusive access.

5.8.25.7.4.10 INVALID_SOCKET Macro

C

```
#define INVALID_SOCKET (-1)    // The socket is invalid or could not be opened
```

Description

The [socket](#) is invalid or could not be opened

5.8.25.8 Files

Files

Name	Description
tcp.h	Transmission Control Protocol (TCP) Communications Layer API
tcp_config.h	TCP configuration file

Description

5.8.25.8.1 tcp.h

- Provides reliable, handshaked transport of application stream oriented data with flow control
- Reference: RFC 793

Enumerations

	Name	Description
	TCP_ADJUST_FLAGS	TCP Adjust Falgs
	TCP_SOCKET_OPTION	TCP Socket Option

Functions

	Name	Description
	TCPIP_TCP_Abort	Aborts a connection.
	TCPIP_TCP_ArrayFind	Searches for a string in the TCP RX buffer.
	TCPIP_TCP_ArrayGet	Reads an array of data bytes from a TCP socket 's receive FIFO. The data is removed from the FIFO in the process.
	TCPIP_TCP_ArrayPeek	Reads a specified number of data bytes from the TCP RX FIFO without removing them from the buffer.
	TCPIP_TCP_ArrayPut	Writes an array from RAM to a TCP socket .
	TCPIP_TCP_Bind	Bind a socket to a local address This function is meant for unconnected server and client sockets. It is similar to TCPIP_TCP_SocketNetSet() that assigns a specific source interface for a socket . If localPort is 0 the stack will assign a unique local port
	TCPIP_TCP_ClientOpen	Opens a TCP socket as a client.
	TCPIP_TCP_Close	Disconnects an open socket and destroys the socket handle, releasing the associated resources.
	TCPIP_TCP_Connect	Connects a client socket .
	TCPIP_TCP_Discard	Discards any pending data in the TCP RX FIFO.

	TCPIP_TCP_Disconnect	Disconnects an open socket .
	TCPIP_TCP_FifoRxFreeGet	Determines how many bytes are free in the RX FIFO.
	TCPIP_TCP_FifoSizeAdjust	Adjusts the relative sizes of the RX and TX buffers.
	TCPIP_TCP_FifoTxFullGet	Determines how many bytes are pending in the TCP TX FIFO.
	TCPIP_TCP_Find	Searches for a byte in the TCP RX buffer.
	TCPIP_TCP_Flush	Immediately transmits all pending TX data.
	TCPIP_TCP_Get	Retrieves a single byte to a TCP socket .
	TCPIP_TCP_GetIsReady	Determines how many bytes can be read from the TCP RX buffer.
	TCPIP_TCP_IsConnected	Determines if a socket has an established connection.
	TCPIP_TCP_OptionsGet	Allows getting the options for a socket like: current Rx/Tx buffer size, etc
	TCPIP_TCP_OptionsSet	Allows setting options to a socket like adjust Rx/Tx buffer size, etc
	TCPIP_TCP_Peek	Peaks at one byte in the TCP RX FIFO without removing it from the buffer.
	TCPIP_TCP_Put	Writes a single byte to a TCP socket .
	TCPIP_TCP_PutIsReady	Determines how much free space is available in the TCP TX buffer.
	TCPIP_TCP_RemoteBind	Bind a socket to a remote address This function is meant for unconnected server and client sockets.
	TCPIP_TCP_ServerOpen	Opens a TCP socket as a server.
	TCPIP_TCP_SocketInfoGet	Obtains information about a currently open socket .
	TCPIP_TCP_SocketNetGet	Gets the MAC interface of an TCP socket
	TCPIP_TCP_SocketNetSet	Sets the interface for an TCP socket
	TCPIP_TCP_StringPut	Writes a null-terminated string from RAM to a TCP socket . The null-terminator is not copied to the socket .
	TCPIP_TCP_WasReset	Self-clearing semaphore indicating socket reset.

Macros

	Name	Description
	INVALID_SOCKET	The socket is invalid or could not be opened
	TCPIP_TCP_FifoRxFullGet	TCP Get Rx First In, First Out Full
	TCPIP_TCP_FifoTxFreeGet	TCP Get TX First In, First Out Free

Structures

	Name	Description
	TCP_MODULE_CONFIG	TCP layer configuration/initialization
	TCP_OPTION_LINGER_DATA	Linger options
	TCP_SOCKET_INFO	TCP Socket Information

Types

	Name	Description
	TCP_PORT	TCP Port
	TCP_SOCKET	TCP Socket

File Name

tcp.h

Company

Microchip Technology Incorporated

5.8.25.8.2 tcp_config.h

Transmission Control Protocol (TCP) Configuration file

This file contains the TCP module configuration options

Macros

	Name	Description
	<u>LOCAL_TCP_PORT_END_NUMBER</u>	Last port number for randomized local port number selection
	<u>LOCAL_TCP_PORT_START_NUMBER</u>	First port number for randomized local port number selection Use the dynamic port range defined by IANA consists of the 49152-65535 range and is meant for the selection of ephemeral ports (RFC 6056). Adjust to your needs but stay within the IANA range
	<u>TCP_AUTO_TRANSMIT_TIMEOUT_VAL</u>	Timeout before automatically transmitting unflushed data, ms; Default 40 ms
	<u>TCP_CLOSE_WAIT_TIMEOUT</u>	Timeout for the CLOSE_WAIT state, ms
	<u>TCP_DELAYED_ACK_TIMEOUT</u>	Timeout for delayed-acknowledgement algorithm, ms
	<u>TCP_FIN_WAIT_2_TIMEOUT</u>	Timeout for FIN WAIT 2 state, ms
	<u>TCP_KEEP_ALIVE_TIMEOUT</u>	Timeout for keep-alive messages when no traffic is sent, ms
	<u>TCP_MAX_RETRIES</u>	Maximum number of retransmission attempts
	<u>TCP_MAX_SEG_SIZE_RX_LOCAL</u>	TCP Maximum Segment Size for RX (MSS). This value is advertised during TCP connection establishment and the remote node should obey it. The value has to be set in such a way to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem). On Ethernet networks the standard value is 1460. On dial-up links, etc. the default values should be 536. Adjust these values according to your network. The stack will use the local... <u>more</u>
	<u>TCP_MAX_SEG_SIZE_RX_NON_LOCAL</u>	Max segment size, non local
	<u>TCP_MAX_SEG_SIZE_TX</u>	TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unhandlably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that we don't cause any TX buffer overflows. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.
	<u>TCP_MAX_SOCKETS</u>	The maximum number of sockets to create in the stack. When defining TCP_MAX_SOCKETS take into account the number of interfaces the stack is supporting
	<u>TCP_MAX_SYN_RETRIES</u>	Smaller than all other retries to reduce SYN flood DoS duration
	<u>TCP_MAX_UNACKED_KEEP_ALIVES</u>	Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection
	<u>TCP_SOCKET_DEFAULT_RX_SIZE</u>	default <u>socket</u> Rx buffer size
	<u>TCP_SOCKET_DEFAULT_TX_SIZE</u>	default <u>socket</u> Tx buffer size
	<u>TCP_START_TIMEOUT_VAL</u>	TCP Timeout and retransmit numbers All timeouts in milliseconds Timeout to retransmit unacked data, ms
	<u>TCP_TASK_TICK_RATE</u>	5 ms default rate
	<u>TCP_WINDOW_UPDATE_TIMEOUT_VAL</u>	Timeout before automatically transmitting a window update due to a <u>TCPIP_TCP_Get()</u> or <u>TCPIP_TCP_ArrayGet()</u> function call, ms

5.8.26 Telnet TCP/IP Stack Library

5.8.26.1 Introduction

Telnet TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the Telnet TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Telnet provides bidirectional, interactive communication between two nodes on the Internet or on a Local Area Network. The sTelnet code included with Microchip's TCP/IP stack is a demonstration of the structure of a Telnet application. This demo begins by listening for a Telnet connection. When a client attempts to make one, the demo will prompt the client for a username and password, and if the correct one is provided, will output and periodically refresh several values obtained from the demo board.

There are several changes that you may need to make to Telnet.c and/or Telnet.h to suit your application. All of the Telnet Public members can be re-defined in the application-specific section of TCPIPConfig.h. You may also wish to change some of the Telnet Internal Member strings, located in Telnet.c, to more accurately reflect your application. You will also need to modify the TelnetTask (see page 574) function to include the functionality you'd like. You may insert or change states in TelnetTask (see page 574) as needed.

5.8.26.2 Release Notes

MPLAB Harmony Version: v0.70b **Telnet TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.26.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.26.4 Using the Library

This topic describes the basic architecture of the Telnet TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [telnet.h](#)

The interface to the Telnet TCP/IP Stack library is defined in the "[telnet.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the Telnet TCP/IP Stack library should include "tcpip.h".

Library File:

The Telnet TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.26.4.1 Abstraction Model

This library provides the API of the Telnet TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Telnet Software Abstraction Block Diagram

Note: This image was not available at time of release and will be added in a future release of MPLAB Harmony.

5.8.26.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Telnet module.

Library Interface Section	Description
Data Types and Constants	This section provides various definitions describing this API

5.8.26.5 Configuring the Library

Macros

	Name	Description
	MAX_TELNET_CONNECTIONS	Maximum number of Telnet connections
	TELNET_PASSWORD	Default Telnet password

	TELNET_PORT	Unsecured Telnet port
	TELNET_USERNAME	Default Telnet user name
	TELNETS_PORT	SSL Secured Telnet port (ignored if TCPIP_STACK_USE_SSL_SERVER is undefined)

Description

The configuration of the Telnet TCP/IP Stack Library is based on the file `tcip_config.h`

This header file contains the configuration selection for the Telnet TCP/IP Stack Library. Based on the selections made, the Telnet TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the Telnet TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.26.5.1 MAX_TELNET_CONNECTIONS Macro**C**

```
#define MAX_TELNET_CONNECTIONS (2u)
```

Description

Maximum number of Telnet connections

5.8.26.5.2 TELNET_PASSWORD Macro**C**

```
#define TELNET_PASSWORD "microchip"
```

Description

Default Telnet password

5.8.26.5.3 TELNET_PORT Macro**C**

```
#define TELNET_PORT 23
```

Description

Unsecured Telnet port

5.8.26.5.4 TELNET_USERNAME Macro**C**

```
#define TELNET_USERNAME "admin"
```

Description

Default Telnet user name

5.8.26.5.5 TELNETS_PORT Macro

C

```
#define TELNETS_PORT 992
```

Description

SSL Secured Telnet port (ignored if TCPIP_STACK_USE_SSL_SERVER is undefined)

5.8.26.6 Building the Library

This section list the files that are available in the \src of the Telnet driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.26.7 Library Interface

Data Types and Constants

	Name	Description
	TELNET_MODULE_CONFIG	Telnet Configuration structure placeholder

Description

There are no user accessible functions in this interface. There is only a place holder for Telnet module configuration.

5.8.26.7.1 Data Types and Constants

5.8.26.7.1.1 TELNET_MODULE_CONFIG Structure

C

```
typedef struct {  
} TELNET_MODULE_CONFIG;
```

Description

Telnet Configuration structure placeholder

5.8.26.8 Files

Files

Name	Description
telnet.h	
telnet_config.h	Configuration file

Description

5.8.26.8.1 telnet.h

Telnet Server Module for Microchip TCP/IP Stack

-Listens on TCP Port 23

Structures

	Name	Description
	TELNET_MODULE_CONFIG	Telnet Configuration structure placeholder

5.8.26.8.2 telnet_config.h

Telnet Configuration file

This file contains the Telnet module configuration options

Macros

	Name	Description
	MAX_TELNET_CONNECTIONS	Maximum number of Telnet connections
	TELNET_PASSWORD	Default Telnet password
	TELNET_PORT	Unsecured Telnet port
	TELNET_USERNAME	Default Telnet user name
	TELNETS_PORT	SSL Secured Telnet port (ignored if TCPIP_STACK_USE_SSL_SERVER is undefined)

5.8.27 TFTP TCP/IP Stack Library

5.8.27.1 Introduction

Trivial File Transfer Protocol (TFTP) TCP/IP Stack Library

for

Microchip Microcontrollers

This library provides the API of the TFTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The Trivial File Transfer Protocol provides unreliable upload and download services to applications connected to the UDP-based TFTP server.

5.8.27.2 Release Notes

MPLAB Harmony Version: v0.70b TFTP TCP/IP Stack Library Version : 1.0 Release Date: 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.27.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.27.4 Using the Library

This topic describes the basic architecture of the TFTP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [tftp.h](#)

The interface to the TFTP TCP/IP Stack library is defined in the "[tftp.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the [tftp.h](#) TCP/IP Stack library should include "tcpip.h".

Library File:

The TFTP TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

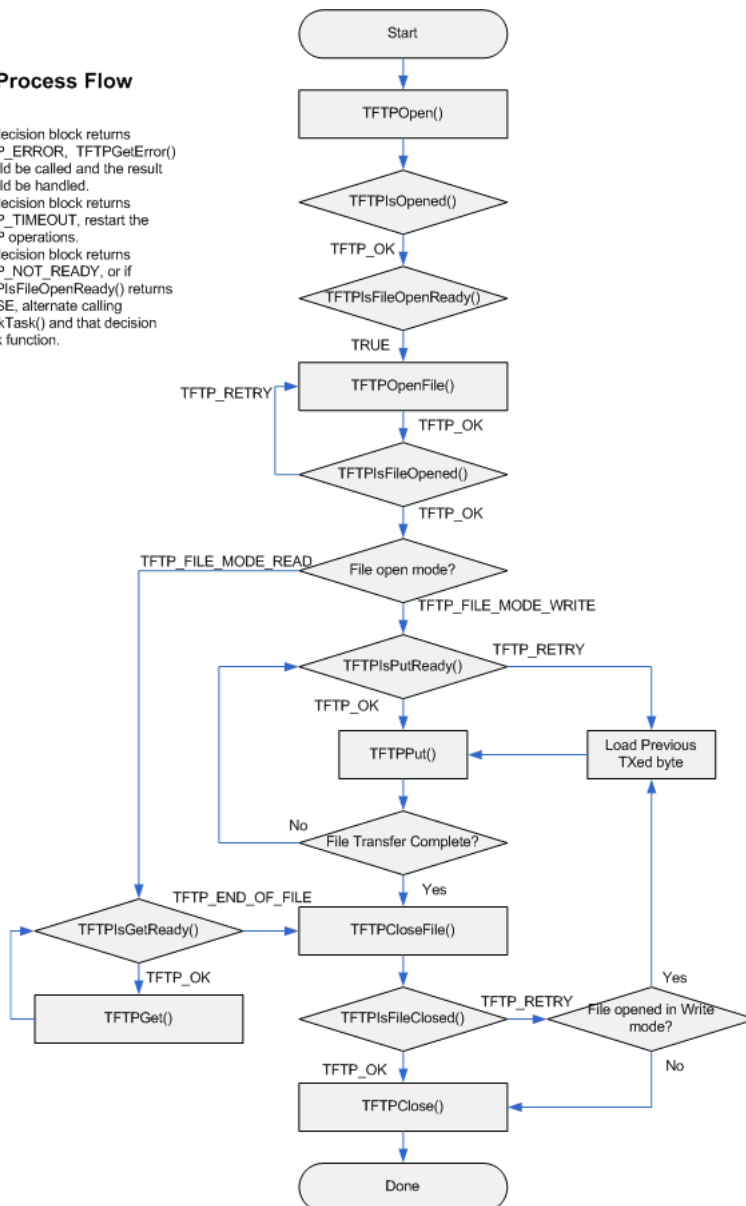
5.8.27.4.1 Abstraction Model

This library provides the API of the TFTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description**TFTP Software Abstraction Block Diagram**

TFTP Process Flow Notes

- If a decision block returns `TFTP_ERROR`, `TFTPGetError()` should be called and the result should be handled.
- If a decision block returns `TFTP_TIMEOUT`, restart the TFTP operations.
- If a decision block returns `TFTP_NOT_READY`, or if `TFTPisFileOpenReady()` returns `FALSE`, alternate calling `StackTask()` and that decision block function.



5.8.27.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the TFTP module.

Library Interface Section	Description
General Functions	This section provides general interface routines to TFTP
Data Types and Constants	This section provides various definitions describing this API

5.8.27.5 Configuring the Library

Macros

	Name	Description
	TFTP_ARP_TIMEOUT_VAL	Number of seconds to wait before declaring TIMEOUT error on Put, seconds
	TFTP_BLOCK_SIZE	The size of a TFTP block - 512 bytes
	TFTP_BLOCK_SIZE_MSB	The MSB of the TFTP_BLOCK_SIZE
	TFTP_CLIENT_PORT	The TFTP Client port - a unique port on this device
	TFTP_GET_TIMEOUT_VAL	Number of seconds to wait before declaring TIMEOUT error on Get, seconds.
	TFTP_MAX_RETRIES	Number of attempts before declaring TIMEOUT error.
	TFTP_SERVER_PORT	The TFTP Server Port

Description

The configuration of the TFTP TCP/IP Stack Library is based on the file `tcip_config.h`

This header file contains the configuration selection for the TFTP TCP/IP Stack Library. Based on the selections made, the TFTP TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the TFTP TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.27.5.1 TFTP_ARP_TIMEOUT_VAL Macro

C

```
#define TFTP_ARP_TIMEOUT_VAL (3uL)
```

Description

Number of seconds to wait before declaring TIMEOUT error on Put, seconds

5.8.27.5.2 TFTP_BLOCK_SIZE Macro

C

```
#define TFTP_BLOCK_SIZE (0x200L)
```

Description

The size of a TFTP block - 512 bytes

5.8.27.5.3 TFTP_BLOCK_SIZE_MSB Macro

C

```
#define TFTP_BLOCK_SIZE_MSB (0x02u)
```

Description

The MSB of the [TFTP_BLOCK_SIZE](#)

5.8.27.5.4 TFTP_CLIENT_PORT Macro

C

```
#define TFTP_CLIENT_PORT 65352L
```

Description

The TFTP Client port - a unique port on this device

5.8.27.5.5 TFTP_GET_TIMEOUT_VAL Macro

C

```
#define TFTP_GET_TIMEOUT_VAL (3uL)
```

Description

Number of seconds to wait before declaring TIMEOUT error on Get, seconds.

5.8.27.5.6 TFTP_MAX_RETRIES Macro

C

```
#define TFTP_MAX_RETRIES (3u)
```

Description

Number of attempts before declaring TIMEOUT error.

5.8.27.5.7 TFTP_SERVER_PORT Macro

C

```
#define TFTP_SERVER_PORT (69L)
```

Description

The TFTP Server Port

5.8.27.6 Building the Library

This section list the files that are available in the \src of the TFTP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.27.7 Library Interface

Data Types and Constants

	Name	Description
	<u>TFTP_UPLOAD_COMPLETE</u>	Status codes for <u>TCPIP_TFTP_UploadStatusGet()</u> function. Zero means upload success, >0 means working and <0 means fatal error.
	<u>TFTP_UPLOAD_CONNECT</u>	This is macro TFTP_UPLOAD_CONNECT.
	<u>TFTP_UPLOAD_CONNECT_TIMEOUT</u>	This is macro TFTP_UPLOAD_CONNECT_TIMEOUT.

	TFTP_UPLOAD_GET_DNS	This is macro TFTP_UPLOAD_GET_DNS.
	TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT	This is macro TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT.
	TFTP_UPLOAD_RESOLVE_HOST	This is macro TFTP_UPLOAD_RESOLVE_HOST.
	TFTP_UPLOAD_SEND_DATA	This is macro TFTP_UPLOAD_SEND_DATA.
	TFTP_UPLOAD_SEND_FILENAME	This is macro TFTP_UPLOAD_SEND_FILENAME.
	TFTP_UPLOAD_SERVER_ERROR	This is macro TFTP_UPLOAD_SERVER_ERROR.
	TFTP_UPLOAD_WAIT_FOR_CLOSURE	This is macro TFTP_UPLOAD_WAIT_FOR_CLOSURE.
	TFTPClose	Macro: void TFTPClose(void) Closes TFTP client socket .
	TFTPGetError	Macro: uint16_t TFTPGetError(void) Returns previously saved error code.
	TFTPisFileOpenReady	Macro: bool TFTPisFileOpenReady(void) Checks to see if it is okay to send TFTP file open request to remote server.
	TFTP_ACCESS_ERROR	Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError() .
	TFTP_CHUNK_DESCRIPTOR	This is type TFTP_CHUNK_DESCRIPTOR.
	TFTP_FILE_MODE	File open mode as used by TFTPFileOpen().
	TFTP_RESULT	Enum. of results returned by most of the TFTP functions.

Functions

	Name	Description
	TCPIP_TFTP_DataByteGet	This is function TCPIP_TFTP_DataByteGet.
	TCPIP_TFTP_DataBytePut	This is function TCPIP_TFTP_DataBytePut.
	TCPIP_TFTP_DataIsGetReady	This is function TCPIP_TFTP_DataIsGetReady.
	TCPIP_TFTP_FileClose	This is function TCPIP_TFTP_FileClose.
	TCPIP_TFTP_FileIsClosed	This is function TCPIP_TFTP_FileIsClosed.
	TCPIP_TFTP_FileIsOpen	This is function TCPIP_TFTP_FileIsOpen.
	TCPIP_TFTP_FragRAMFileUploadToHost	This is function TCPIP_TFTP_FragRAMFileUploadToHost.
	TCPIP_TFTP_IsOpen	This is function TCPIP_TFTP_IsOpen.
	TCPIP_TFTP_IsReadyForPut	This is function TCPIP_TFTP_IsReadyForPut.
	TCPIP_TFTP_Open	This is function TCPIP_TFTP_Open.
	TCPIP_TFTP_RAMFileUploadToHost	This is function TCPIP_TFTP_RAMFileUploadToHost.
	TCPIP_TFTP_UploadStatusGet	This is function TCPIP_TFTP_UploadStatusGet.
	TFTPOpenFile	This is function TFTPOpenFile.

Description

This section describes the Application Programming Interface (API) functions of the TFTP TCP/IP Stack.

Refer to each section for a detailed description.

5.8.27.7.1 Functions

5.8.27.7.1.1 TCPIP_TFTP_DataByteGet Function

C

```
uint8_t TCPIP_TFTP_DataByteGet();
```

Description

This is function TCPIP_TFTP_DataByteGet.

5.8.27.7.1.2 TCPIP_TFTP_DataBytePut Function**C**

```
void TCPIP_TFTP_DataBytePut(  
    uint8_t c  
);
```

Description

This is function TCPIP_TFTP_DataBytePut.

5.8.27.7.1.3 TCPIP_TFTP_DataIsGetReady Function**C**

```
TFTP_RESULT TCPIP_TFTP_DataIsGetReady();
```

Description

This is function TCPIP_TFTP_DataIsGetReady.

5.8.27.7.1.4 TCPIP_TFTP_FileClose Function**C**

```
void TCPIP_TFTP_FileClose();
```

Description

This is function TCPIP_TFTP_FileClose.

5.8.27.7.1.5 TCPIP_TFTP_FileIsClosed Function**C**

```
TFTP_RESULT TCPIP_TFTP_FileIsClosed();
```

Description

This is function TCPIP_TFTP_FileIsClosed.

5.8.27.7.1.6 TCPIP_TFTP_FileIsOpen Function**C**

```
TFTP_RESULT TCPIP_TFTP_FileIsOpen();
```

Description

This is function TCPIP_TFTP_FileIsOpen.

5.8.27.7.1.7 TCPIP_TFTP_FragRAMFileUploadToHost Function**C**

```
void TCPIP_TFTP_FragRAMFileUploadToHost(  
    const uint8_t * vRemoteHost,  
    const uint8_t * vFilename,  
    TFTP_CHUNK_DESCRIPTOR * vFirstChunkDescriptor  
);
```

Description

This is function TCPIP_TFTP_FragRAMFileUploadToHost.

5.8.27.7.1.8 TCPIP_TFTP_IsOpen Function**C**

```
TFTP_RESULT TCPIP_TFTP_IsOpen();
```

Description

This is function TCPIP_TFTP_IsOpen.

5.8.27.7.1.9 TCPIP_TFTP_IsReadyForPut Function**C**

```
TFTP_RESULT TCPIP_TFTP_IsReadyForPut();
```

Description

This is function TCPIP_TFTP_IsReadyForPut.

5.8.27.7.1.10 TCPIP_TFTP_Open Function**C**

```
void TCPIP_TFTP_Open(  
    IPV4_ADDR * host  
);
```

Description

This is function TCPIP_TFTP_Open.

5.8.27.7.1.11 TCPIP_TFTP_RAMFileUploadToHost Function**C**

```
void TCPIP_TFTP_RAMFileUploadToHost(  
    const uint8_t * vRemoteHost,  
    const uint8_t * vFilename,  
    uint8_t * vData,  
    uint16_t wDataLength  
);
```

Description

This is function TCPIP_TFTP_RAMFileUploadToHost.

5.8.27.7.1.12 TCPIP_TFTP_UploadStatusGet Function**C**

```
int8_t TCPIP_TFTP_UploadStatusGet();
```

Description

This is function TCPIP_TFTP_UploadStatusGet.

5.8.27.7.1.13 TFTPOpenFile Function

C

```
void TFTPOpenFile(  
    const uint8_t * fileName,  
    TFTP_FILE_MODE mode  
);
```

Description

This is function TFTPOpenFile.

5.8.27.7.2 Data Types and Constants

5.8.27.7.2.1 TFTP_UPLOAD_COMPLETE Macro

C

```
#define TFTP_UPLOAD_COMPLETE 0
```

Description

Status codes for [TCPIP_TFTP_UploadStatusGet\(\)](#) function. Zero means upload success, >0 means working and <0 means fatal error.

5.8.27.7.2.2 TFTP_UPLOAD_CONNECT Macro

C

```
#define TFTP_UPLOAD_CONNECT 3
```

Description

This is macro TFTP_UPLOAD_CONNECT.

5.8.27.7.2.3 TFTP_UPLOAD_CONNECT_TIMEOUT Macro

C

```
#define TFTP_UPLOAD_CONNECT_TIMEOUT -2
```

Description

This is macro TFTP_UPLOAD_CONNECT_TIMEOUT.

5.8.27.7.2.4 TFTP_UPLOAD_GET_DNS Macro

C

```
#define TFTP_UPLOAD_GET_DNS 1
```

Description

This is macro TFTP_UPLOAD_GET_DNS.

5.8.27.7.2.5 TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT Macro

C

```
#define TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT -1
```

Description

This is macro TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT.

5.8.27.7.2.6 TFTP_UPLOAD_RESOLVE_HOST Macro

C

```
#define TFTP_UPLOAD_RESOLVE_HOST 2
```

Description

This is macro TFTP_UPLOAD_RESOLVE_HOST.

5.8.27.7.2.7 TFTP_UPLOAD_SEND_DATA Macro

C

```
#define TFTP_UPLOAD_SEND_DATA 5
```

Description

This is macro TFTP_UPLOAD_SEND_DATA.

5.8.27.7.2.8 TFTP_UPLOAD_SEND_FILENAME Macro

C

```
#define TFTP_UPLOAD_SEND_FILENAME 4
```

Description

This is macro TFTP_UPLOAD_SEND_FILENAME.

5.8.27.7.2.9 TFTP_UPLOAD_SERVER_ERROR Macro

C

```
#define TFTP_UPLOAD_SERVER_ERROR -3
```

Description

This is macro TFTP_UPLOAD_SERVER_ERROR.

5.8.27.7.2.10 TFTP_UPLOAD_WAIT_FOR_CLOSURE Macro

C

```
#define TFTP_UPLOAD_WAIT_FOR_CLOSURE 6
```

Description

This is macro TFTP_UPLOAD_WAIT_FOR_CLOSURE.

5.8.27.7.2.11 TFTPclose Macro

C

```
#define TFTPclose(void) TCPIP_UDP_Close(_tftpSocket)
```

Description

Macro: void TFTPclose(void)

Closes TFTP client [socket](#).

Preconditions

[TCPIP_TFTP_Open](#) is already called and [TCPIP_TFTP_IsOpen\(\)](#) returned TFTP_OK.

Returns

None

Side Effects

None

Remarks

Once closed, application must do [TCPIP_TFTP_Open](#) to perform any new TFTP operations.

If TFTP server does not change during application life-time, one may not need to call TFTP_CLOSE and keep TFTP [socket](#) open.

5.8.27.7.2.12 TFTPGetError Macro

C

```
#define TFTPGetError (_tftpError)
```

Description

Macro: uint16_t TFTPGetError(void)

Returns previously saved error code.

Preconditions

One of the TFTP function returned with TFTP_ERROR result.

Returns

Error code as returned by remote server. Application may use [TFTP_ACCESS_ERROR](#) enum. to decode standard error code.

Side Effects

None

Remarks

None

5.8.27.7.2.13 TFTP_IsFileOpenReady Macro

C

```
#define TFTP_IsFileOpenReady UDPIsPutReady(_tftpSocket)
```

Description

Macro: bool TFTP_IsFileOpenReady(void)

Checks to see if it is okay to [send](#) TFTP file open request to remote server.

Preconditions

[TCPIP_TFTP_Open](#) is already called and [TCPIP_TFTP_IsOpen\(\)](#) returned TFTP_OK.

Returns

true, if it is ok to call [TFTPOpenFile\(\)](#) false, if otherwise.

Side Effects

None

Remarks

None

5.8.27.7.2.14 TFTP_ACCESS_ERROR Type

C

```
typedef enum _TFTP_ACCESS_ERROR TFTP_ACCESS_ERROR;
```

Description

Standard error codes as defined by TFTP spec. Use to decode value retuned by [TFTPGetError\(\)](#).

5.8.27.7.2.15 TFTP_CHUNK_DESCRIPTOR Structure

C

```
typedef struct {
    uint8_t * vDataPointer;
    uint16_t wDataLength;
} TFTP_CHUNK_DESCRIPTOR;
```

Description

This is type TFTP_CHUNK_DESCRIPTOR.

5.8.27.7.2.16 TFTP_FILE_MODE Type

C

```
typedef enum _TFTP_FILE_MODE TFTP_FILE_MODE;
```

Description

File open mode as used by TFTPFileOpen().

5.8.27.7.2.17 TFTP_RESULT Type

C

```
typedef enum _TFTP_RESULT TFTP_RESULT;
```

Description

Enum. of results returned by most of the TFTP functions.

5.8.27.8 Files

Files

Name	Description
tftpc.h	
tftpc_config.h	(TFTP) Configuration file

Description

5.8.27.8.1 tftpc.h

TFTP Client module for Microchip TCP/IP Stack

Functions

	Name	Description
	TCPIP_TFTP_DataByteGet	This is function TCPIP_TFTP_DataByteGet.
	TCPIP_TFTP_DataBytePut	This is function TCPIP_TFTP_DataBytePut.
	TCPIP_TFTP_DatalsGetReady	This is function TCPIP_TFTP_DatalsGetReady.
	TCPIP_TFTP_FileClose	This is function TCPIP_TFTP_FileClose.
	TCPIP_TFTP_FileIsClosed	This is function TCPIP_TFTP_FileIsClosed.
	TCPIP_TFTP_FileIsOpen	This is function TCPIP_TFTP_FileIsOpen.
	TCPIP_TFTP_FragRAMFileUploadToHost	This is function TCPIP_TFTP_FragRAMFileUploadToHost.
	TCPIP_TFTP_IsOpen	This is function TCPIP_TFTP_IsOpen.
	TCPIP_TFTP_IsReadyForPut	This is function TCPIP_TFTP_IsReadyForPut.
	TCPIP_TFTP_Open	This is function TCPIP_TFTP_Open.
	TCPIP_TFTP_RAMFileUploadToHost	This is function TCPIP_TFTP_RAMFileUploadToHost.
	TCPIP_TFTP_UploadStatusGet	This is function TCPIP_TFTP_UploadStatusGet.
	TFTPOpenFile	This is function TFTPOpenFile.

Macros

	Name	Description
	TFTP_MAX_RETRIES	Number of attempts before declaring TIMEOUT error.
	TFTP_UPLOAD_COMPLETE	Status codes for TCPIP_TFTP_UploadStatusGet() function. Zero means upload success, >0 means working and <0 means fatal error.
	TFTP_UPLOAD_CONNECT	This is macro TFTP_UPLOAD_CONNECT.
	TFTP_UPLOAD_CONNECT_TIMEOUT	This is macro TFTP_UPLOAD_CONNECT_TIMEOUT.
	TFTP_UPLOAD_GET_DNS	This is macro TFTP_UPLOAD_GET_DNS.
	TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT	This is macro TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT.
	TFTP_UPLOAD_RESOLVE_HOST	This is macro TFTP_UPLOAD_RESOLVE_HOST.
	TFTP_UPLOAD_SEND_DATA	This is macro TFTP_UPLOAD_SEND_DATA.
	TFTP_UPLOAD_SEND_FILENAME	This is macro TFTP_UPLOAD_SEND_FILENAME.
	TFTP_UPLOAD_SERVER_ERROR	This is macro TFTP_UPLOAD_SERVER_ERROR.
	TFTP_UPLOAD_WAIT_FOR_CLOSURE	This is macro TFTP_UPLOAD_WAIT_FOR_CLOSURE.
	TFTPClose	Macro: void TFTPClose(void) Closes TFTP client socket .
	TFTPGetError	Macro: uint16_t TFTPGetError(void) Returns previously saved error code.
	TFTPisFileOpenReady	Macro: bool TFTPisFileOpenReady(void) Checks to see if it is okay to send TFTP file open request to remote server.

Structures

	Name	Description
	TFTP_CHUNK_DESCRIPTOR	This is type TFTP_CHUNK_DESCRIPTOR.

Types

	Name	Description
	TFTP_ACCESS_ERROR	Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError() .
	TFTP_FILE_MODE	File open mode as used by TFTPFileOpen() .
	TFTP_RESULT	Enum. of results returned by most of the TFTP functions.

5.8.27.8.2 tftpc_config.h

Trivial File Transfer Protocol (TFTP) Configuration file

This file contains the TFTP module configuration options

Macros

	Name	Description
	TFTP_ARP_TIMEOUT_VAL	Number of seconds to wait before declaring TIMEOUT error on Put, seconds
	TFTP_BLOCK_SIZE	The size of a TFTP block - 512 bytes
	TFTP_BLOCK_SIZE_MSB	The MSB of the TFTP_BLOCK_SIZE
	TFTP_CLIENT_PORT	The TFTP Client port - a unique port on this device
	TFTP_GET_TIMEOUT_VAL	Number of seconds to wait before declaring TIMEOUT error on Get, seconds.
	TFTP_SERVER_PORT	The TFTP Server Port

5.8.28 UDP TCP/IP Stack Library**5.8.28.1 Introduction****UDP TCP/IP Stack Library**

for

Microchip Microcontrollers

This library provides the API of the UDP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

UDP is a standard transport layer protocol described in RFC 768. It provides fast but unreliable data-gram based transfers over networks, and forms the foundation SNTP, SNMP, DNS, and many other protocol standards.

5.8.28.2 Release Notes

MPLAB Harmony Version: v0.70b **UDP TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.28.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.28.4 Using the Library

This topic describes the basic architecture of the UDP TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header File: [udp.h](#)

The interface to the UDP TCP/IP Stack library is defined in the "[udp.h](#)" header file. This file is included by the "tcpip.h" file. Any C language source (.c) file that uses the UDP TCP/IP Stack library should include "tcpip.h".

Library File:

The UDP TCPIP Stack library archive (.a) file installed with MPLAB Harmony.

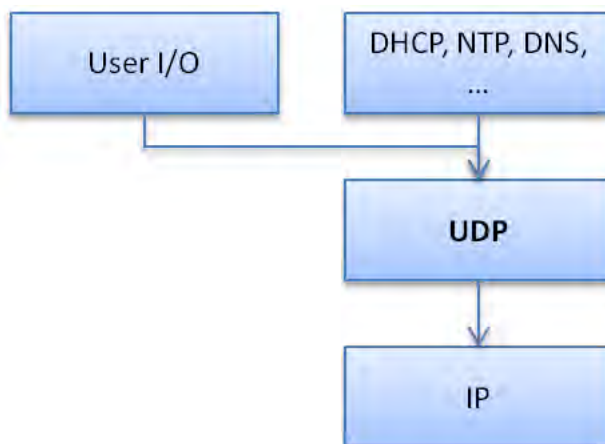
Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.28.4.1 Abstraction Model

This library provides the API of the UDP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description**UDP Software Abstraction Block Diagram**

This module provides software abstraction of the UDP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

Typical UDP Implementation

5.8.28.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the UDP module.

Library Interface Section	Description
Socket Management Functions	Routines for Managing UDP Sockets
Transmit Data Transfer Functions	Routines for Managing Outgoing Data Transmissions
Receive Data Transfer Functions	Routines for Managing Incoming Data Transmissions
Data Types and Constants	This section provides various definitions describing This API

5.8.28.4.3 How the Library Works

Connections over UDP should be thought of as data-gram based transfers. Each packet is a separate entity, the application should expect some packets to arrive out-of-order or even fail to reach the destination node. This is in contrast to TCP, in which the connection is thought of as a stream and network errors are automatically corrected. These tradeoffs in reliability are made for an increase in throughput. In general, UDP transfers operate 2 to 3 times faster than those made over TCP.

Since UDP is packet-oriented, each packet must be dealt with in its entirety by your application before returning to the main stack loop. When a packet is received, your application will be called to handle it. This packet will no longer be available the next time your application is called, so you must either perform all necessary processing or copy the data elsewhere before returning. When transmitting a packet, your application must build and transmit the complete packet in one cycle.

5.8.28.4.3.1 Core Functionality

The UDP flow diagram below provides an overview for the use of the UDP module:

Note: This image was not available at time of release and will be added in a future release of MPLAB Harmony.

Sockets are opened using `UDPOpen`. This function can either open a listening [socket](#) to wait for incoming segments, or can make a client connection to a remote node. When making a client connection, you will need to perform any required DNS and/or ARP resolution using those modules directly before invoking `UDPOpen`.

Once the [socket](#) is opened, you can immediately begin transmitting data. To transmit a segment, call [UDP_IsPutReady](#) to determine how many bytes can be written and to designate a currently active [socket](#). Then, use any of the [TCPIP_UDP_Put](#) family of functions to write data to the [socket](#). Once all data has been written, call [TCPIP_UDP_Flush](#) to build and transmit the packet. This sequence must be accomplished all in one step. If your application returns to the main stack loop after calling [TCPIP_UDP_Put](#) but before calling [TCPIP_UDP_Flush](#), the data may be lost or the module may behave unpredictably.

5.8.28.5 Configuring the Library

Macros

	Name	Description
	UDP_LOCAL_PORT_END_NUMBER	Last port number for randomized local port number selection
	UDP_LOCAL_PORT_START_NUMBER	The dynamic port range defined by IANA consists of the 49152-65535 range and is meant for the selection of ephemeral ports (RFC 6056). Adjust to your needs but stay within the IANA range First port number for randomized local port number selection
	UDP_MAX_SOCKETS	global number of UDP sockets created dynamically
	UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT	the maximum number of packets that can be queued by an UDP socket at a certain time. For sockets that need to transfer a lot of data (iperf, for example), especially on slow connections this limit prevents running out of memory because the MAC transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the UDP_SOCKET_DEFAULT_TX_SIZE , the connection speed and the amount of memory available to the stack.
	UDP_SOCKET_DEFAULT_TX_SIZE	default socket Tx buffer size
	UDP_SOCKET_POOL_BUFFER_SIZE	size of the buffers in the UDP pool any UDP socket that is enabled to use the pool and has the TX size <= than this size can use a buffer from the pool
	UDP_SOCKET_POOL_BUFFERS	use 0 to disable the feature
	UDP_USE_RX_CHECKSUM	check incoming packets to have proper checksum
	UDP_USE_TX_CHECKSUM	This slows UDP TX performance by nearly 50%, except when using the ENCX24J600, which has a super fast DMA and incurs virtually no speed penalty.

Description

The configuration of the UDP TCP/IP Stack Library is based on the file `sys_config.h`

This header file contains the configuration selection for the UP TCP/IP Stack Library. Based on the selections made, the UDP TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the UDP TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.28.5.1 UDP_LOCAL_PORT_END_NUMBER Macro

C

```
#define UDP_LOCAL_PORT_END_NUMBER (65535)
```

Description

Last port number for randomized local port number selection

5.8.28.5.2 UDP_LOCAL_PORT_START_NUMBER Macro

C

```
#define UDP_LOCAL_PORT_START_NUMBER (49152)
```

Description

The dynamic port range defined by IANA consists of the 49152-65535 range and is meant for the selection of ephemeral ports (RFC 6056). Adjust to your needs but stay within the IANA range First port number for randomized local port number selection

5.8.28.5.3 UDP_MAX_SOCKETS Macro

C

```
#define UDP_MAX_SOCKETS (10)
```

Description

global number of UDP sockets created dynamically

5.8.28.5.4 UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT Macro

C

```
#define UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT 3
```

Description

the maximum number of packets that can be queued by an UDP [socket](#) at a certain time. For sockets that need to transfer a lot of data (iperf, for example), especially on slow connections this limit prevents running out of memory because the MAC transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the [UDP_SOCKET_DEFAULT_TX_SIZE](#), the connection speed and the amount of memory available to the stack.

5.8.28.5.5 UDP_SOCKET_DEFAULT_TX_SIZE Macro

C

```
#define UDP_SOCKET_DEFAULT_TX_SIZE 512
```

Description

default [socket](#) Tx buffer size

5.8.28.5.6 UDP_SOCKET_POOL_BUFFER_SIZE Macro

C

```
#define UDP_SOCKET_POOL_BUFFER_SIZE 512 // size of the buffers in the UDP pool
```

Description

size of the buffers in the UDP pool any UDP [socket](#) that is enabled to use the pool and has the TX size <= than this size can use a buffer from the pool

5.8.28.5.7 UDP_SOCKET_POOL_BUFFERS Macro

C

```
#define UDP_SOCKET_POOL_BUFFERS 4           // use 0 to disable the feature
```

Description

use 0 to disable the feature

5.8.28.5.8 UDP_USE_RX_CHECKSUM Macro

C

```
#define UDP_USE_RX_CHECKSUM
```

Description

check incoming packets to have proper checksum

5.8.28.5.9 UDP_USE_TX_CHECKSUM Macro

C

```
#define UDP_USE_TX_CHECKSUM
```

Description

This slows UDP TX performance by nearly 50%, except when using the ENC24J600, which has a super fast DMA and incurs virtually no speed penalty.

5.8.28.6 Building the Library

This section list the files that are available in the \src of the UDP driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

- udp.c

5.8.28.7 Library Interface

Data Types and Constants

	Name	Description
	INVALID_UDP_SOCKET	Indicates a UDP socket that is not valid
	TCPIP_UDP_IsOpened	backward compatibility call
	UDP_MODULE_CONFIG	UDP layer configuration/initialization
	UDP_PORT	Stores a UDP Port Number
	UDP_SOCKET	Provides a handle to a UDP Socket
	UDP_SOCKET_BCAST_TYPE	This is type UDP_SOCKET_BCAST_TYPE .
	UDP_SOCKET_INFO	Information about a socket
	UDP_SOCKET_OPTION	UDP socket options

Receive Data Transfer Functions

	Name	Description
≡	TCPIP_UDP_Discard	Discards any remaining RX data from a UDP socket .
≡	TCPIP_UDP_Get	Reads a byte from the UDP socket .
≡	TCPIP_UDP_GetIsReady	Determines how many bytes can be read from the UDP socket .
≡	TCPIP_UDP_RxOffsetSet	Moves the pointer within the RX buffer.

Socket Management Functions

	Name	Description
≡	TCPIP_UDP_BcastIPv4AddressSet	Sets the broadcast IP address of a socket . Allows an UDP socket to send broadcasts.
≡	TCPIP_UDP_Bind	Bind a socket to a local address and port. This function is meant for client sockets. It assigns a specific source address and port for a socket .
≡	TCPIP_UDP_ClientOpen	Opens a UDP socket as a client.
≡	TCPIP_UDP_Close	Closes a UDP socket and frees the handle.
≡	TCPIP_UDP_DestinationIPAddressSet	Sets the destination IP address of a socket .
≡	TCPIP_UDP_IsConnected	Determines if a socket has an established connection.
≡	TCPIP_UDP_OptionsGet	Allows getting the options for a socket like: current Rx/Tx buffer size, etc
≡	TCPIP_UDP_OptionsSet	Allows setting options to a socket like adjust Rx/Tx buffer size, etc
≡	TCPIP_UDP_RemoteBind	Bind a socket to a remote address. This function is meant for server sockets.
≡	TCPIP_UDP_ServerOpen	Opens a UDP socket as a server.
≡	TCPIP_UDP_SocketInfoGet	Points handle at socket info for socket hUDP
≡	TCPIP_UDP_SocketNetGet	Gets the network interface of an UDP socket .
≡	TCPIP_UDP_SocketNetSet	Sets the network interface for an UDP socket .
≡	TCPIP_UDP_SourceIPAddressSet	Sets the source IP address of a socket .
≡	TCPIP_UDP_TxOffsetSet	Moves the pointer within the TX buffer.

Transmit Data Transfer Functions

	Name	Description
≡	TCPIP_UDP_ArrayGet	Reads an array of bytes from the UDP socket .
≡	TCPIP_UDP_Flush	Transmits all pending data in a UDP socket .
≡	TCPIP_UDP_Put	Writes a byte to the UDP socket .
≡	TCPIP_UDP_StringPut	Writes null-terminated string to the UDP socket .
≡	TCPIP_UDP_TxCountGet	Returns the amount of bytes written into the UDP socket .
≡	TCPIP_UDP_TxPutIsReady	Determines how many bytes can be written to the UDP socket .
≡	TCPIP_UDP_ArrayPut	Writes an array of bytes to the UDP socket .
≡	UDP_IsPutReady	Determines how many bytes can be written to the UDP socket .

Description

This section describes the Application Programming Interface (API) functions of the UDP TCP/IP Stack Library.

Refer to each section for a detailed description.

5.8.28.7.1 Socket Management Functions

5.8.28.7.1.1 TCPIP_UDP_BcastIPv4AddressSet Function

C

```
bool TCPIP_UDP_BcastIPv4AddressSet(  
    UDP_SOCKET s,  
    UDP_SOCKET_BCAST_TYPE bcastType,  
    TCPIP_NET_HANDLE hNet  
);
```

Description

It sets the broadcast address for the [socket](#)

Preconditions

UDP initialized UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). s - valid [socket](#)

Parameters

Parameters	Description
s	the UDP socket
bcastType	Type of broadcast
hNet	handle of an interface to use for the network directed broadcast Not used for network limited broadcast

Returns

True if success False otherwise

Remarks

This function allows changing of the destination IPv4 address dynamically.

However, the call will fail if the [socket](#) was previously set to broadcast using the [TCPIP_UDP_OptionsSet](#) call. [TCPIP_UDP_OptionsSet](#) takes precedence.

5.8.28.7.1.2 TCPIP_UDP_Bind Function

C

```
bool TCPIP_UDP_Bind(  
    UDP_SOCKET hUDP,  
    IP_ADDRESS_TYPE addType,  
    UDP_PORT localPort,  
    IP_MULTI_ADDRESS* localAddress  
);
```

Description

Sockets don't need specific binding, it is done automatically by the stack However, specific binding can be requested using these functions. Works for both client and server sockets.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	The socket to bind .
addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
localPort	The local port to bind to.

localAddress	Local address to use.

Returns

True if success False otherwise

Remarks

If localAddress == 0 the local address of the [socket](#) won't be changed.

If localPort is 0 the stack will assign a unique local port

5.8.28.7.1.3 TCPIP_UDP_ClientOpen Function

C

```
UDP_SOCKET TCPIP_UDP_ClientOpen(  
    IP_ADDRESS_TYPE addType,  
    UDP_PORT remotePort,  
    IP_MULTI_ADDRESS* remoteAddress  
);
```

Description

Provides a unified method for opening UDP client sockets.

Preconditions

UDP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
UDP_PORT remotePort	The remote UDP port to which a connection should be made. The local port for client sockets will be automatically picked by the UDP module.
IP_MULTI_ADDRESS* remoteAddress	The remote address to connect to.

Returns

[INVALID_SOCKET](#) - No sockets of the specified type were available to be opened.

Otherwise - A [UDP_SOCKET](#) handle. Save this handle and use it when calling all other UDP APIs.

Remarks

IP_ADDRESS_TYPE_ANY is not supported!

5.8.28.7.1.4 TCPIP_UDP_Close Function

C

```
void TCPIP_UDP_Close(  
    UDP_SOCKET hUDP  
);
```

Description

Closes a UDP [socket](#) and frees the handle. Call this function to release a [socket](#) and return it to the pool for use by future communications.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	The socket handle to be released.

Returns

None

5.8.28.7.1.5 TCPIP_UDP_DestinationIPAddressSet Function**C**

```
bool TCPIP_UDP_DestinationIPAddressSet(  
    UDP_SOCKET s,  
    IP_ADDRESS_TYPE addType,  
    IP_MULTI_ADDRESS* remoteAddress  
);
```

Description

- It sets the IP destination address This allows changing the destination IPv4 address dynamically.

Preconditions

UDP initialized UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). s - valid [socket](#) remoteAddress - valid address pointer

Parameters

Parameters	Description
s	the UDP socket
addType	Type of address: IPv4/IPv6
remoteAddress	pointer to an address to use

Returns

true if success false otherwise

Remarks

The call will fail if the [socket](#) was previously set to broadcast using the [TCPIP_UDP_OptionsSet](#) call. [TCPIP_UDP_OptionsSet](#) takes precedence.

The call will fail if remoteAddress is 0. The destination IP address will not be changed.

5.8.28.7.1.6 TCPIP_UDP_IsConnected Function**C**

```
bool TCPIP_UDP_IsConnected(  
    UDP_SOCKET hUDP  
);
```

Description

This function determines if a [socket](#) has an established connection to a remote node . Call this function after calling [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#) to determine when the connection is set up and ready for use.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	The socket to check.

Return Values

Return Values	Description
true	The socket has been opened and ARP has been resolved.
false	The socket is not currently connected.

Remarks

None

5.8.28.7.1.7 TCPIP_UDP_OptionsGet Function**C**

```
bool TCPIP_UDP_OptionsGet(  
    UDP_SOCKET hUDP,  
    UDP_SOCKET_OPTION option,  
    void* optParam  
);
```

Description

Various options can be get at the [socket](#) level. This function provides compatibility with BSD implementations.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	socket to get options for
option	specific option to get
optParam	pointer to an area that will receive the option value; this is option dependent the size of the area has to be large enough to accomodate the specific option UDP_OPTION_STRICT_PORT -- pointer to boolean UDP_OPTION_STRICT_NET -- pointer to boolean UDP_OPTION_BROADCAST -- UDP_SOCKET_BCAST_TYPE pointer to boolean UDP_OPTION_BUFFER_POOL -- pointer to boolean UDP_OPTION_TX_BUFF -- pointer to a 16 bit value to receive bytes of the TX buffer UDP_OPTION_TX_QUEUE_LIMIT -- pointer to an 8 bit value to receive the TX queue limit

Returns

true if success false otherwise

5.8.28.7.1.8 TCPIP_UDP_OptionsSet Function**C**

```
bool TCPIP_UDP_OptionsSet(  
    UDP_SOCKET hUDP,  
    UDP_SOCKET_OPTION option,  
    void* optParam  
);
```

Description

Various options can be set at the [socket](#) level. This function provides compatibility with BSD implementations.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	socket to set options for
option	specific option to be set
optParam	the option value; this is option dependent UDP_OPTION_STRICT_PORT -- boolean enable/disable UDP_OPTION_STRICT_NET -- boolean enable/disable UDP_OPTION_BROADCAST -- UDP_SOCKET_BCAST_TYPE UDP_OPTION_BUFFER_POOL -- boolean enable/disable

Returns

true if success false otherwise

Remarks

changing the UDP_OPTION_BUFFER_POOL will discard the data in the current [socket](#) buffer UDP_OPTION_TX_BUFF -- 16 bit value in bytes of the TX buffer

the UDP_OPTION_TX_BUFF will discard the data in the current [socket](#) buffer UDP_OPTION_TX_QUEUE_LIMIT -- 8 bit value of the TX queue limit

5.8.28.7.1.9 TCPIP_UDP_RemoteBind Function**C**

```
bool TCPIP_UDP_RemoteBind(
    UDP_SOCKET hUDP,
    IP_ADDRESS_TYPE addType,
    UDP_PORT remotePort,
    IP_MULTI_ADDRESS* remoteAddress
);
```

Description

Sockets don't need specific remote binding, they should [accept](#) connections on any incoming interface. Thus the binding is done automatically by the stack. However, specific binding can be requested using these functions. For a server [socket](#) it can be used to restrict accepting connections from a specific remote host. For a client [socket](#) it will just change the default binding done when the [socket](#) was opened. TBD: the call should fail if the [socket](#) is already bound to an interface (a server [socket](#) is connected or a client [socket](#) already sent the data on an interface). Implementation pending

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	The socket to bind .
addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
remotePort	The remote port to bind to.
remoteAddress	Remote address to use.

Returns

True if success False otherwise

Remarks

If remoteAddress == 0 the remote address of the [socket](#) won't be changed. The remote port is always changed, even if remotePort == 0.

5.8.28.7.1.10 TCPIP_UDP_ServerOpen Function**C**

```
UDP_SOCKET TCPIP_UDP_ServerOpen(  
    IP_ADDRESS_TYPE addType,  
    UDP_PORT localPort,  
    IP_MULTI_ADDRESS* localAddress  
);
```

Description

Provides a unified method for opening UDP server sockets.

Preconditions

UDP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
UDP_PORT localPort	UDP port on which to listen for connections
IP_MULTI_ADDRESS* localAddress	Local address to use. Can be NULL if any incoming interface will do.

Returns

[INVALID_SOCKET](#) - No sockets of the specified type were available to be opened.

Otherwise - A [UDP_SOCKET](#) handle. Save this handle and use it when calling all other UDP APIs.

5.8.28.7.1.11 TCPIP_UDP_SocketInfoGet Function**C**

```
bool TCPIP_UDP_SocketInfoGet(  
    UDP_SOCKET hUDP,  
    UDP_SOCKET_INFO* pInfo  
);
```

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
UDP_SOCKET hUDP	Socket to obtain info for.
UDP_SOCKET_INFO* pInfo	pointer to reference info location.

5.8.28.7.1.12 TCPIP_UDP_SocketNetGet Function**C**

```
TCPIP_NET_HANDLE TCPIP_UDP_SocketNetGet(  
    UDP_SOCKET hUDP  
);
```

Description

This function returns the interface handle of an UDP [socket](#)

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	The UDP socket

Returns

IHandle of the interface that [socket](#) currently uses.

5.8.28.7.1.13 TCPIP_UDP_SocketNetSet Function**C**

```
bool TCPIP_UDP_SocketNetSet(  
    UDP_SOCKET hUDP,  
    TCPIP_NET_HANDLE hNet  
);
```

Description

This function sets the network interface for an UDP [socket](#)

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	The UDP socket
hNet	interface handle

Returns

true if success false otherwise.

Remarks

An invalid hNet can be passed (0) so that the current [socket](#) network interface selection will be cleared

5.8.28.7.1.14 TCPIP_UDP_SourceIPAddressSet Function**C**

```
bool TCPIP_UDP_SourceIPAddressSet(  
    UDP_SOCKET s,  
    IP_ADDRESS_TYPE addType,  
    IP_MULTI_ADDRESS* localAddress  
);
```

Description

- It sets the IP source address This allows changing the source IPv4 address dynamically.

Preconditions

UDP initialized UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). s - valid [socket](#) localAddress - valid address pointer

Parameters

Parameters	Description
s	the UDP socket
addType	Type of address: IPv4/IPv6
localAddress	pointer to an address to use

Returns

true if success false otherwise

Remarks

The call will fail if localAddress is 0. The source IP address will not be changed.

5.8.28.7.1.15 TCPIP_UDP_TxOffsetSet Function**C**

```
bool TCPIP_UDP_TxOffsetSet(  
    UDP_SOCKET hUDP,  
    uint16_t wOffset,  
    bool relative  
);
```

Description

This function allows the write location within the TX buffer to be specified. Future calls to [TCPIP_UDP_Put](#), [TCPIP_UDP_ArrayPut](#), [TCPIP_UDP_StringPut](#), etc will write data from the indicated location.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle
wOffset	Offset in the UDP packet data payload to move the write pointer.
relative	if true, the wOffset is added to the current write pointer. else the wOffset is from the beginning of the UDP buffer

Returns

true if the offset is a valid one false otherwise

5.8.28.7.2 Transmit Data Transfer Functions**5.8.28.7.2.1 TCPIP_UDP_ArrayGet Function****C**

```
uint16_t TCPIP_UDP_ArrayGet(  
    UDP_SOCKET hUDP,  
    uint8_t * cData,  
    uint16_t wDataLen  
);
```

Description

This function reads an array of bytes from the UDP [socket](#), while decrementing the remaining bytes available. [TCPIP_UDP_GetIsReady](#) could be used before calling this function to get the number of the available bytes in the [socket](#).

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle
cData	The buffer to receive the bytes being read. If NULL, the bytes are simply discarded without being written anywhere (effectively skips over the bytes in the RX buffer, although if you need to skip a lot of data, seeking using the TCPIP_UDP_RxOffsetSet() will be more efficient).
wDataLen	Number of bytes to be read from the socket .

Returns

The number of bytes successfully read from the UDP buffer. If this value is less than wDataLen, then the buffer was emptied and no more data is available.

5.8.28.7.2.2 TCPIP_UDP_Flush Function**C**

```
uint16_t TCPIP_UDP_Flush(  
    UDP_SOCKET hUDP  
);
```

Description

This function builds a UDP packet with the pending TX data and marks it for transmission over the network interface. There is no UDP state machine to [send](#) the [socket](#) data automatically. The UDP [socket](#) client must call this function to actually [send](#) the data over the network.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle

Returns

The number of bytes that currently were in the [socket](#) TX buffer and have been flushed.

Remarks

Note that unlike [TCPIP_TCP_Flush](#), TCPIP_UDP_Flush must be called before returning to the main stack loop. There is no auto transmit for UDP segments.

5.8.28.7.2.3 TCPIP_UDP_Put Function**C**

```
uint16_t TCPIP_UDP_Put(  
    UDP_SOCKET hUDP,  
    uint8_t v  
);
```

Description

This function writes a single byte to the UDP [socket](#), while incrementing the buffer length. [TCPIP_UDP_TxPutIsReady](#) could be used before calling this function to verify that there is room in the [socket](#) buffer.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle
v	The byte to be loaded into the transmit buffer.

Return Values

Return Values	Description
1	The byte was successfully written to the socket .
0	The transmit buffer is already full and so the write failed.

Remarks

The following function is very inefficient. A buffered approach ([TCPIP_UDP_ArrayPut](#)) should be preferred

5.8.28.7.2.4 TCPIP_UDP_StringPut Function**C**

```
const uint8_t* TCPIP_UDP_StringPut(  
    UDP_SOCKET hUDP,  
    const uint8_t * strData  
);
```

Description

This function writes a null-terminated string to the UDP [socket](#) while incrementing the buffer length. [TCPIP_UDP_TxPutIsReady](#) could be used before calling this function to verify that there is room in the [socket](#) buffer.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle
cData	Pointer to the string to be written to the socket .

Returns

A pointer to the byte following the last byte written. Note that this is different than the [TCPIP_UDP_ArrayPut](#) functions. If this pointer does not dereference to a NULL byte, then the buffer became full and the input data was truncated.

5.8.28.7.2.5 TCPIP_UDP_TxCountGet Function**C**

```
uint16_t TCPIP_UDP_TxCountGet(  
    UDP_SOCKET hUDP  
);
```

Description

This function returns the amount of bytes written into the UDP [socket](#).

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle

5.8.28.7.2.6 TCPIP_UDP_TxPutIsReady Function**C**

```
uint16_t TCPIP_UDP_TxPutIsReady(  
    UDP_SOCKET hUDP,  
    unsigned short count  
);
```

Description

This function determines how many bytes can be written to the specified UDP [socket](#).

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle
count	Number of bytes requested

Returns

The number of bytes that can be written to this [socket](#).

Remarks

The function won't increase the size of the UDP TX buffer. If this is needed use [TCPIP_UDP_OptionsSet\(\)](#). The count variable is not used.

5.8.28.7.2.7 TCPIP_UDP_ArrayPut Function**C**

```
uint16_t TCPIP_UDP_ArrayPut(  
    UDP_SOCKET hUDP,  
    const uint8_t * cData,  
    uint16_t wDataLen  
);
```

Description

This function writes an array of bytes to the UDP [socket](#), while incrementing the buffer length. [TCPIP_UDP_TxPutIsReady](#) could be used before calling this function to verify that there is room in the [socket](#) buffer.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
cData	The array to write to the socket .
wDataLen	Number of bytes from cData to be written.

Returns

The number of bytes successfully placed in the UDP transmit buffer. If this value is less than wDataLen, then the buffer became

full and the input was truncated.

5.8.28.7.2.8 UDPIsPutReady Function

C

```
uint16_t UDPIsPutReady(  
    UDP_SOCKET hUDP  
);
```

Description

This function determines how many bytes can be written to the specified UDP [socket](#).

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle

Returns

The number of bytes that can be written to this [socket](#).

Remarks

The function is similar to the [TCPIP_UDP_TxPutIsReady](#) and maintained for backward compatibility.

5.8.28.7.3 Receive Data Transfer Functions

5.8.28.7.3.1 TCPIP_UDP_Discard Function

C

```
void TCPIP_UDP_Discard(  
    UDP_SOCKET hUDP  
);
```

Description

This function discards any remaining received data in the UDP [socket](#).

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	socket handle

Returns

None

Remarks

It is safe to call this function more than is necessary. If no data is available, this function does nothing.

5.8.28.7.3.2 TCPIP_UDP_Get Function

C

```
uint16_t TCPIP_UDP_Get(  
    UDP_SOCKET hUDP,  
    uint8_t * v  
);
```

Description

This function reads a single byte from the UDP [socket](#), while decrementing the remaining buffer length. [TCPIP_UDP_GetIsReady](#) could be used before calling this function to get the number of bytes available in the [socket](#).

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	socket handle
v	The buffer to receive the data being read.

Return Values

Return Values	Description
1	A byte was successfully read
0	No data remained in the read buffer or invalid socket

Remarks

The following function is very inefficient. A buffered approach ([TCPIP_UDP_ArrayGet](#)) should be preferred

5.8.28.7.3.3 TCPIP_UDP_GetIsReady Function

C

```
uint16_t TCPIP_UDP_GetIsReady(  
    UDP_SOCKET hUDP  
);
```

Description

This function determines if bytes can be read from the specified UDP [socket](#).

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle

Returns

The number of bytes that can be read from this [socket](#).

5.8.28.7.3.4 TCPIP_UDP_RxOffsetSet Function

C

```
void TCPIP_UDP_RxOffsetSet(  
    UDP_SOCKET hUDP,
```

```
    uint16_t rOffset  
);
```

Description

This function allows the read location within the RX buffer to be specified. Future calls to [TCPIP_UDP_Get](#) and [TCPIP_UDP_ArrayGet](#) will read data from the indicated location forward.

Preconditions

UDP [socket](#) should have been opened with [TCPIP_UDP_ServerOpen\(\)](#)/[TCPIP_UDP_ClientOpen\(\)](#). hUDP - valid [socket](#)

Parameters

Parameters	Description
hUDP	UDP socket handle
wOffset	Offset from beginning of UDP packet data payload to place the read pointer.

Returns

None

5.8.28.7.4 Data Types and Constants

5.8.28.7.4.1 INVALID_UDP_SOCKET Macro

```
C  
#define INVALID_UDP_SOCKET (-1)           // Indicates a UDP socket that is not valid
```

Description

Indicates a UDP [socket](#) that is not valid

5.8.28.7.4.2 TCPIP_UDP_IsOpened Macro

```
C  
#define TCPIP_UDP_IsOpened(s) TCPIP_UDP_IsConnected(s)
```

Description

backward compatibility call

5.8.28.7.4.3 UDP_MODULE_CONFIG Structure

```
C  
typedef struct {  
    uint16_t nSockets;  
    uint16_t sktTxBuffSize;  
    uint16_t poolBuffers;  
    uint16_t poolBufferSize;  
} UDP_MODULE_CONFIG;
```

Description

UDP layer configuration/initialization

Members

Members	Description
uint16_t nSockets;	number of sockets to be created

uint16_t sktTxBuffSize;	size of the socket tx buffer
uint16_t poolBuffers;	number of buffers in the pool; 0 if none
uint16_t poolBufferSize;	size of the buffers in the pool; all equal

5.8.28.7.4.4 UDP_PORT Type

C

```
typedef uint16_t UDP_PORT;
```

Description

Stores a UDP Port Number

5.8.28.7.4.5 UDP_SOCKET Type

C

```
typedef int16_t UDP_SOCKET;
```

Description

Provides a handle to a UDP Socket

5.8.28.7.4.6 UDP_SOCKET_BCAST_TYPE Enumeration

C

```
typedef enum {
    UDP_BCAST_NONE,
    UDP_BCAST_NETWORK_LIMITED,
    UDP_BCAST_NETWORK_DIRECTED
} UDP_SOCKET_BCAST_TYPE;
```

Description

This is type UDP_SOCKET_BCAST_TYPE.

Members

Members	Description
UDP_BCAST_NONE	no broadcast
UDP_BCAST_NETWORK_LIMITED	network limited broadcast
UDP_BCAST_NETWORK_DIRECTED	network directed broadcast

5.8.28.7.4.7 UDP_SOCKET_INFO Structure

C

```
typedef struct {
    IP_ADDRESS_TYPE addressType;
    IP_MULTI_ADDRESS remoteIPAddress;
    IP_MULTI_ADDRESS localIPAddress;
    UDP_PORT remotePort;
    UDP_PORT localPort;
} UDP_SOCKET_INFO;
```

Description

Information about a [socket](#)

Members

Members	Description
UDP_PORT remotePort;	Port number associated with remote node
UDP_PORT localPort;	local port number

5.8.28.7.4.8 UDP_SOCKET_OPTION Enumeration**C**

```
typedef enum {
    UDP_OPTION_STRICT_PORT,
    UDP_OPTION_STRICT_NET,
    UDP_OPTION_BROADCAST,
    UDP_OPTION_BUFFER_POOL,
    UDP_OPTION_TX_BUFF,
    UDP_OPTION_TX_QUEUE_LIMIT
} UDP_SOCKET_OPTION;
```

Description

UDP [socket](#) options

Members

Members	Description
UDP_OPTION_STRICT_PORT	When connection is done the socket stores the remote host local port. If option is enabled the remote host local port is always checked to match the initial one. If disabled the remote host local port is not checked. Disabled by default on a socket server. Enabled by default on a client socket .
UDP_OPTION_STRICT_NET	When connection is done the socket stores the network interface the connection occurred on. If option is enabled the socket accepts data only from the interface that matches the initial connection. If disabled the socket receives data from any remote host irrespective of the interface on which the packet arrived. Disabled by default on a socket server. Enabled by default on a client socket .
UDP_OPTION_BROADCAST	Enables the Broadcast transmission by the socket
UDP_OPTION_BUFFER_POOL	Enables the socket to use the private UDP buffers pool. The size of the TX buffer has to be less than the size of the buffers in the pool
UDP_OPTION_TX_BUFF	request different TX buffer size. Has to call TCPIP_UDP_OptionsGet to see the exact space allocated
UDP_OPTION_TX_QUEUE_LIMIT	Sets the maximum number of packets that can be queued/allocated by an UDP socket at a certain time. For sockets that need to transmit a lot of data (iperf client, for example), especially on slow connections this limit prevents running out of memory because the MAC transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the size of the UDP TX buffer, the connection speed and the amount of memory available to the stack.

5.8.28.8 Files**Files**

Name	Description
udp.h	
udp_config.h	UDP onfiguration file

Description

5.8.28.8.1 udp.h

UDP Module Defs for Microchip TCP/IP Stack

Enumerations

	Name	Description
	UDP_SOCKET_BCAST_TYPE	This is type UDP_SOCKET_BCAST_TYPE.
	UDP_SOCKET_OPTION	UDP socket options

Functions

	Name	Description
	TCPIP_UDP_ArrayGet	Reads an array of bytes from the UDP socket .
	TCPIP_UDP_ArrayPut	Writes an array of bytes to the UDP socket .
	TCPIP_UDP_BcastIPv4AddressSet	Sets the broadcast IP address of a socket . Allows an UDP socket to send broadcasts.
	TCPIP_UDP_Bind	Bind a socket to a local address and port. This function is meant for client sockets. It assigns a specific source address and port for a socket .
	TCPIP_UDP_ClientOpen	Opens a UDP socket as a client.
	TCPIP_UDP_Close	Closes a UDP socket and frees the handle.
	TCPIP_UDP_DestinationIPAddressSet	Sets the destination IP address of a socket .
	TCPIP_UDP_Discard	Discards any remaining RX data from a UDP socket .
	TCPIP_UDP_Flush	Transmits all pending data in a UDP socket .
	TCPIP_UDP_Get	Reads a byte from the UDP socket .
	TCPIP_UDP_GetIsReady	Determines how many bytes can be read from the UDP socket .
	TCPIP_UDP_IsConnected	Determines if a socket has an established connection.
	TCPIP_UDP_OptionsGet	Allows getting the options for a socket like: current Rx/Tx buffer size, etc
	TCPIP_UDP_OptionsSet	Allows setting options to a socket like adjust Rx/Tx buffer size, etc
	TCPIP_UDP_Put	Writes a byte to the UDP socket .
	TCPIP_UDP_RemoteBind	Bind a socket to a remote address. This function is meant for server sockets.
	TCPIP_UDP_RxOffsetSet	Moves the pointer within the RX buffer.
	TCPIP_UDP_ServerOpen	Opens a UDP socket as a server.
	TCPIP_UDP_SocketInfoGet	Points handle at socket info for socket hUDP
	TCPIP_UDP_SocketNetGet	Gets the network interface of an UDP socket .
	TCPIP_UDP_SocketNetSet	Sets the network interface for an UDP socket .
	TCPIP_UDP_SourceIPAddressSet	Sets the source IP address of a socket .
	TCPIP_UDP_StringPut	Writes null-terminated string to the UDP socket .
	TCPIP_UDP_TxCountGet	Returns the amount of bytes written into the UDP socket .
	TCPIP_UDP_TxOffsetSet	Moves the pointer within the TX buffer.
	TCPIP_UDP_TxPutIsReady	Determines how many bytes can be written to the UDP socket .
	UDPIsPutReady	Determines how many bytes can be written to the UDP socket .

Macros

	Name	Description
	INVALID_UDP_SOCKET	Indicates a UDP socket that is not valid
	TCPIP_UDP_IsOpened	backward compatibility call

Structures

	Name	Description
	UDP_MODULE_CONFIG	UDP layer configuration/initialization
	UDP_SOCKET_INFO	Information about a socket

Types

	Name	Description
	UDP_PORT	Stores a UDP Port Number
	UDP_SOCKET	Provides a handle to a UDP Socket

5.8.28.8.2 udp_config.h

User Datagram Protocol (UDP) Configuration file

This file contains the UDP module configuration options

Macros

	Name	Description
	UDP_LOCAL_PORT_END_NUMBER	Last port number for randomized local port number selection
	UDP_LOCAL_PORT_START_NUMBER	The dynamic port range defined by IANA consists of the 49152-65535 range and is meant for the selection of ephemeral ports (RFC 6056). Adjust to your needs but stay within the IANA range First port number for randomized local port number selection
	UDP_MAX_SOCKETS	global number of UDP sockets created dynamically
	UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT	the maximum number of packets that can be queued by an UDP socket at a certain time. For sockets that need to transfer a lot of data (iperf, for example), especially on slow connections this limit prevents running out of memory because the MAC transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the UDP_SOCKET_DEFAULT_TX_SIZE , the connection speed and the amount of memory available to the stack.
	UDP_SOCKET_DEFAULT_TX_SIZE	default socket Tx buffer size
	UDP_SOCKET_POOL_BUFFER_SIZE	size of the buffers in the UDP pool any UDP socket that is enabled to use the pool and has the TX size <= than this size can use a buffer from the pool
	UDP_SOCKET_POOL_BUFFERS	use 0 to disable the feature
	UDP_USE_RX_CHECKSUM	check incoming packets to have proper checksum
	UDP_USE_TX_CHECKSUM	This slows UDP TX performance by nearly 50%, except when using the ENCX24J600, which has a super fast DMA and incurs virtually no speed penalty.

5.8.29 Zeroconf TCP/IP Stack Library

5.8.29.1 Introduction

Zero Configuration (Zeroconf) TCP/IP Stack Library for Microchip Microcontrollers

This library provides the API of the Zero Conf TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard.

5.8.29.2 Release Notes

MPLAB Harmony Version: v0.70b **Zero Conf TCP/IP Stack Library Version :** 1.0 **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.8.29.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.8.29.4 Using the Library

This topic describes the basic architecture of the Zeroconf TCP/IP Stack Library and provides information and examples on how to use it.

Interface Header Files: `zero_conf_helper.h`, `zero_conf_link_local.h`, and `zero_conf_multicast_dns.h`

The interface to the Zeroconf TCP/IP Stack library is defined in the "`zero_conf_helper.h`", "`zero_conf_link_local.h`", and "`zero_conf_multicast_dns.h`" header files. This file is included by the "`tcpip.h`" file. Any C language source (.c) file that uses the Zeroconf TCP/IP Stack library should include "`tcpip.h`".

Library File:

The Zeroconf TCP/IP Stack library archive (.a) file installed with MPLAB Harmony.

Please refer to the [MPLAB Harmony Overview](#) for how the TCP/IP Stack interacts with the framework.

5.8.29.4.1 Abstraction Model

This library provides the API of the Zeroconf TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Zero Conf Software Abstraction Block

Note: This image was not available at time of release and will be added in a future release of MPLAB Harmony.

5.8.29.4.2 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Zeroconf module.

Library Interface Section	Description
Multicast DNS Functions	This section provides multicast DNS interface routines to Zeroconf
Link Local Functions	This section provides Link Local interface routines to Zeroconf
Data Types and Constants	This section provides various definitions describing this API

5.8.29.4.3 Core Functionality

5.8.29.4.3.1 Enabling

Zeroconf can be enabled by setting the following two defines in `TCPIPConfig.h`:

- `STACK_USE_ZEROCONF_LINK_LOCAL`
- `STACK_USE_ZEROCONF_MDNS_SD`

Currently, the use of Zeroconf is limited to the W-iFi demo applications (and the MRF24WB0M module). Future versions of the stack should enable Zeroconf support across all Ethernet solutions.

5.8.29.4.3.2 Link Local (Zeroconf)

The first component of Zeroconf is the ability to self-assign an IP address to each member of a network. Normally, a DHCP server would handle such situations. However, in cases where no DHCP server exists, Zeroconf enabled devices negotiate unique IP addresses amongst themselves.

5.8.29.4.3.3 mDNS

The second component of Zeroconf is the ability to self-assign human-readable hostnames for themselves. Multicast DNS provides a local network the ability to have the features of a DNS server. Users can use easily remembered hostnames to access the devices on the network. In the event that devices elect to use the same hostname, as in the IP address resolution, each of the devices will auto-negotiate new names for themselves (usually by appending a number to the end of the name).

5.8.29.4.3.4 Service Discovery

The last component of Zeroconf is service discovery. All Zeroconf devices can broadcast what services they provide. For instance, a printer can broadcast that it has printing services available. A thermostat can broadcast that it has an HVAC control service. Other interested parties on the network who are looking for certain services can then see a list of devices that have the capability of providing the service, and [connect](#) directly to it. This further eliminates the need to know whether something exists on a network (and what its IP or hostname is). As an end-user, all you would need to do is query the network if a certain service exists, and easily [connect](#) to it.

5.8.29.4.3.5 Demonstration

The demo, when enabled, shows all three items above working together. Each development kit in the network assumes the hostname of MCHPBOARD-x.local, where x is an incrementing number from 1 (only in the case where multiple kits are programmed for the network). Each board will broadcast its service, which is the DemoWebServer.

5.8.29.4.3.6 Zeroconf Enabled Environments

All Apple products have Zeroconf enabled by default. On Windows, you'll need to download the Safari web browser, and during the install, enable support for Bonjour. Note that in the Safari browser, you can browse and see a list of all Bonjour enabled devices, and click through to them automatically.

5.8.29.5 Configuring the Library

The configuration of the Zeroconf TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Zeroconf TCP/IP Stack Library. Based on the selections made, the Zeroconf TCP/IP Stack Library will support or not support selected features. These configuration settings will apply to all instances of the Zeroconf TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

5.8.29.6 Building the Library

This section lists the files that are available in the `\src` of the Zeroconf driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

5.8.29.7 Library Interface

Data Types and Constants

	Name	Description
	MDNSD_ERR_CODE	void DisplayHostName(uint8_t *HostName);
	ZCLL_MODULE_CONFIG	Placeholder for Zero Configuration Link Layer module configuration.

Link Local Functions

	Name	Description
	TCPIP_ZCLL_Disable	disables the ZCLL on an interface
	TCPIP_ZCLL_Enable	enables ZCLL on an interface
	TCPIP_ZCLL_IsEnabled	returns true if ZCLL is currently enabled on that interface

Multicast DNS Functions

	Name	Description
	TCPIP_MDNS_ServiceDeregister	DNS-Service Discovery API for end-user to De-register a service-advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister API.
	TCPIP_MDNS_ServiceRegister	DNS-Service Discovery API for end-users to register a service-advertisement. The service is associated with all interfaces.
	TCPIP_MDNS_ServiceUpdate	DNS-Service Discovery API for end-user to update the service -advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister

Description

This section describes the Application Programming Interface (API) functions of the Zeroconf TCP/IP Stack.

Refer to each section for a detailed description.

5.8.29.7.1 Multicast DNS Functions

5.8.29.7.1.1 TCPIP_MDNS_ServiceDeregister Function

C

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceDeregister(
    TCPIP_NET_HANDLE netH
);
```

Description

This API is used by end-user application to de-register DNS-Service Discovery on a local network. When this gets invoked by end-user DNS-SD stack sends out Good-Bye packets to update all peer machines that service will no longer be present. All peer machines remove the corresponding entry from Browser list.

This is the last API that needs to be invoked by end-user application to free-up the DNS-SD stack for some other app.

Preconditions

[TCPIP_MDNS_ServiceRegister](#) must be invoked before this call.

Parameters

Parameters	Description
netH	handle of the network to be de-registered

Returns

MDNSD_ERR_CODE - Returns Error-code to indicate registration is success or not. 1) MDNSD_SUCCESS - returns on success of call 2) MDNSD_ERR_INVALID - When the input parameters are invalid or if the API is invoked in invalid state

5.8.29.7.1.2 TCPIP_MDNS_ServiceRegister Function**C**

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceRegister(
    TCPIP_NET_HANDLE netH,
    const char * srv_name,
    const char * srv_type,
    uint16_t port,
    const uint8_t * txt_record,
    uint8_t auto_rename,
    void (*call_back)(char *name, MDNSD_ERR_CODE err, void *context),
    void * context
);
```

Description

DNS-Service Discovery APIs

This API is used by end-user application to announce its service on local network. All peer machines that are compliant with Multicast-DNS & DNS-Service Discovery protocol can detect the announcement and lists out an entry in Service-Browser list. End-User selects an entry to [connect](#) to this service. So ultimately this is an aid to end-user to discover any service of interest on a local network.

This is the first API that needs to be invoked by end-user application. Presently Multicast-DNS & Service-discovery stack supports only single service-advertisement. Once the application wants to terminate the service it has to invoke [TCPIP_MDNS_ServiceDeregister\(\)](#) API to free-up the DNS-SD stack for some other application.

Preconditions

None

Parameters

Parameters	Description
netH	handle of the network to be registered
srv_name	Service Name, which is being advertised
srv_type	For a HTTP-Service its "_http._tcp.local"
_http	is application protocol preceeded with under-score
_tcp	is lower-layer protocol on which service runs
local	is to represent service is on local-network For a iTunes Music Sharing "_daap._tcp.local" For a Printing Service "_ipp._tcp.local" Refer to http://www.dns-sd.org/ServiceTypes.html for more service types
port	Port number on which service is running
txt_len	For additional information about service like default page (eg "index.htm") for HTTP-service. Length of such additional information
txt_record	String of additional information (eg "index.htm") for HTTP-service.
auto_rename	A flag to indicate DNS-SD stack, whether to rename the service automatically or not. If this is set to '0' Callback parameter will be used to indicate the conflict error and user has to select different name and re-register with this API. If this is set to '1' service-name will be automatically renamed with numerical suffix.

callback	Callback function, which is user-application defined.
This callback gets invoked on completion of service	advertisement. If an service name-conflict error is detected and auto_rename is set to '0' callback gets invoked with MDNSD_ERR_CONFLICT as error-code.
context	Opaque context (pointer to opaque data), which needs to be used in callback function.

Returns

MDNSD_ERR_CODE - Returns Error-code to indicate registration is success or not. 1) MDNSD_SUCCESS - returns on success of call 2) MDNSD_ERR_BUSY - When already some other service is being advertised using this DNS-SD stack 3) MDNSD_ERR_INVALID - Invalid Parameter

5.8.29.7.1.3 TCPIP_MDNS_ServiceUpdate Function**C**

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceUpdate(
    TCPIP_NET_HANDLE netH,
    uint16_t port,
    const uint8_t * txt_record
);
```

Description

This API is used by end-user application to update its service which was previously registered. With this API end-user app update the port number on which the service is running. It can update the additional information of service. For example: the default page can be updated to new page and corresponding page name can be input to this API to update all peer machines. The modified service will be announced with new contents on local network.

This is an optional API and should be invoked only if it is necessary.

Preconditions

TCPIP_MDNS_ServiceRegister must be invoked before this call.

Parameters

Parameters	Description
netH	handle of the network to perform the service update
port	Port number on which service is running
txt_record	String of additional information (eg "index.htm") for HTTP-service.

Returns

MDNSD_ERR_CODE - Returns Error-code to indicate registration is success or not. 1) MDNSD_SUCCESS - returns on success of call 2) MDNSD_ERR_INVALID - When the input parameters are invalid or if the API is invoked in invalid state

5.8.29.7.2 Link Local Functions**5.8.29.7.2.1 TCPIP_ZCLL_Disable Function****C**

```
bool TCPIP_ZCLL_Disable(
    TCPIP_NET_HANDLE hNet
);
```

Description

disables the ZCLL on an interface

5.8.29.7.2.2 TCPIP_ZCLL_Enable Function

C

```
bool TCPIP_ZCLL_Enable(  
    TCPIP_NET_HANDLE hNet  
);
```

Description

enables ZCLL on an interface

5.8.29.7.2.3 TCPIP_ZCLL_IsEnabled Function

C

```
bool TCPIP_ZCLL_IsEnabled(  
    TCPIP_NET_HANDLE hNet  
);
```

Description

returns true if ZCLL is currently enabled on that interface

5.8.29.7.3 Data Types and Constants

5.8.29.7.3.1 MDNSD_ERR_CODE Enumeration

C

```
typedef enum {  
    MDNSD_SUCCESS = 0,  
    MDNSD_ERR_BUSY = 1,  
    MDNSD_ERR_CONFLICT = 2,  
    MDNSD_ERR_INVAL = 3  
} MDNSD_ERR_CODE;
```

Description

void DisplayHostName(uint8_t *HostName);

Members

Members	Description
MDNSD_ERR_BUSY = 1	Already Being used for another Service
MDNSD_ERR_CONFLICT = 2	Name Conflict
MDNSD_ERR_INVAL = 3	Invalid Parameter

5.8.29.7.3.2 ZCLL_MODULE_CONFIG Structure

C

```
typedef struct {  
} ZCLL_MODULE_CONFIG;
```

Description

Placeholder for Zero Configuration Link Layer module configuration.

5.8.29.8 Files

Files

Name	Description
zero_conf_link_local.h	
zero_conf_multicast_dns.h	This is file zero_conf_multicast_dns.h.

Description

5.8.29.8.1 zero_conf_link_local.h

Zero Confiruation (Zeroconf) IPV4 Link Local Addressing Module for Microchip TCP/IP Stack

Functions

	Name	Description
	TCPIP_ZCLL_Disable	disables the ZCLL on an interface
	TCPIP_ZCLL_Enable	enables ZCLL on an interface
	TCPIP_ZCLL_IsEnabled	returns true if ZCLL is currently enabled on that interface

Structures

	Name	Description
	ZCLL_MODULE_CONFIG	Placeholder for Zero Configuration Link Layer module configuration.

5.8.29.8.2 zero_conf_multicast_dns.h

This is file zero_conf_multicast_dns.h.

Enumerations

	Name	Description
	MDNSD_ERR_CODE	void DisplayHostName(uint8_t *HostName);

Functions

	Name	Description
	TCPIP_MDNS_ServiceDeregister	DNS-Service Discovery API for end-user to De-register a service-advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister API.
	TCPIP_MDNS_ServiceRegister	DNS-Service Discovery API for end-users to register a service-advertisement. The service is associated with all interfaces.
	TCPIP_MDNS_ServiceUpdate	DNS-Service Discovery API for end-user to update the service -advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister

5.9 USB Library Help

5.9.1 USB Device Library - Getting Started

5.9.1.1 Introduction

Provides an introduction to the MPLAB Harmony USB Device Library

Description

The MPLAB Harmony USB Device Library (referred to as the USB Device Library) provides embedded application developers with a framework to design and develop a wide variety of USB Devices. A choice of Full Speed only or Full Speed and Hi-Speed USB operations are available, depending on the selected PIC32 microcontroller. The USB Device Library facilitates development of standard USB devices through function drivers that implement standard USB Device class specification. Custom USB devices can also be implemented via a Generic Function driver. The USB Device Library is modular, thus allowing application developers to readily design composite USB devices. The USB Device Library is a part of the MPLAB Harmony installation and is accompanied by demonstration applications that highlight library usage. These demonstrations can also be modified or updated to build custom applications. The USB Device Library also features the following:

- Support for different USB device classes (CDC, Audio, HID, MSC and Generic)
- Supports multiple instance of the same class in a composite device
- Supports multiple configurations at different speeds
- Supports full speed and high speed operation
- Supports multiple USB peripherals (allows multiple device stacks)
- Modular and Layered architecture
- Supports deferred control transfer responses.
- Completely non-blocking
- Supports both polled and interrupt operation.
- Works readily in an RTOS environment.

This document serves as getting started guide and provides information on the following:

- USB Device Stack Architecture
- USB Device Library - Application Interaction
- Creating your own USB device

Note: This document assumes that the reader is familiar with the USB 2.0 specification (available at www.usbif.org). While the document, for the sake completeness, does cover certain aspects of the USB 2.0 protocol, it is recommended that the reader

refer to the specification for a complete description of USB operation.

5.9.1.2 Release Notes

MPLAB Harmony Version: v0.70b **Release Date:** 18Nov2013

New This Release:

Nothing to report in this release.

Known Issues:

- USB Device Stack:
 - Limited USB chapter 9 testing performed on HID and MSD function drivers.
 - The stack has not been tested for USB Interoperability.
 - The USB Device Stack has not been tested with a Real Time Operating System.
 - Attach/Detach behavior has been tested in a limited capacity.
 - The device stack only supports interrupt mode operation. Polled mode operation will be added in a future release.
 - While running the device stack on PIC32MZ microcontroller, the stack requires 3 seconds to initialize. This is due to a hardware issue with PIC32MZ USB module.
 - Device stack operation has been tested with Windows XP and Windows 7 OS based PC USB Host.
 - The USB_DEVICE_ResumeStart() and USB_DEVICE_ResumeStop() functions are not implemented.
- USB Host Stack:
 - The host Stack has not been tested with a Real Time Operating System.
 - Attach/Detach behavior has been tested in a limited capacity.
 - The host stack only supports interrupt mode operation. Polled mode operation will be added in a future release.
 - While running the host stack on PIC32MZ microcontroller, the stack requires 3 seconds to initialize. This is due to a hardware issue with PIC32MZ USB module. The 3 second delay is implemented using a while loop. This will be replaced with System Timer services in the next release.
 - MSD Host has been tested with a limited number of commercially available USB Flash Drives.
 - MSD Host and the USB Host Layer have not been tested for read/write throughput. This will be done in a future release.
 - Host stack has not been tested with low speed devices.
 - The PIC32MZ host layer performs direct access of Timer 2 peripheral SFR, for its timing requirements. This will be replaced with Harmony System Timer services in the next release of the host stack.
 - USB_HOST_DeviceSuspend() and the USB_HOST_DeviceResume() functions are not implemented.

5.9.1.2.1 Previous Versions

5.9.1.2.1.1 v0.51b

Release notes for version 0.51b of the MPLAB Harmony USB Stack.

Description

New features in this release:

- Added Device Stack support for Mass Storage device class.
- Added Device Stack support for Audio Device class.
- Added demonstration applications for Audio and Mass Storage Device classes.
- Added support for PIC32MZ devices and High Speed operation.
- HID Basic Demonstration application tested for USB 2.0 chapter-9 conformance.
- MSD Basic Demonstration application tested for USB2.0 chapter-9 conformance.
- First Release of Host Layer and CDC Host Driver along with a demonstration application.

Limitations and Known Issues:

The following are the known issues in this release. These issues will be fixed in the subsequent releases of the MPLAB Harmony USB Stack.

- The USB device stack is tested with Windows environment only. Performance and working of the device stack is not tested with other OS environments.
- The stack is not thoroughly tested for inter operability with other USB devices when connected to hub along with other devices.
- All demos are tested in dynamic interrupt mode only.
- Limited testing is done with hub.
- Limited testing is done with RTOS.
- APIs may change in subsequent releases and are not final.
- PIC32MZ USB Driver supports only on instance of the USB peripheral.
- The Mass Storage Device Class demonstration application works with Windows XP OS only.
- The Audio Speaker demo may not restart audio playback if the playback is paused from the PC.
- The CDC Host layer has been in tested in a limited capacity.
- PIC32MZ and Hi-Speed USB support is tested with CDC and HID device classes only.
- HID Keyboard demonstration application does not receive output report from PC USB host.

5.9.1.2.1.2 v0.51.02b

Release notes for the current version of the MPLAB Harmony USB Stack

Description**New features in this release:**

- Updated documentation to include a Getting Started with USB Device Library.
- Updated the documentation for CDC and HID Function Drivers. Improved code size and RAM requirement of the demos.
- The following issues from the v0.51b are fixed in this release
 - HID Keyboard demonstration application does not receive output report from PC USB host.
 - The Audio Speaker demo may not restart audio playback if the playback is paused from the PC.
 - The Mass Storage Device Class demonstration application works with Windows XP OS only. The demo now works with Windows 7.

Limitations and Known Issues:

The following are the known issues in this release. These issues will be fixed in the subsequent releases of the MPLAB Harmony USB Stack.

- The USB device stack is tested with Windows environment only. Performance and working of the device stack is not tested with other OS environments.

- The stack is not thoroughly tested for inter operability with other USB devices when connected to hub along with other devices.
- All demos are tested in dynamic interrupt mode only.
- Limited testing is done with hub. Limited testing is done with RTOS.
- Audio Speaker demo may not work with all USB Hosts. This has been observed with faster PC Hosts.
- APIs are not final and may change in subsequent releases.
- PIC32MZ USB Driver supports only one instance of the USB peripheral.
- The CDC Host layer has been tested in a limited capacity.
- PIC32MZ and Hi-Speed USB support is tested with CDC and HID device classes only.
- Polling operation is not implemented
- Static USB Controller Driver is not available.

5.9.1.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.9.1.4 USB Device Library Architecture

Describes the USB Device Library Architecture.

Description

The USB Device Library features a modular and layered architecture. This is shown in figure 1.

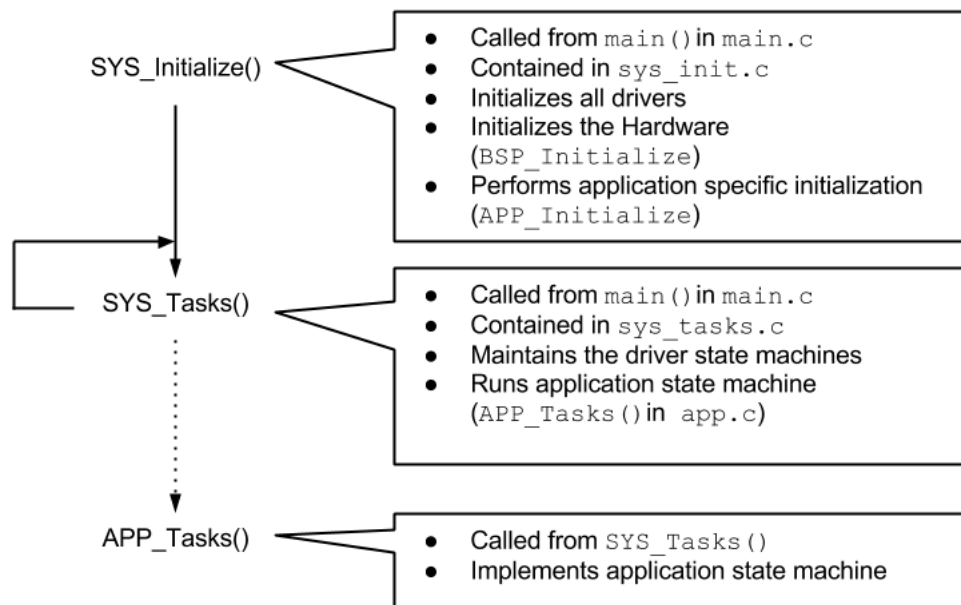


Figure 1: USB Device Library Architecture

As seen in figure 1, the USB Device Library consists of the 3 major components

1. The **USB Controller Driver (USBCD)** manages the state of the USB peripheral and provides the Device Layer with structured data access methods to the USB. It also provides the device layer with USB events. The USBCD is a MPLAB Harmony driver and uses the Harmony framework components (the USB peripheral Library, the Interrupt System Service) for its operation. It supports only one client per instance of the USB Peripheral. This client would typically be the Device Layer. In case of multiple USB peripherals, the USBCD can manage multiple USB peripherals, each being accessed by one client. The driver is accessed exclusively by the Device Layer in the USB Device Layer Architecture. It is initialized by the Device Layer (when the Device Layer is initialized) and in case of polling operation, it's Tasks routine is called by the Device Layer. The USBCD provides functions to

- enable, disable and stall endpoints
- schedule USB transfers.
- attach or detach the device
- control resume signalling

2. The **Device Layer** responds to the enumeration requests issued by the USB host. It has exclusive access to an instance of the USBCD and Endpoint 0 which is the control endpoint. When the host issues a class specific control transfer request, the Device Layer will analyze the setup packet of the control transfer and will route the control transfer to the appropriate function driver. The Device Layer must be initialized with the following data:

- Master Descriptor Table: This is a table of all the configuration descriptors and string descriptors.
- Function Driver Registration Table : This table contains information about the function drivers in the application
- USBCD initialization information: This specifies the USB peripheral interrupt, the USB Peripheral instance and sleep mode operation options.

The Device Layer initializes all function drivers that are registered with it when it receives a Set Configuration (for a supported configuration) from the host. It de-initializes the function drivers when a USB reset event occurs. It initializes the USBCD, opens it and registers a event handler to receive USB events. The Device Layer can also be opened by the application (the application

becomes a client to the Device Layer). The application can then receive bus and device events and respond to control transfer requests. The Device Layer provides events to the application such as device configured or device reset. The application must then respond appropriately to these events.

3. The **Function Drivers** implements various USB device classes as per the class specification. The USB Device Library architecture can support multiple instances of a function driver. An example would be a USB CDC device that emulates 2 serial ports. Function drivers provide an abstracted and an easy to use interface to the application. The application must register an event handler with the function driver to receive function driver events and must respond to some of these events with control transfer read/write functions. Function driver access the bus through the Device Layer.

5.9.1.5 USB Device Library - Application Interaction

Describes how the application must interact with the USB Device Stack

Description

Figure 2 highlights the steps that the application must follow in order to use the USB Device Library.

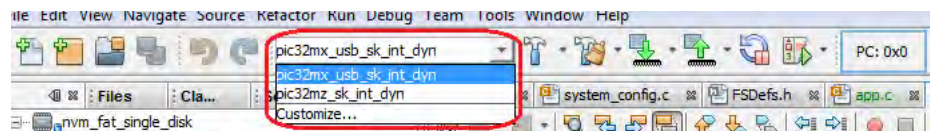


Figure 2: Application Interaction with Device Layer

The application must first initialize the Device Layer. As a part of the Device Layer initialization process, the Device Layer initialization structure must be defined which in turn requires the following data structures to be designed

- The master descriptor table
- The function driver registration table

Figure 3 shows a pictorial representation of the data that forms the Device Layer initialization structure. Additional information on Device Layer initialization is available in the Device Layer Help File.

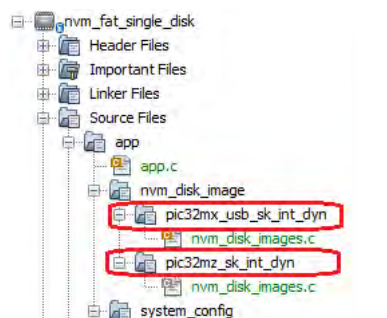


Figure 3. Device Layer Initialization

After successful initialization of the device layer, the application can open the device layer and register a device layer event handler. The device layer event handler receives device level events such as device configured, device de-configured, device reset and device suspended. The device configured event and de-configured event are important. The application can use the device deconfigured event to re-initialize its internal state machine. When the application receives a device configured event, it must register event handlers for each function driver that is relevant to the configuration that was set. The function driver event handler registration must be done in the device configured event context because the device layer acknowledges the set configuration request from the host when it exits the device configured event handler context. The application at this point should be ready to respond to function driver events which require application intervention.

Note: Not registering the function driver event handler in the device layer device configured event could cause the device to not

respond to the host requests and hence be non-compliant.

Once configured, the device is now ready to serve its intended function on the USB. The application interacts with the device layer and function drivers through API function and event handlers. The application must be aware of function driver events which require application response. For example, the `USB_DEVICE_CDC_EVENT_SET_LINE_CODING` event from the CDC Function Driver requires the application to respond with a `USB_DEVICE_CDC_ControlRead()` function. This function provides the buffer to **accept** the line coding parameters that the host will provide in the data stage of the Set Line Coding control transfer. Figure 4 shows the application interaction with device layer and function driver after the device has been configured.

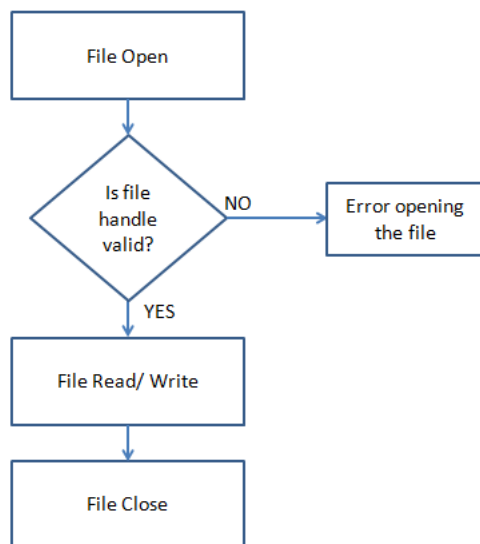


Figure 4: Application - Device Layer Interaction after device configuration

In figure 4, the application should have registered the device layer event handler before attaching the device on the bus. It should have registered the function driver event handler before exiting the device configured - device layer event. The application will then receive function driver instance specific events via the function driver event handlers.

Deferring Control Transfer Responses:

Some function driver events require the application to respond to the data stage and/or the status stage of the function driver (class) specific control transfer associated with the event. The application responds to the data stage by specifying a buffer that either contains the data that needs to be sent to host (control read) or will contain the data that host sends to the device (control write). In case of a control write, the application may wish to inspect the data which was received from the host and then acknowledge or stall the status stage of the control transfer. In such cases, it is possible that the application may not be ready within the function driver event handler context, with data that needs to be sent in the control read transfer or the success/ failure decision in response to the control write transfer. Performing extended processing or waiting for external hardware within the function driver event handler context is not recommended as USB 2.0 specification places restrictions on the control transfer response time.

In cases where the application is not ready to respond to control transfer requests within the function driver event handler context, the USB Device Library provides the option of deferring the response to the event. The application can respond to the control transfer request out of function driver event handler context. The function driver generates a control transfer handle, which the application must use while responding to the function driver event outside the event handler context. The application must still observe the USB 2.0 specification control transfer timing requirements while responding to the control transfer. Deferring the response in such a manner provides the application with flexibility to analyze the control transfer without affecting the dynamic response of the device.

5.9.1.6 Creating Your Own USB Device

Describes how to create a USB device with MPLAB Harmony USB Device Library

Description

The first step in creating a USB device is identifying if the desired device function fits into any of the standard USB device class functions. Using standard USB classes may be advantageous as major operating systems feature ready Host driver support for standard USB devices. The application may however not wish to tolerate the overhead associated with standard USB device class protocols, in which case, a custom USB device can be implemented. A custom USB device can be implemented by using the Generic Device function driver in the USB Device Library, but these devices will require development of host drivers for their operation. Having identified the device class to be used, the following approaches are available for developing a USB device by using the USB Device Library.

1. Use the available library demonstration applications. The USB Device Library release package contains a set of demo applications which are representative of common USB devices. These can be modified easily to include application specific initialization and application logic. The application logic must be non-blocking and could be implemented as a state machine.

- The application specific initialization can be called in the `SYS_Initialize()` function (in the `sys_init.c` file). The `SYS_Initialize` is called when the device comes out of power on reset.

- The application logic can be called in the `SYS_Tasks()` function (in the `sys_tasks.c` file). The application logic can interact with the function driver and the device layer by using relevant API calls.

- The application logic can track device events by processing the events in the application USB device event handler function (`APP_USBDeviceEventHandler()` function in `app.c`).

2. Build a USB Device Library application from scratch. In a case where the available demo applications do not meet the end application requirements, a USB device can be created by building a USB Device Library application from scratch. Figure 5 shows a flowchart for the steps that need to be followed to do this.

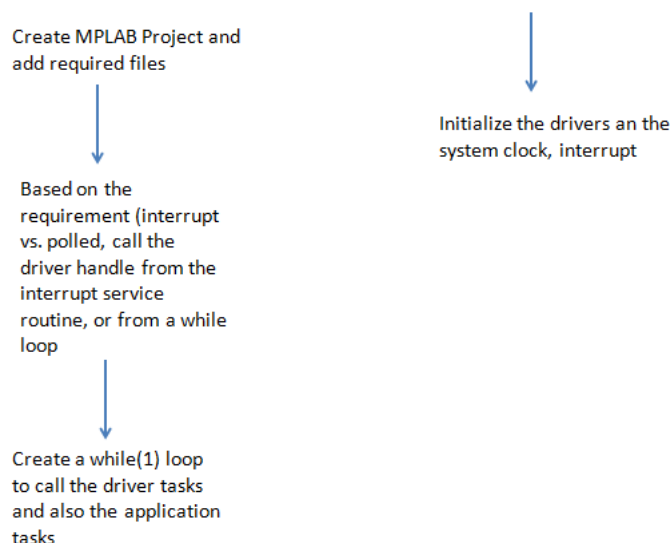


Figure 5: Steps to create a USB Device Library Application

Step 1:

Create a MPLAB Project and add the required USB Device Library file to the project. The following files are needed to build the a USB Device Library project

- system_config.h - This file should contain the compile time configuration macros for the Function Driver, Device Layer and Controller Driver. Refer to the "Configuring the Library" section in the documentation
- usb_device.c - This is the device layer implementation
- drv_usb.c and drv_usb_device.c - These files implement the USB device mode Controller Driver.
- sys_int_pic32.c and plib_int_pic32.c - These files implement the system interrupt service that is required by the USB CD.
- function driver implementation files - This depends on the USB device function being implemented. The "Using the Library" section in the function driver specific help file provides more details on the files to be included while using the function driver.
- Application specific files - These file will implement the application logic

Step 2:

Setup the fuse configuration bits to enable PLL. The code snippets below can be used to do this for PIC32MX and PIC32MZ processors

```

/*****
 * The following code snippet shows the USB PLL related
 * fuse configuration bits for PIC32MX processors. A 8MHz
 * external crystal/oscillator is assumed.
 *****/
#pragma config UPLLEN    = ON        /* Enable USB PLL */
#pragma config UPLLIDIV = DIV_2     /* USB PLL input divider */
/*****
 * The following code snippet shows the USB PLL related
 * fuse configuration bits for PIC32MZ processors. A 24MHz
 * external crystal/oscillator is assumed.
 *****/
#pragma config UPLLFSEL = FREQ_24MHZ // USB PLL Input Frequency Selection
#pragma config UPLLEN   = ON

```

Step 3:

Design the USB Device, Configuration and Class specific descriptors that are required for successful enumeration of the USB device. These should comply with the USB 2.0 specification. Class Specific descriptors should comply with the class specifications. The device and configuration descriptors should be added to the Master Descriptor Table. A Function Driver Registration Table should be created and registered with the Device Layer. The function driver registration table associates instance of a function driver with a Device Layer instance. Refer to "Setting up Master Descriptor Table" and "Function Driver Registration" in the Device Layer help file for more information.

Step 4:

The application should create a Device Layer Event Handler which will handle all the events generated by the Device Layer. The following code snippet shows a list of all possible events that are generated by the Device Layer. The code snippet can be used in the application and updated to meet the application requirements. Refer to "Device Events" in the Device Layer documentation for more details on the Device Layer Event Handling.

```

void APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, USB_DEVICE_EVENT_DATA * eventData)
{
    switch(event)
    {
        case USB_DEVICE_EVENT_RESET:
            break;
        case USB_DEVICE_EVENT_DECONFIGURED:
            break;
        case USB_DEVICE_EVENT_CONFIGURED:
            break;
        case USB_DEVICE_EVENT_SUSPENDED:
            break;
        case USB_DEVICE_EVENT_RESUMED:
            break;
    }
}

```

```

        case USB_DEVICE_EVENT_ATTACHED:
            break;
        case USB_DEVICE_EVENT_DETACHED:
            break;
        case USB_DEVICE_EVENT_ERROR:
            break;
        default:
            break;
    }
}

```

Step 5:

The application should create a Function Driver Event Handler which will handle all the events generated by the Function Driver. Refer to "Event Handling" section in the applicable function driver documentation for more details on the Function Driver Event Handling.

Step 6:

If the application intends to operate the USB Device Library in polling mode, the a interrupt service routine is not required. If the application intends to operate in interrupt mode, a USB Interrupt Handler should be defined as shown here:

```

/* Use this for PIC32MX */
void __ISR ( _USB_1_VECTOR ) _InterruptHandler_USB_stub ( void )
{
    DRV_USB_Tasks_ISR((SYS_MODULE_OBJ)0);
}

/* Use this for PIC32MZ */
void __ISR ( _USB_VECTOR,ipl4 ) _InterruptHandler_USB_stub ( void )
{
    DRV_USB_Tasks_ISR((SYS_MODULE_OBJ)0);
}

```

Step 7:

The application should create a while(1) loop that continuously updates the USB Device Layer State machine and the application state machine. This requires the application state machine to be non-blocking.

```

/* This while(1) loop will continuously update Device Layer State machine
 * and the application state machine */
while(1)
{
    USB_DEVICE_Tasks(deviceLayerObject);
    APP_Tasks();
}

```

Step 8:

If interrupt based operation is needed, the interrupts need to be enabled first. The application should then initialize and open the device layer. When the device layer is opened successfully, a device layer event handler can be registered. The device can then be attached on the bus. Refer to the "Initializing the Library" section in the USB Device Layer documentation for more details on initializing the USB Device Layer

```

/* Initialize the interrupt system */
SYS_INT_Initialize();

/* Initialize the global interrupts */
SYS_INT_Enable();

SYS_INT_VectorPrioritySet(INT_VECTOR_USB, INT_PRIORITY_LEVEL3);
SYS_INT_VectorSubprioritySet(INT_VECTOR_USB, INT_SUBPRIORITY_LEVEL3);

/* Initialize the USB device layer */
deviceLayerObject = USB_DEVICE_Initialize (USB_DEVICE_INDEX_0 ,

```

```

        ( SYS_MODULE_INIT* ) & usbDevInitData);

/* check if the object returned by the device layer is valid */
SYS_ASSERT((SYS_MODULE_OBJ_INVALID != deviceLayerObject), "Invalid USB DEVICE object");

/* Open an instance of the device layer */
appData.deviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0,
                                         DRV_IO_INTENT_READWRITE );

/* Register a callback with device layer to get
 * event notification (for end point 0) */

USB_DEVICE_EventCallBackSet(appData.deviceHandle, APP_USBDeviceEventHandler);

/* Attach the device */

USB_DEVICE_Attach(appData.deviceHandle);

```

5.9.1.7 USB Device Stack Migration Guide

5.9.1.7.1 Introduction

This topic provides information for migrating from USB Device from MLA to MPLAB Harmony.

5.9.1.7.2 Source Files to Include

This topic provides information on the source files to be included when migrating from USB device projects from MLA to MPLAB Harmony.

Description

The following table lists the source files that must be included for MLA based and MPLAB Harmony based USB device projects. In MLA there is no separate controller driver implementation. Both enumeration functionality and controller driver is implemented in `usb_device.c` file. In MPLAB Harmony USB Device Stack, the controller driver is implemented in `drv_usb.c` and `drv_usb_device.c` files and enumeration functionality is implemented in `usb_device.c` file.

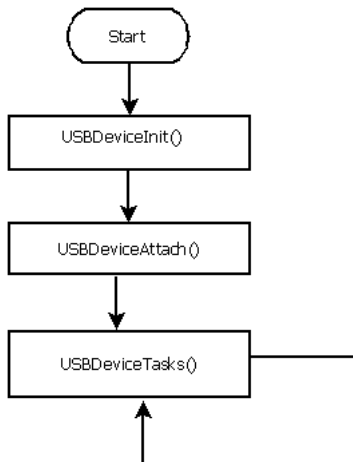
MLA	MPLAB Harmony	Description
<code>usb_device.c</code>	<code>usb_device.c</code>	In MLA, this file contains enumeration functionality and controller driver implementations. In MPLAB Harmony, this file implements enumeration functionality only.
<code>usb_function_xxx.c</code> (Where xxx stands for function driver name like hid, msd, etc.)	<code>usb_device_xxx.c</code> (Where xxx stands for function driver name like hid, msd, etc.)	Contains all function driver implementations.
<code>usb_descriptors.c</code>	<code>system_config.c</code>	Contains USB stack configurations
N/A	<code>drv_usb.c</code> and <code>drv_usb_device.c</code>	USB peripheral driver implementation.

5.9.1.7.3 Initializing the USB Device Stack

This topic provides initialization information when migrating a USB device project from MLA to MPLAB Harmony.

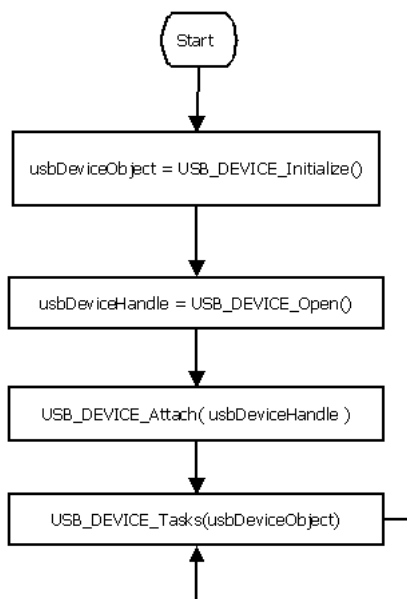
Description

In MLA, initializing the USB device stack consists of calling USB functions in the sequence shown in the following flow chart.



`USBDeviceInit()` function will initialize the USB Device stack. `USBDeviceAttach()` function will attach the device to bus. `USBDeviceTasks()` function will run the stack task routine.

In MPLAB Harmony USB Device Stack, initializing the USB device stack consists of calling USB functions in the sequence shown in the following chart.



The `USB_DEVICE_Initialize()` function will initialize the USB device stack. On successful initialization, this function returns a valid system object. `USB_DEVICE_Open()` returns a client handle to the application. The application must use this client handle for any further communication with the USB device stack. `USB_DEVICE_Attach()` function attaches the device to USB bus. `USB_DEVICE_Tasks()` will run the task routine of the stack.

5.9.1.7.4 Configuring the Stack

This topic provides stack configuration information when migrating from a USB device project from MLA to MPLAB Harmony.

Description

To configure the stack in MLA, USB descriptors have to be defined in the `usb_descriptors.c` file. The configuration is done in `usb_config.h`. Whereas in MPLAB Harmony, USB descriptors are defined in `system_config.c`. The MPLAB Harmony USB Device Stack configuration is done in `system_config.h`. Apart from descriptors, user also will have to define a function registration table and a master descriptor table in `system_config.c`. Refer to USB device help file of MPLAB Harmony for more information on the function registration table and the master descriptor table.

5.9.1.7.5 Event Handling

This topic provides event handling information when migrating a USB device project from MLA to MPLAB Harmony.

Description

In MLA, user will have to handle events inside `USER_USB_CALLBACK_EVENT_HANDLER` function. Whereas in case of MPLAB Harmony, user application client will have to register an event handler with the USB device stack using `USB_DEVICE_EventCallBackSet()` function. All USB bus events are then forwarded to this user registered event handler function. In case of multi client environment, it is possible for each client to register its own event handler with the USB stack. This allows each application clients to individually manage the events separately. Following code snippet shows an example of registering the event handler in case of MPAB Harmony stack using `USB_DEVICE_EventCallBackSet()` function. In the following code snippet, `APP_UsbDeviceEventCallBack()` function is the application event handler.

```
/* Initialize the USB device layer */
usbDevObject = USB_DEVICE_Initialize (USB_DEVICE_INDEX_0 ,
                                     { SYS_MODULE_INIT* } & usbDevInitData);

/* open an instance of the device layer */
usbDevHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0, DRV_IO_INTENT_READWRITE );

/* Register a callback with device layer to get event notification (for and point 0) */
USB_DEVICE_EventCallBackSet(usbDevHandle, &APP_UsbDeviceEventCallBack);

USB_DEVICE_HID_EventHandlerSet(0, APP_usbHID_CallBack, (uintptr_t)NULL );

USB_DEVICE_Attach(usbDevHandle);
```

The following code snippet shows the implementation of `APP_UsbDeviceEventCallBack` function.

```
void APP_UsbDeviceEventCallBack(USB_DEVICE_EVENT event, USB_DEVICE_EVENT_DATA * eventData)
{
    if((event == USB_DEVICE_EVENT_DECONFIGURED) || (event == USB_DEVICE_EVENT_RESET))
    {
        appData.deviceConfigured = false;
    }
    else if(event == USB_DEVICE_EVENT_CONFIGURED)
    {
        // Set the flag indicating device is configured.
        appData.deviceConfigured = true;

        // Save the other details for later use.
        appData.configurationValue = eventData->eventConfigured.configurationValue;
        appData.usbSpeed = eventData->eventConfigured.speed ;
    }
}
```

5.9.1.7.6 Initializing and Communicating with the Endpoint

This topic provides endpoint initialization and communication information when migrating a USB device project from MLA to MPLAB Harmony.

Description

In MLA, user application will have to explicitly initialize the required endpoints and can directly read and write to the endpoints. The following code snippet shows how MLA initializes the endpoints in event handler when the device is configured.

```

*****
BOOL USER_USB_CALLBACK_EVENT_HANDLER(int event, void *pdata, WORD size)
{
    switch(event)
    {
        case EVENT_CONFIGURED:
            //enable the HID endpoint
            USBEnableEndpoint(HID_EP,
                USB_IN_ENABLED|USB_OUT_ENABLED|USB_HANDSHAKE_ENABLED|USB_DISALLOW_SETUP);
            //Prepare the OUT endpoint for the next packet
            USBOutHandle = HIDRxPacket(HID_EP, (BYTE*)&ReceivedDataBuffer, 64);
            break;

        case EVENT_TRANSFER:
            //Add application specific callback task or callback function here if desired.
            break;

        case EVENT_SOF:
            USBCB_SOF_Handler();
            break;

        case EVENT_SUSPEND:
            USBCBSuspend();
    }
}

```

Endpoint configured in MLA

In case of MPLAB Harmony the user application has no direct access to endpoints. USB device layer initializes the endpoint and any communication to endpoint must happen through function driver APIs. The USB device layer initializes the endpoint based on the information provided by the user in the configurations descriptor. The USB device layer does this by parsing the descriptors table at run time. So the user does not have to explicitly enable or disable the endpoint.

5.9.1.7.7 Handling Endpoint 0 (EP0) Packets

This topic provides information for handling Endpoint 0 packets when migrating a USB device project from MLA to MPLAB Harmony.

Description

Standard device requests are handled by device stack in both MLA and MPLAB Harmony. The difference lies in handling class specific requests. In MLA, the class specific requests are forwarded to the application as EP0 events and application forwards them to appropriate function driver. The following code snippet shows how application forwards them to appropriate function driver in the event handler.

```

*****
BOOL USER_USB_CALLBACK_EVENT_HANDLER(int event, void *pdata, WORD size)
{
    switch(event)
    {
        case EVENT_EP0_REQUEST:
            USBCheckHIDRequest();
            break;

        case EVENT_TRANSFER:
            //Add application specific callback task or callback function here if desi
            break;

        case EVENT_CONFIGURED:
            //enable the HID endpoint
            USBEnableEndpoint(HID_EP,
                USB_IN_ENABLED|USB_OUT_ENABLED|USB_HANDSHAKE_ENABLED|USB_DISALLOW_SETUP)
    }
}

```

EP0 request is forwarded to HID

In the case of MPLAB Harmony USB Device Stack, EP0 events are generated from appropriate function driver as meaningful events. For example the HID function driver in MPLAB harmony parses the EP0 packet to generate events like USB_DEVICE_HID_EVENT_GET_IDLE, USB_DEVICE_HID_EVENT_SET_REPORT and USB_DEVICE_HID_EVENT_SET_IDLE etc.

The following code snippet shows how the event handler callback is registered with the HID function driver using `USB_DEVICE_HID_EventHandlerSet()` function.

```
/* Register a callback with device layer to get event notification (for end poi
USB_DEVICE_EventCallbackSet(usbDevHandle, &APP_UsbDeviceEventCallback);

USB_DEVICE_HID_EventHandlerSet(USB_DEVICE_HID_INDEX_0, APP_usbHID_Callback,
                               (uintptr_t)NULL );
```

The following code snippet shows the implementation of the HID event handler function in the application. Note how the HID function driver converts EP0 requests to meaningful format for the application.

```
USB_DEVICE_HID_EVENT_RESPONSE APP_usbHID_Callback ( USB_DEVICE_HID_INDEX iHID,
                                                    USB_DEVICE_HID_EVENT event,
                                                    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
                                                    void * eventData,
                                                    uintptr_t userData )
{
    USB_DEVICE_HID_EVENT_DATA_REPORT_SENT * reportSent;
    USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED * reportReceived;

    /* Check type of event */
    switch (event)
    {
        case USB_DEVICE_HID_EVENT_SET_IDLE:
            /* For now we just accept this request as is. We acknowledge
             * this request using the USB_DEVICE_HID_ControlStatus()
             * function with a USB_DEVICE_CONTROL_STATUS_OK flag */
            USB_DEVICE_HID_ControlStatus(iHID, controlTransferHandle,
                                         USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_HID_EVENT_REPORT_SENT:
            /* The eventData parameter will be USB_DEVICE_HID_EVENT_REPORT_SENT
             * pointer type containing details about the report that was
             * sent. */
            reportSent = (USB_DEVICE_HID_EVENT_DATA_REPORT_SENT *) eventData;
            if(reportSent->transferHandle == appData.txTransferHandle )
            {
                // Transfer progressed.
```

5.9.2 USB Audio Device Library

5.9.2.1 Introduction

Introduces the MPLAB Harmony USB Device Audio Library.

Description

This library provides a high-level abstraction of the Universal Serial Bus (USB) Audio Device Class function driver that is part of the MPLAB Harmony USB stack with a convenient C language interface. This library supports revision 1.0 of the USB Audio specification release by the USB Implementers forum.

The Audio Device Class Definition applies to all devices or functions embedded in composite devices that are used to manipulate audio, voice, and sound-related functionality. This includes both audio data (analog and digital) and the functionality that is used to directly control the audio environment, such as Volume and Tone Control. Audio device can also MIDI Data stream through USB.

USB Audio is used with many applications related to voice telephony, audio playback, and recording eg:- Speaker, Microphone,

Mixer etc.

The MPLAB Harmony USB Device Audio library offers services to the application to interact and respond to the host requests. The MPLAB Harmony USB Device Audio library is also referred to as Audio function driver in this document.

5.9.2.2 Release Notes

MPLAB Harmony Version: v0.70b **USB Audio Device Library Version :** 0.01 Alpha **Release Date:** 18Nov2013

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

New This Release:

Nothing to report in this release.

Known Issues:

Nothing to report in this release.

5.9.2.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.9.2.4 Library Architecture

Provides an Architectural Overview of the Audio Function Driver.

Description

The USB Audio function driver offers various services to a USB Audio device to communicate with the host by abstracting USB specification details. It must be used along with the USB Device layer and USB controller to communicate with the USB host. Figure 1 shows a block diagram representation of the MPLAB Harmony USB Architecture and where the USB Audio Function Driver is placed.

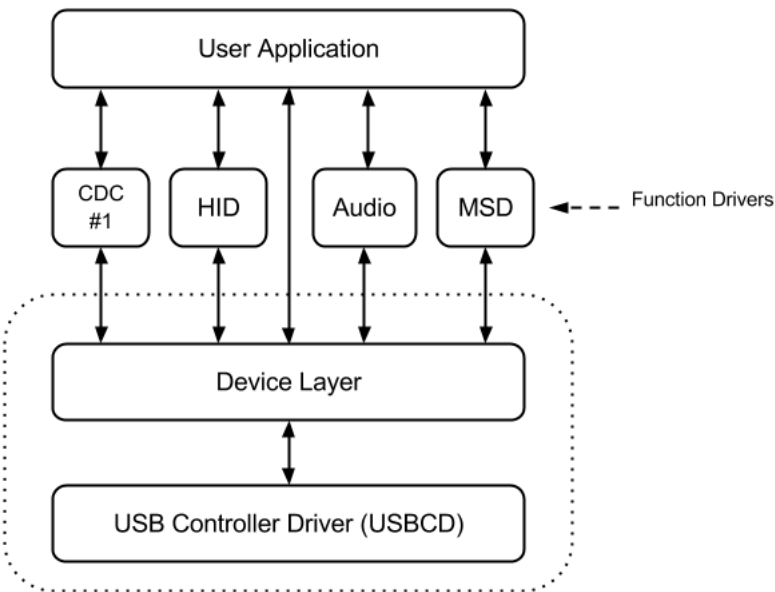


Figure 1: USB Device Audio Function Driver

The USB controller driver takes the responsibility of managing the USB peripheral on the device. The USB device layer handles the device enumeration etc. The Audio function driver receives The USB Device layer forwards all Audio specific control transfers to the Audio Function Driver. The Audio Function Driver interprets the control transfers and requests application's intervention through event handlers and well defined set of API. The application must respond to the Audio events either in or out of the event handler. The application must interact directly with the Audio Function Driver to [send](#)/receive data and Audio Control data.

5.9.2.5 Using the Library

5.9.2.5.1 Files to Include

Describes which files to include in project while using the Audio Function Driver.

Description

Table 2 shows the files that must be included in the project in order to use the Audio Function Driver. These files are located in the framework folder of the MPLAB Harmony installation. It is assumed that the Device Layer files (and the files needed by the Device Layer) are already included in the project.

Filename	Description
\framework\usb\src\dynamic\usb_device_audio.c	This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with the USB device Audio function driver.
\framework\usb\usb_device_audio.h	Must be included in every source file that needs to invoke Audio function driver API.
system_config.h	User created file which contains the Audio function driver configuration.

Table 2: Files to be included in USB Audio Device Project.

5.9.2.5.2 Initializing the Library

Describes how Audio function driver is initialized.

Description

The Audio function driver instance for a USB device configuration is initialized by the Device Layer when the configuration is set by the host. This process does not require application intervention. Each instance of the Audio function driver should be registered with the Device layer through the Device Layer Function Driver Registration Table. The Audio function driver does not require a initialization data structure. The funcDriverInit member of the function driver registration table entry for the Audio function driver instance should be set to NULL. The audioFuncDriver object is a global object provided by the Audio function driver and points to the Audio function driver - Device Layer interface functions which are required by the Device Layer. The code snippet below shows an example of how multiple instances of Audio Function driver can registered with the Device Layer.

```
/* This code snippet shows an example of how an Audio function
 * driver instances can be registered with the Device Layer
 * via the Device Layer Function Driver Registration Table.
 * In this case Device Configuration 1 consists of one Audio
 * function driver instance. */

const USB_DEVICE_FUNC_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
        .speed = USB_SPEED_FULL|USB_SPEED_HIGH,    // Supported speed
        .configurationValue = 1,                   // To be initialized for Configuration 1
        .interfaceNumber = 0,                       // Starting interface number.
        .numberOfInterfaces = 2,                    // Number of interfaces in this instance
        .funcDriverIndex = 0,                       // Function Driver instance index is 0
        .funcDriverInit = NULL,                     // Function Driver does not need
        initialization data structure
        .driver = &audioFuncDriver                  // Pointer to Function Driver - Device
        Layer interface functions
    },
};
```

The sequence for initializing the USB Audio Function driver is depicted in the Figure 2.

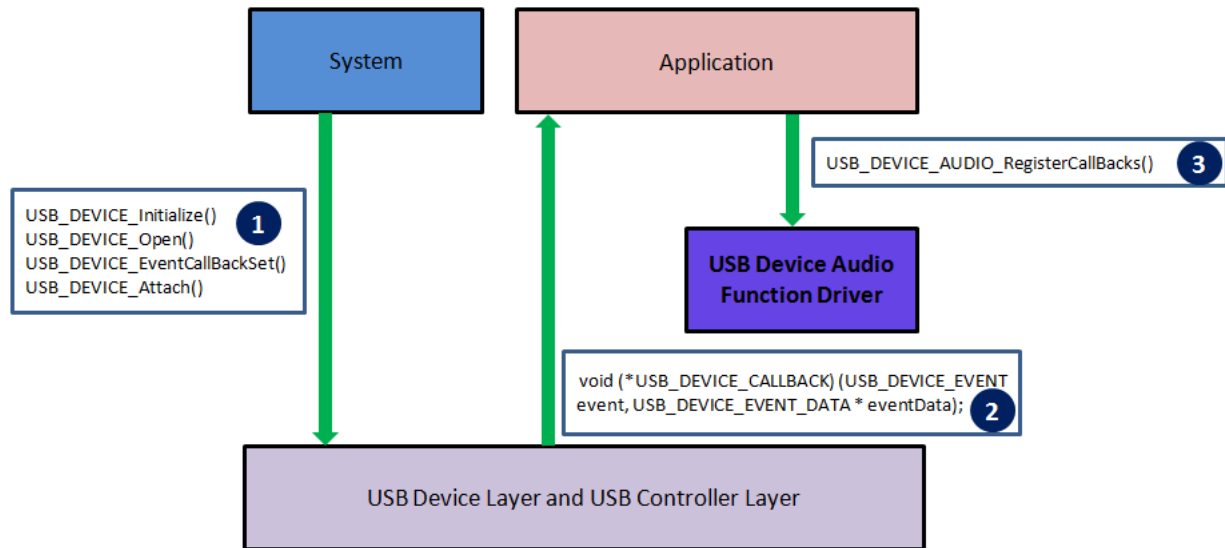


Figure 2. USB Audio function driver Initialization

1. Call set of APIs to initialize USB Device Layer. Refer USB Device Layer documentation for details about these APIs.
2. The Device Layer provides a callback to the application for any USB Device events like attached, powered, configured etc. The application should receive a callback with an event `USB_DEVICE_EVENT_CONFIGURED` to proceed.
3. Once the Device Layer is configured the application needs to register callback functions with USB Audio function driver to receive Audio Control transfers and also other audio function driver events. Now the application can use audio function driver APIs to communicate with USB Host.

5.9.2.5.3 Event Handling

Describes Audio function driver event handler registration and event handling.

Description

Registering Audio Function Driver callback functions:

While creating USB Audio Device based application, an event handler must be registered with the Device Layer (the Device Layer Event Handler) and every Audio function driver instance (Audio Function Driver Event Handler). The application needs to register two callback functions with Audio function driver

1. For receiving Audio Control Requests from Host like Volume Control, Mute Control etc.
2. For handling other events from Audio function driver. eg:- Data Write Complete or Data Read Complete

The callback functions should be registered before the USB device layer acknowledges the SET CONFIGURATION request from the USB Host. In order to ensure this, the callback functions should be set in the `USB_DEVICE_EVENT_CONFIGURED` event that is generated by the device layer. The code example below shows an example of how this can be done.

```
/* This a sample Application Device Layer Event Handler
 * Note how the Audio Function Driver callback functions
 * APP_AudioEntitySettingsCallback() and APP_AudioEventCallback()
 * are registered in the USB_DEVICE_EVENT_CONFIGURED event. */
```

```
void APP_USBDeviceEventCallback ( USB_DEVICE_EVENT event,
                                USB_DEVICE_EVENT_DATA * eventData )
```

```

{
    switch ( event )
    {
        case USB_DEVICE_EVENT_RESET:
        case USB_DEVICE_EVENT_DECONFIGURED:

            // USB device is reset or device is deconfigured.
            // This means that USB device layer is about to deinitalize
            // all function drivers.

            break;

        case USB_DEVICE_EVENT_CONFIGURED:

            /* check the configuration */
            if ( eventData->eventConfigured.configurationValue == 1)
            {

                /* Register the callback function for handling Audio Control requests.*/
                USB_DEVICE_AUDIO_RegisterCallbacks (USB_DEVICE_CDC_INDEX_0 ,
                                                    APP_AudioEntitySettingsCallback ,
                                                    USB_DEVICE_AUDIO_CALLBACK_ENTITY_SETTINGS );

                /* Register the callback function for handling Audio function driver Events*/
                USB_DEVICE_AUDIO_RegisterCallbacks (USB_DEVICE_CDC_INDEX_0 ,
                                                    APP_AudioEventCallback ,
                                                    USB_DEVICE_AUDIO_CALLBACK_EVENTS );

                /* mark that set configuration is complete */
                appData.isConfigured = true;

            }
            break;

        case USB_DEVICE_EVENT_SUSPENDED:

            break;

        case USB_DEVICE_EVENT_RESUMED:
        case USB_DEVICE_EVENT_ATTACHED:
        case USB_DEVICE_EVENT_DETACHED:
        case USB_DEVICE_EVENT_ERROR:
        default:
            break;
    }
}

```

Handling Audio Control Requests:

The audio function driver notifies the application by calling the provided callback function, whenever an Audio Control request is received from USB Host. Below example shows how to handle Audio Control request.

// This code example shows handling Audio Control requests. The below handles a Mute request (both SET GET) received from USB Host.

```

//The below are ID of each terminal/Unit in the USB audio function.
//Device makes these IDs known to Host through USB descriptors
// if that terminal/unit is present in this USB audio device application.
#define APP_ID_INPUT_TERMINAL 0x01
#define APP_ID_OUTPUT_TERMINAL 0x02
#define APP_ID_MIXER_UNIT 0x03
#define APP_ID_SELECTOR_UNIT 0x04
#define APP_ID_FEATURE_UNIT 0x05

```

```

#define APP_ID_PROCESSING_UNIT 0x06
#define APP_ID_EXTENSION_UNIT 0x07

int APP_AudioEntitySettingsCallback
(
    USB_DEVICE_AUDIO_INDEX iAudio ,
    uintptr_t controlHandle,
    SETUP_PKT *setupPkt
)
{
    int8_t err=-1;
    USB_AUDIO_CONTROL_INTERFACE_REQUEST* controlRequest;
    controlRequest = (USB_AUDIO_CONTROL_INTERFACE_REQUEST*) setupPkt;
    switch(controlRequest->entityID)
    {
        case APP_ID_INPUT_TERMINAL:
        case APP_ID_OUTPUT_TERMINAL:
            err = APP_TerminalRequestHandler (controlRequest);
            break;

        case APP_ID_FEATURE_UNIT:
            err = APP_FeatureUnitRequestHandler (iAudio, controlHandle, controlRequest);
            break;
        case APP_ID_MIXER_UNIT:
        case APP_ID_SELECTOR_UNIT:
        case APP_ID_PROCESSING_UNIT:
        case APP_ID_EXTENSION_UNIT:
        default:
            //This USB Audio Speaker application does not support any other control request
            // received for other unit from Host. So Stall if any other request recieved from
            Host.
            USB_DEVICE_ControlStatus(appData.usbDevHandle,
            controlHandle,USB_DEVICE_CONTROL_STATUS_STALL);
            break;
    }//end of switch(controlRequest->entityID)
    return err;
}//end of function APP_AudioEntitySettingsCallback

int APP_FeatureUnitRequestHandler
(
    USB_DEVICE_AUDIO_INDEX iAudio ,
    uintptr_t controlHandle,
    USB_AUDIO_CONTROL_INTERFACE_REQUEST* controlRequest
)
{
    USB_AUDIO_FEATURE_CONTROL_REQUEST* featureRequest;
    featureRequest = (USB_AUDIO_FEATURE_CONTROL_REQUEST*) controlRequest;

    int8_t err = 0;

    switch(featureRequest->controlSelector)
    {
        case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_MUTE_CONTROL:
        {
            if (featureRequest->bRequest == USB_AUDIO_CLASS_SPECIFIC_REQUEST_CODE_SET_CUR )
            {
                //A control write transfer recived from Host. Now recieve data from Host.
                USB_DEVICE_ControlReceive(appData.usbDevHandle, controlHandle, (uint8_t *)
                &DacMute, 1 );
            }
            else if(featureRequest->bRequest == USB_AUDIO_CLASS_SPECIFIC_REQUEST_CODE_GET_CUR)
            {
                /*Handle Get request*/
                USB_DEVICE_ControlSend(appData.usbDevHandle, controlHandle, (uint8_t
                *)&DacMute, 1 );
            }
        }
    }
}

```

```

    }
    else
    {
        USB_DEVICE_ControlStatus( appData.usbDevHandle, controlHandle,
        USB_DEVICE_CONTROL_STATUS_STALL);
    }
}
break;

case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_VOLUME_CONTROL:
case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_BASS_CONTROL:
case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_MID_CONTROL:
case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_TREBLE_CONTROL:
case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_GRAPHIC_EQUALIZER_CONTROL:
case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_AUTOMATIC_GAIN_CONTROL:
case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_DELAY_CONTROL:
case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_BASS_BOOST_CONTROL:
case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_LOUDNESS_CONTROL:
case USB_AUDIO_FEATURE_UNIT_CONTROL_SELECTOR_FU_CONTROL_UNDEFINED:
default:
    //This USB Audio Speaker application does not support any other feature unit
    request
    // from Host. So Stall if any other feature unit request recieved from Host.
    USB_DEVICE_ControlStatus (appData.usbDevHandle,controlHandle,
    USB_DEVICE_CONTROL_STATUS_STALL);
    break;

} // end of switch(featureRequest->controlSelector)
return err;
} //end of function APP_FeatureUnitRequestHandler

```

Handling Audio function driver events :

The audio function driver events are discussed at [USB_DEVICE_AUDIO_EVENT Enumeration](#).

5.9.2.5.4 Transferring Data

Describes how to [send](#)/receive data to/from USB Host using this USB Audio function driver.

Description

The USB Audio function driver offers API to [send](#), receive data.

Receiving Data :

USB_DEVICE_AUDIO_Read Function can be used to schedule a data read from the USB Host. Whenever there is data available from Host, the USB audio function driver calls back USB audio event handler with an event USB_DEVICE_AUDIO_EVENT_READ_COMPLETE. User can take necessary actions once the data is received like playback the audio samples through a CODEC and/or schedule next read request for audio samples.

Sending Data :

USB_DEVICE_AUDIO_Write Function can be used to schedule a data [send](#) to the Host. The data will be [send](#) to Host, whenever it is ready to receive. Once the data [send](#) is complete the audio function driver callback USB audio event handler with an event USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE. User can take necessary action once the data write is complete like schedule next data write, etc.

5.9.2.6 Configuring the Library

Describes how to configure the Audio Function Driver.

Description

The application designer must specify the following configuration parameters while using the Audio Function Driver. The configuration macros that implement these parameters must be located in the `system_config.h` file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

Configuration Macro Name	Description	Comments
USB_DEVICE_AUDIO_INSTANCES_NUMBER	Number of Audio Function Driver Instances in the application	This macro defines the number of instances of the Audio Function Driver. For example, if the application needs to implement two instances of the Audio Function Driver on one USB Device, the macro should <code>USB_DEVICE_AUDIO_INSTANCES_NUMBER</code> be set to 2. Note that implementing a USB Device that features multiple Audio interfaces requires appropriate USB configuration descriptors.
USB_DEVICE_AUDIO_QUEUE_SIZE	Audio Function Driver Buffer Queue size	This macro defines the Audio function driver internal buffer queue. The choice of this parameter defines the readiness of the function driver to respond successfully to read and write requests. The queue is shared by <code>USB_DEVICE_AUDIO_INSTANCES_NUMBER</code> number of Audio instances. A buffer in the queue is allocated to a read, write or a serial state notification request. In a simple application, that does not require buffer queuing and serial state notification, the <code>USB_DEVICE_AUDIO_QUEUE_SIZE</code> macro can be set to 2. In a case where higher throughput is required, this parameter should be set to a higher value. Note that increasing the queue size requires more RAM.

5.9.2.7 Library Interface

Data Types and Constants

	Name	Description
	<code>USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE</code>	USB Device Audio Function Driver Event Handler Response Type None.
	<code>USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID</code>	USB Device Audio Function Driver Invalid Transfer Handle Definition.
	<code>USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE</code>	USB Device Audio Function Driver Control Transfer Handle
	<code>USB_DEVICE_AUDIO_EP_INSTANCE</code>	This is type <code>USB_DEVICE_AUDIO_EP_INSTANCE</code> .
	<code>USB_DEVICE_AUDIO_EVENT</code>	USB Device Audio Function Driver Control Transfer Status

	USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE	USB Device Audio Function Driver Read and Write Complete Event Data.
	USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE	USB Device Audio Function Driver Read and Write Complete Event Data.
	USB_DEVICE_AUDIO_EVENT_HANDLER	USB Device Audio Event Handler Function Pointer Type.
	USB_DEVICE_AUDIO_EVENT_RESPONSE	USB Device Audio Function Driver Event Callback Response Type
	USB_DEVICE_AUDIO_INDEX	USB Device Audio Function Driver Index
	USB_DEVICE_AUDIO_INSTANCE	This is type USB_DEVICE_AUDIO_INSTANCE .
	USB_DEVICE_AUDIO_INSTANCE_STATE	This is type USB_DEVICE_AUDIO_INSTANCE_STATE .
	USB_DEVICE_AUDIO_INTERFACE_COLLECTION	This is type USB_DEVICE_AUDIO_INTERFACE_COLLECTION .
	USB_DEVICE_AUDIO_INTERFACE_INFO	This is type USB_DEVICE_AUDIO_INTERFACE_INFO .
	USB_DEVICE_AUDIO_INTERFACE_TYPE	This is type USB_DEVICE_AUDIO_INTERFACE_TYPE .
	USB_DEVICE_AUDIO_IRP_OBJECT	USB Device Audio Function Driver IRP object.
	USB_DEVICE_AUDIO_RESULT	USB Device Audio Function Driver USB Device Audio Result enumeration.
	USB_DEVICE_AUDIO_STREAMING_INTERFACE	This is type USB_DEVICE_AUDIO_STREAMING_INTERFACE .
	USB_DEVICE_AUDIO_STREAMING_INTERFACE_ALTERNATE_SETTING	This is type USB_DEVICE_AUDIO_STREAMING_INTERFACE_ALTERNATE_SETTING .
	USB_DEVICE_AUDIO_TRANSFER_HANDLE	USB Device Audio Function Driver Transfer Handle Definition.

Functions

	Name	Description
	USB_DEVICE_AUDIO_ControlReceive	This function allows the application to respond to the Audio function driver specific control transfer requests. It allows the application to receive data from the host in the data stage of the control transfer.
	USB_DEVICE_AUDIO_ControlSend	This function allows the application to respond to the Audio function driver specific control transfer requests. It allows the application to send data to the host in the data stage of the control transfer.
	USB_DEVICE_AUDIO_ControlStatus	This function allows the application to complete the status stage of the the Audio Function Driver specific control transfer request.
	USB_DEVICE_AUDIO_EventHandlerSet	This function registers an event handler for the specified Audio function driver instance.

	USB_DEVICE_AUDIO_Read	This function requests a data read from the USB Device Audio Function Driver Layer.
	USB_DEVICE_AUDIO_Write	This function requests a data write to the USB Device Audio Function Driver Layer.

Description

This section describes the Application Programming Interface (API) functions of the USB Device Audio Library

Refer to each section for a detailed description.

5.9.2.7.1 Functions

5.9.2.7.1.1 USB_DEVICE_AUDIO_ControlReceive Function

C

```
USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_ControlReceive(
    USB_DEVICE_AUDIO_INDEX instanceIndex,
    USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * data,
    size_t size
);
```

Description

This function allows the application to respond to the Audio function driver specific control transfer requests. It allows the application to receive data from the host in the data stage of the control transfer.

For example, when the Host wants to set current Volume settings to Device, the Host first sends a SETUP packet to the Device indicating that it intends to set Volume. After receiving the SETUP packet the Device should be prepared to receive current Volume from Host. Application should make use of USB_DEVICE_AUDIO_ControlReceive() function to receive data from the Host in such cases. The function can be called in the Audio Function Driver event handler or can be called outside the event handler. Calling this function outside the event handler defers the response to the event. This allows the application to prepare the data buffer out of the event handler context, especially if the data buffer to receive the data is not readily available. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device. The application must use the control transfer handle that was generated along with the event as the controlTransferHandle.

Preconditions

This function should only be called in response to a Audio function driver event that requires a control transfer response.

Parameters

Parameters	Description
instance	USB Device Audio Function Driver instance.
controlTransferHandle	Control Transfer handle for the control transfer.
data	Data buffer to receive the data stage of the control transfer.
size	size (in bytes) of the data to sent.

Returns

USB_DEVICE_AUDIO_RESULT_ERROR_CONTROL_TRANSFER_FAILED - The request was not successful because the USB host cancelled the control transfer.

USB_DEVICE_AUDIO_RESULT_OK - The request was successful.

Example

```

// -----
// In this example, the application receives current Mute info
// with in the event handler. The application uses the
// USB_DEVICE_AUDIO_ControlReceive() function to do this.
// -----
uint8_t DACMute = 1; // Holds the status of Mute Control
USB_DEVICE_AUDIO_EVENT_RESPONSE USBDeviceAudioEventHandler
(
    USB_DEVICE_AUDIO_INDEX instanceIndex,
    USB_DEVICE_AUDIO_EVENT event,
    USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    USB_DEVICE_AUDIO_EVENT_DATA * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_AUDIO_EVENT_ENTITY_SETTINGS_RECEIVED:
            //In this example we have received a Set Mute request from the Host.

            // Typcast the recived data into interface request.
            controlRequest = (USB_AUDIO_CONTROL_INTERFACE_REQUEST*) pData;

            //check entity ID (Note: Entity IDs are specified in the USB device descriptor.
            switch(controlRequest->entityID)
            {
                case APP_ID_FEATURE_UNIT :
                    featureRequest = (USB_AUDIO_FEATURE_CONTROL_REQUEST*)pData;
                    switch(featureRequest->controlSelector)
                    {
                        case USB_AUDIO_FCS_MUTE_CONTROL:
                            {
                                if(featureRequest->bRequest == USB_AUDIO_CSRC_SET_CUR)
                                {
                                    //A control write transfer recived from Host. Now recieve data
                                    USB_DEVICE_AUDIO_ControlReceive(iAudio, controlHandle,(uint8_t *)
from Host.
                                    &DACMute, 1 );
                                }
                                break;
                            }
                        break;
                    }
                break;
            }
            break;

        case USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_RECEIVED:
            USB_DEVICE_ControlStatus(appData.usbDevHandle,controlTransferHandle,
            USB_DEVICE_CONTROL_STATUS_OK );
            if (appData.currentInterfaceAlternateSetting == 1)
                AK4953ADACMute(pCodecHandle, DACMute);
            break;
    }
    Remarks:    None.
}

```

5.9.2.7.1.2 USB_DEVICE_AUDIO_ControlSend Function**C**

```

USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_ControlSend(
    USB_DEVICE_AUDIO_INDEX instanceIndex,
    USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * data,
    size_t size
);

```

Description

This function allows the application to respond to the Audio function driver specific control transfer requests. It allows the application to **send** data to the host in the data stage of the control transfer. For example, if the Host requests current Volume settings to Device, then the Device needs to **send** a control transfer data to the Host. The function can be called in the Audio Function Driver event handler or can be called outside the event handler. Calling this function outside the event handler defers the response to the event. This allows the application to prepare the response data out of the event handler context, especially if the data is not readily available. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device. The application must use the control transfer handle that was generated along with the event as the controlTransferHandle.

Preconditions

This function should only be called in response to a Audio function driver event that requires a control transfer response.

Parameters

Parameters	Description
instance	USB Device Audio Function Driver instance.
controlTransferHandle	Control Transfer handle for the control transfer.
data	Data that represents the data stage of the control transfer.
size	size (in bytes) of the data to sent.

Returns

USB_DEVICE_AUDIO_RESULT_ERROR_CONTROL_TRANSFER_FAILED - The request was not successful because the USB host cancelled the control transfer.

USB_DEVICE_AUDIO_RESULT_OK - The request was successful.

Example

```
// -----
// In this example, the application responds to an Audio Control request
// with in the event handler. The application uses the
// USB_DEVICE_AUDIO_ControlSend() function to do this.
// -----
uint8_t DACMute = 1; // Holds the status of Mute Control
USB_DEVICE_AUDIO_EVENT_RESPONSE USBDeviceAudioEventHandler
(
    USB_DEVICE_AUDIO_INDEX instanceIndex,
    USB_DEVICE_AUDIO_EVENT event,
    USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    USB_DEVICE_AUDIO_EVENT_DATA * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_AUDIO_EVENT_ENTITY_SETTINGS_RECEIVED:
            //In this example we have received a Get Mute request from the Host.

            // Typcast the recived data into interface request.
            controlRequest = (USB_AUDIO_CONTROL_INTERFACE_REQUEST*) pData;

            //check entity ID (Note: Entity IDs are specified in the USB device descriptor.
            switch(controlRequest->entityID)
            {
                case APP_ID_FEATURE_UNIT :
                    featureRequest = (USB_AUDIO_FEATURE_CONTROL_REQUEST*)pData;
                    switch(featureRequest->controlSelector)
                    {
                        case USB_AUDIO_FCS_MUTE_CONTROL:
```

```

        {
            if(featureRequest->bRequest == USB_AUDIO_CSRC_GET_CUR)
            {
                // In this case, the application should send the current mute
                // status to the host. The application must send the
                // USB_DEVICE_AUDIO_ControlSend() function to send the data.
                USB_DEVICE_AUDIO_ControlSend(iAudio, controlHandle, (uint8_t
*)&DACMute, 1 );
            }
        }
        break;
    }
    break;
}
break;
}
Remarks:    None.

```

5.9.2.7.1.3 USB_DEVICE_AUDIO_ControlStatus Function

C

```

USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_ControlStatus(
    USB_DEVICE_AUDIO_INDEX instanceIndex,
    USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    USB_DEVICE_AUDIO_CONTROL_STATUS status
);

```

Description

This function allows the application to complete the status stage of the the Audio Function Driver specific control transfer request. The application can either **accept** the data stage or stall it. Calling this function with status set to USB_DEVICE_AUDIO_CONTROL_STATUS_OK will acknowledge the status stage of the control transfer. The control transfer can be stalled by setting the status parameter to USB_DEVICE_AUDIO_CONTROL_STATUS_STALL. The function can be called in the Audio Function Driver event handler or can be called outside the event handler. Calling this function outside the event handler defers the response to the event. This allows the application to analyze the event response outside the event handler. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device. The application must use the control transfer handle that was generated along with the event as the controlTransferHandle. The application must be aware of events and associated control transfers that do or do not require data stages. Incorrect usage of the USB_DEVICE_AUDIO_ControlStatus() function could cause the device function to be non-compliant.

Preconditions

This function should only be called in response to a Audio function driver event that requires a control transfer response.

Parameters

Parameters	Description
instance	USB Device Audio Function Driver instance.
controlTransferHandle	Control Transfer handle for the control transfer.
status	Set to USB_DEVICE_AUDIO_CONTROL_STATUS_OK to acknowledge the control transfer. Set to USB_DEVICE_AUDIO_CONTROL_STATUS_STALL to stall the transfer.

Returns

USB_DEVICE_AUDIO_RESULT_ERROR_CONTROL_TRANSFER_FAILED - The request was not successful because the USB host cancelled the control transfer.

USB_DEVICE_AUDIO_RESULT_OK - The request was successful.

Example

Remarks: None.

5.9.2.7.1.4 USB_DEVICE_AUDIO_EventHandlerSet Function**C**

```
USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_EventHandlerSet(
    USB_DEVICE_AUDIO_INDEX iAudio,
    USB_DEVICE_AUDIO_EVENT_HANDLER eventHandler,
    uintptr_t context
);
```

Description

This function registers a event handler for the specified Audio function driver instance. This function should be called by the client when it receives a SET CONFIGURATION event from the device layer. A event handler must be registered for function driver to respond to function driver specific commands. If the event handler is not registered, the device layer will stall function driver specific commands and the USB device may not function.

Preconditions

This function should be called when the function driver has been initialized as a result of a set configuration.

Parameters

Parameters	Description
instance	Instance of the Audio Function Driver.
eventHandler	A pointer to event handler function.
context	Application specific context that is returned in the event handler.

Returns

USB_DEVICE_AUDIO_RESULT_OK - The operation was successful
 USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
 USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_HANDLER - The eventHandler parameter is NULL

Remarks

None.

Example

```
// This code snippet shows an example registering an event handler. Here
// the application specifies the context parameter as a pointer to an
// application object (appObject) that should be associated with this
// instance of the Audio function driver.
```

```
USB_DEVICE_AUDIO_RESULT result;

USB_DEVICE_AUDIO_EVENT_RESPONSE APP_USBDeviceAUDIOEventHandler
(
    USB_DEVICE_AUDIO_INDEX instanceIndex ,
    USB_DEVICE_AUDIO_EVENT event ,
    USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE controlTransferHandle
    void* pData, uintptr_t context
)
{
    // Event Handling comes here

    switch(event)
    {
        ...
    }
}
```

```

    return(USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE);

}

result = USB_DEVICE_AUDIO_EventHandlerSet ( 0 ,&APP_EventHandler, (uintptr_t) &appObject);
if(USB_DEVICE_AUDIO_RESULT_OK != result)
{
    SYS_ASSERT ( false , "invalid event handler function" );
}

```

5.9.2.7.1.5 USB_DEVICE_AUDIO_Read Function

C

```

USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_Read(
    USB_DEVICE_AUDIO_INDEX iAudio,
    USB_DEVICE_AUDIO_TRANSFER_HANDLE* transferHandle,
    uint8_t interfaceNum,
    void * data,
    size_t size
);

```

Description

This function requests a data read from the USB Device Audio Function Driver Layer. The function places a requests with driver, the request will get serviced as data is made available by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_AUDIO_EVENT_READ_COMPLETE event. The amount of data read and the transfer handle associated with the request is returned along with the event. The transfer handle expires when event handler for the USB_DEVICE_AUDIO_EVENT_READ_COMPLETE exits. If the read request could not be accepted, the function returns an error code and transferHandle will contain the value [USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID](#).

Preconditions

The function driver should have been configured.

Parameters

Parameters	Description
instance	USB Device Audio Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_AUDIO_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the read request was successful.
interfaceNum	The USB Audio streaming interface number on which read request is to be placed.
data	pointer to the data buffer where read data will be stored.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.

Returns

USB_DEVICE_AUDIO_RESULT_OK - The read request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The write request could not be added.

USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_SIZE_INVALID - The specified transfer size was not a multiple of endpoint size or is 0.

USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application

and is invalid.

Remarks

None

Example

```
// Shows an example of how to read. This assumes that
// driver was opened successfully.
USB_DEVICE_AUDIO_INDEX iAudio;
USB_DEVICE_AUDIO_TRANSFER_HANDLE transferHandle;
uint8_t interfaceNumber;
uint8_t rxBuffer[192];
USB_DEVICE_AUDIO_RESULT readRequestResult;
iAudio = 0; //specify the Audio Function driver instance number.
interfaceNumber = 1; //Specify the Audio Streaming interface number.
readRequestResult = USB_DEVICE_AUDIO_Read
(
    iAudio,
    &transferHandle,
    interfaceNumber,
    &rxBuffer,
    192
);

if(USB_DEVICE_AUDIO_RESULT_OK != readRequestResult)
{
    //Do Error handling here
}

// The completion of the read request will be indicated by the
// USB_DEVICE_AUDIO_EVENT_READ_COMPLETE event.
```

5.9.2.7.1.6 USB_DEVICE_AUDIO_Write Function

C

```
USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_Write(
    USB_DEVICE_AUDIO_INDEX iAudio,
    USB_DEVICE_AUDIO_TRANSFER_HANDLE* transferHandle,
    uint8_t interfaceNum,
    void * data,
    size_t size
);
```

Description

This function requests a data write to the USB Device Audio Function Driver Layer. The function places a requests with driver, the request will get serviced as data is requested by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE event. The amount of data written and the transfer handle associated with the request is returned along with the event in writeCompleteData member of the pData parameter in the event handler.

The transfer handle expires when event handler for the USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE exits. If the read request could not be accepted, the function returns an error code and transferHandle will contain the value [USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID](#).

Preconditions

The function driver should have been configured.

Parameters

Parameters	Description
instance	USB Device Audio Function Driver instance.

transferHandle	Pointer to a USB_DEVICE_AUDIO_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the write request was successful.
interfaceNum	The USB Audio streaming interface number on which the write request is to placed.
data	pointer to the data buffer contains the data to be written.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.

Returns

USB_DEVICE_AUDIO_RESULT_OK - The read request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The write request could not be added.

USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_SIZE_INVALID - The specified transfer size was not a multiple of endpoint size or is 0.

USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Remarks

None

Example

```
// Shows an example of how to write. This assumes that
// driver was opened successfully.
USB_DEVICE_AUDIO_INDEX iAudio;
USB_DEVICE_AUDIO_TRANSFER_HANDLE transferHandle;
uint8_t interfaceNumber;
uint8_t txBuffer[192];
USB_DEVICE_AUDIO_RESULT writeRequestResult;
iAudio = 0; //specify the Audio Function driver instance number.
interfaceNumber = 1; //Specify the Audio Streaming interface number.

writeRequestResult = USB_DEVICE_AUDIO_Write
(
    iAudio,
    &transferHandle,
    interfaceNumber,
    &txBuffer,
    192
);

if(USB_DEVICE_AUDIO_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

// The completion of the write request will be indicated by the
// USB_DEVICE_AUDIO_EVENT_Write_COMPLETE event.
```

5.9.2.7.2 Data Types and Constants**5.9.2.7.2.1 USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE Macro**

C

```
#define USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE
```

Description

USB Device Audio Function Driver Event Handler Response None

This is the definition of the Audio Function Driver Event Handler Response Type none.

Remarks

Intentionally defined to be empty.

5.9.2.7.2.2 USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID Macro

```
C
#define USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID ((USB_DEVICE_AUDIO_TRANSFER_HANDLE)(-1))
```

Description

USB Device Audio Function Driver Invalid Transfer Handle Definition

This definition defines a USB Device Audio Function Driver Invalid Transfer Handle. A Invalid Transfer Handle is returned by the USB_DEVICE_Audio_Write() and USB_DEVICE_Audio_Read() functions when the request was not successful.

Remarks

None.

5.9.2.7.2.3 USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE Type

```
C
typedef USB_DEVICE_CONTROL_TRANSFER_HANDLE USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE;
```

Description

USB Device Audio Function Driver Control Transfer Handle

This is returned by the Audio function driver event handler and should be used by the application while responding to Audio function driver control transfer requests.

Remarks

None.

5.9.2.7.2.4 USB_DEVICE_AUDIO_EP_INSTANCE Structure

```
C
typedef struct {
    uint8_t epAddr;
    uint16_t epMaxPacketSize;
    USB_DEVICE_AUDIO_IRP_OBJECT irpObject[USB_DEVICE_AUDIO_QUEUE_SIZE];
} USB_DEVICE_AUDIO_EP_INSTANCE;
```

Description

This is type USB_DEVICE_AUDIO_EP_INSTANCE.

Members

Members	Description
uint8_t epAddr;	End point address
uint16_t epMaxPacketSize;	End point maximum payload
USB_DEVICE_AUDIO_IRP_OBJECT irpObject[USB_DEVICE_AUDIO_QUEUE_SIZE];	IRP object Q per end point

5.9.2.7.2.5 USB_DEVICE_AUDIO_EVENT Enumeration

C

```
typedef enum {
    USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE,
    USB_DEVICE_AUDIO_EVENT_READ_COMPLETE,
    USB_DEVICE_AUDIO_EVENT_INTERAFCE_ALTERNATE_SETTING,
    USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_RECEIVED,
    USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_SENT,
    USB_DEVICE_AUDIO_EVENT_ENTITY_SETTINGS_RECEIVED
} USB_DEVICE_AUDIO_EVENT;
```

Description

USB Device Audio Function Driver Control Transfer Status

This flag is used along with the [USB_DEVICE_AUDIO_ControlStatus\(\)](#) function to indicate success or failure of an Audio class specific control transfer request.

Members

Members	Description
USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE	This event occurs when a write operation scheduled by calling the USB_DEVICE_AUDIO_Write() function has completed. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE_DATA type.
USB_DEVICE_AUDIO_EVENT_READ_COMPLETE	This event occurs when a read operation scheduled by calling the USB_DEVICE_AUDIO_Read() function has completed. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_READ_COMPLETE_DATA type.
USB_DEVICE_AUDIO_EVENT_INTERAFCE_ALTERNATE_SETTING	USB spec allows Devices to have multiple alternate settings for the same interface. This event occurs when Host trying set an alternate setting for an interface present in this audio function. The callback function USB_DEVICE_AUDIO_EVENT_CALLBACK also sends Interface Number and Alternate setting number when this event occurs. An application need to take necessary action based on the interface alternate setting.
USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_RECEIVED	This event occurs when the data stage of a control write transfer has completed. This would occur after the application would respond with a USB_DEVICE_ControlReceive() function to a control write request from Host. This event notifies the application that the data is recived from Host and is available at the location passed by the USB_DEVICE_ControlReceive() function. The application should respond to Host with Zero Length Packet by calling USB_DEVICE_ControlStatus() function.

USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_SENT	This event occurs when the data stage of a control read transfer has completed. This event would occur after the application uses the USB_DEVICE_ControlSend() function to respond to an entity request recived from Host.

Remarks

None.

5.9.2.7.2.6 USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE Structure

C

```
typedef struct {
    USB_DEVICE_AUDIO_TRANSFER_HANDLE handle;
    uint16_t length;
} USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE;
```

Description

USB Device Audio Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_AUDIO_EVENT_READ_COMPLETE and USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE events.

Members

Members	Description

5.9.2.7.2.7 USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE Structure

C

```
typedef struct {
    USB_DEVICE_AUDIO_TRANSFER_HANDLE handle;
    uint16_t length;
} USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE;
```

Description

USB Device Audio Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_AUDIO_EVENT_READ_COMPLETE and USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE events.

Members

Members	Description

5.9.2.7.2.8 USB_DEVICE_AUDIO_EVENT_HANDLER Type

C

```
typedef USB_DEVICE_AUDIO_EVENT_RESPONSE (*
USB_DEVICE_AUDIO_EVENT_HANDLER)(USB_DEVICE_AUDIO_INDEX iAudio , USB_DEVICE_AUDIO_EVENT event ,
USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE controlTransferHandle, void * pData, uintptr_t
context);
```

Description

USB Device Audio Event Handler Function Pointer Type.

This data type defines the required function signature USB Device Audio Function Driver event handling callback function. The application must register a pointer to an Audio Function Driver events handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the Audio Function

Driver. The function driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

instanceIndex - Instance index of the Audio Function Driver that generated the event.

event - Type of event generated.

controlTransferHandle - Control Transfer Handle for Audio function driver events that require application response. The application should use this handle when the USB Audio Device Control Transfer functions to respond to the events.

pData - This parameter should be type casted to a event specific pointer type based on the event that has occurred. Refer to the [USB_DEVICE_AUDIO_EVENT](#) enumeration description for more details.

context - Value identifying the context of the application that registered the event handling function.

Remarks

None.

5.9.2.7.2.9 USB_DEVICE_AUDIO_EVENT_RESPONSE Type

C

```
typedef void USB_DEVICE_AUDIO_EVENT_RESPONSE;
```

Description

USB Device Audio Function Driver Event Handler Response Type

This is the return type of the Audio Function Driver event handler.

Remarks

None.

5.9.2.7.2.10 USB_DEVICE_AUDIO_INDEX Type

C

```
typedef uintptr_t USB_DEVICE_AUDIO_INDEX;
```

Description

USB Device Audio Function Driver Index

This uniquely identifies a Audio Function Driver instance.

Remarks

None.

5.9.2.7.2.11 USB_DEVICE_AUDIO_INSTANCE Structure

C

```
typedef struct {
    USB_DEVICE_HANDLE devLayerHandle;
    USB_DEVICE_AUDIO_INDEX audioIndex;
    USB_DEVICE_AUDIO_INTERFACE_COLLECTION infCollection;
    USB_DEVICE_AUDIO_INSTANCE_STATE state;
    USB_DEVICE_AUDIO_EVENT_HANDLER appEventCallback;
    uintptr_t userData;
} USB_DEVICE_AUDIO_INSTANCE;
```

Description

This is type USB_DEVICE_AUDIO_INSTANCE.

Members

Members	Description
USB_DEVICE_HANDLE devLayerHandle;	device layer instance associated with this function driver instance
USB_DEVICE_AUDIO_INDEX audiolIndex;	instance index
USB_DEVICE_AUDIO_INSTANCE_STATE state;	Current state of the function driver instance
USB_DEVICE_AUDIO_EVENT_HANDLER appEventCallBack;	Application callback
uintptr_t userData;	Application user data

5.9.2.7.2.12 USB_DEVICE_AUDIO_INSTANCE_STATE Enumeration**C**

```
typedef enum {
    USB_DEVICE_AUDIO_INSTANCE_NOT_INITIALIZED = 0,
    USB_DEVICE_AUDIO_INSTANCE_INITIALIZED,
    USB_DEVICE_AUDIO_INSTANCE_OPENED,
    USB_DEVICE_AUDIO_INSTANCE_CLOSED
} USB_DEVICE_AUDIO_INSTANCE_STATE;
```

Description

This is type USB_DEVICE_AUDIO_INSTANCE_STATE.

5.9.2.7.2.13 USB_DEVICE_AUDIO_INTERFACE_COLLECTION Structure**C**

```
typedef struct {
    uint8_t bControlInterfaceNum;
    uint8_t numStreamingInf;
    uint16_t bcdADC;
    USB_DEVICE_AUDIO_EP_INSTANCE intEp[1];
    bool isIntEpExists;
    USB_DEVICE_AUDIO_STREAMING_INTERFACE streamInf[USB_DEVICE_AUDIO_MAX_STREAMING_INTERFACES];
} USB_DEVICE_AUDIO_INTERFACE_COLLECTION;
```

Description

This is type USB_DEVICE_AUDIO_INTERFACE_COLLECTION.

Members

Members	Description
uint8_t bControlInterfaceNum;	control interface number
uint8_t numStreamingInf;	number of streaming interfaces
uint16_t bcdADC;	Audio spec in BCD 0x100 or 0x200
USB_DEVICE_AUDIO_EP_INSTANCE intEp[1];	optional interrupt ep info
bool isIntEpExists;	presence or absence of the interrupt EP

5.9.2.7.2.14 USB_DEVICE_AUDIO_INTERFACE_INFO Structure**C**

```
typedef struct {
    uint8_t interfaceNumber;
    uint8_t interfaceAlternateSetting;
} USB_DEVICE_AUDIO_INTERFACE_INFO;
```

Description

This is type USB_DEVICE_AUDIO_INTERFACE_INFO.

5.9.2.7.2.15 USB_DEVICE_AUDIO_INTERFACE_TYPE Enumeration

C

```
typedef enum {
    USB_DEVICE_AUDIO_INTERFACE_STREAM_UNDEFINED = 0,
    USB_DEVICE_AUDIO_INTERFACE_STREAM_AUDIO,
    USB_DEVICE_AUDIO_INTERFACE_STREAM_MIDI,
    USB_DEVICE_AUDIO_INTERFACE_FEEDBACK
} USB_DEVICE_AUDIO_INTERFACE_TYPE;
```

Description

This is type USB_DEVICE_AUDIO_INTERFACE_TYPE.

5.9.2.7.2.16 USB_DEVICE_AUDIO_IRP_OBJECT Structure

C

```
typedef struct {
    USB_DEVICE_IRP irp;
    USB_DEVICE_AUDIO_INDEX iAudio;
} USB_DEVICE_AUDIO_IRP_OBJECT;
```

Description

USB Device Audio Function Driver IRP object.
IRP object used by the Audio to service application requests.

Remarks

None.

5.9.2.7.2.17 USB_DEVICE_AUDIO_RESULT Enumeration

C

```
typedef enum {
    USB_DEVICE_AUDIO_RESULT_OK,
    USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_SIZE_INVALID,
    USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_QUEUE_FULL,
    USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID,
    USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_NOT_CONFIGURED,
    USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_HANDLER,
    USB_DEVICE_AUDIO_RESULT_ERROR_CONTROL_TRANSFER_FAILED,
    USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_INTERFACE_ID,
    USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_BUFFER
} USB_DEVICE_AUDIO_RESULT;
```

Description

USB Device Audio Function Driver USB Device Audio Result enumeration.
This enumeration lists the possible USB Device Audio Function Driver operation results. These values are returned by the [USB_DEVICE_AUDIO_Write\(\)](#) and [USB_DEVICE_AUDIO_Read\(\)](#) functions.

Members

Members	Description
USB_DEVICE_AUDIO_RESULT_OK	The operation was successful

USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_SIZE_INVALID	The transfer size is invalid. Refer to the description of the read or write function for more details
USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_QUEUE_FULL	The transfer queue is full and no new transfers can be scheduled
USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID	The specified instance is not provisioned in the system
USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_NOT_CONFIGURED	The specified instance is not configured yet
USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_HANDLER	The event handler provided is NULL
USB_DEVICE_AUDIO_RESULT_ERROR_CONTROL_TRANSFER_FAILED	The control transfer was aborted
USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_INTERFACE_ID	Interface number passed to the read or write function is invalid.

Remarks

None.

5.9.2.7.2.18 USB_DEVICE_AUDIO_STREAMING_INTERFACE Structure

C

```
typedef struct {
    uint8_t interfaceNum;
    USB_DEVICE_AUDIO_INTERFACE_TYPE infType;
    uint8_t activeSetting;
    USB_DEVICE_AUDIO_STREAMING_INTERFACE_ALTERNATE_SETTING
    alterntSetting[USB_DEVICE_AUDIO_MAX_ALTERNATE_SETTING];
} USB_DEVICE_AUDIO_STREAMING_INTERFACE;
```

Description

This is type USB_DEVICE_AUDIO_STREAMING_INTERFACE.

Members

Members	Description
uint8_t interfaceNum;	interface number
uint8_t activeSetting;	currently active alternate setting

5.9.2.7.2.19 USB_DEVICE_AUDIO_STREAMING_INTERFACE_ALTERNATE_SETTING Structure

C

```
typedef struct {
    uint8_t numEndPoints;
    USB_DEVICE_AUDIO_EP_INSTANCE isoDataEp;
    USB_DEVICE_AUDIO_EP_INSTANCE isoSyncEp;
} USB_DEVICE_AUDIO_STREAMING_INTERFACE_ALTERNATE_SETTING;
```

Description

This is type USB_DEVICE_AUDIO_STREAMING_INTERFACE_ALTERNATE_SETTING.

Members

Members	Description
uint8_t numEndPoints;	number of end-points in this interface
USB_DEVICE_AUDIO_EP_INSTANCE isoDataEp;	end points associated with this interface

5.9.2.7.2.20 USB_DEVICE_AUDIO_TRANSFER_HANDLE Type

C

```
typedef uintptr_t USB_DEVICE_AUDIO_TRANSFER_HANDLE;
```

Description

USB Device Audio Function Driver Transfer Handle Definition

This definition defines a USB Device Audio Function Driver Transfer Handle. A Transfer Handle is owned by the application but its value is modified by the [USB_DEVICE_AUDIO_Write\(\)](#) and [USB_DEVICE_AUDIO_Read\(\)](#) functions. The transfer handle is valid for the life time of the transfer and expires when the transfer related event had occurred.

Remarks

None.

5.9.2.8 Files

Files

Name	Description
usb_device_audio.h	USB Device AUDIO function Driver Interface

Description

5.9.2.8.1 usb_device_audio.h

USB DEVICE AUDIO Function Driver Interface

This file describes the USB Device AUDIO Function Driver interface.

Enumerations

	Name	Description
	USB_DEVICE_AUDIO_EVENT	USB Device Audio Function Driver Control Transfer Status
	USB_DEVICE_AUDIO_INSTANCE_STATE	This is type USB_DEVICE_AUDIO_INSTANCE_STATE.
	USB_DEVICE_AUDIO_INTERFACE_TYPE	This is type USB_DEVICE_AUDIO_INTERFACE_TYPE.
	USB_DEVICE_AUDIO_RESULT	USB Device Audio Function Driver USB Device Audio Result enumeration.

Functions

	Name	Description
	USB_DEVICE_AUDIO_ControlReceive	This function allows the application to respond to the Audio function driver specific control transfer requests. It allows the application to receive data from the host in the data stage of the control transfer.
	USB_DEVICE_AUDIO_ControlSend	This function allows the application to respond to the Audio function driver specific control transfer requests. It allows the application to send data to the host in the data stage of the control transfer.
	USB_DEVICE_AUDIO_ControlStatus	This function allows the application to complete the status stage of the the Audio Function Driver specific control transfer request.
	USB_DEVICE_AUDIO_EventHandlerSet	This function registers an event handler for the specified Audio function driver instance.

	USB_DEVICE_AUDIO_Read	This function requests a data read from the USB Device Audio Function Driver Layer.
	USB_DEVICE_AUDIO_Write	This function requests a data write to the USB Device Audio Function Driver Layer.

Macros

	Name	Description
	USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE	USB Device Audio Function Driver Event Handler Response Type None.
	USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID	USB Device Audio Function Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	USB_DEVICE_AUDIO_EP_INSTANCE	This is type USB_DEVICE_AUDIO_EP_INSTANCE .
	USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE	USB Device Audio Function Driver Read and Write Complete Event Data.
	USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE	USB Device Audio Function Driver Read and Write Complete Event Data.
	USB_DEVICE_AUDIO_INSTANCE	This is type USB_DEVICE_AUDIO_INSTANCE .
	USB_DEVICE_AUDIO_INTERFACE_COLLECTION	This is type USB_DEVICE_AUDIO_INTERFACE_COLLECTION .
	USB_DEVICE_AUDIO_INTERFACE_INFO	This is type USB_DEVICE_AUDIO_INTERFACE_INFO .
	USB_DEVICE_AUDIO_IRP_OBJECT	USB Device Audio Function Driver IRP object.
	USB_DEVICE_AUDIO_STREAMING_INTERFACE	This is type USB_DEVICE_AUDIO_STREAMING_INTERFACE .
	USB_DEVICE_AUDIO_STREAMING_INTERFACE_ALTERNATE_SETTING	This is type USB_DEVICE_AUDIO_STREAMING_INTERFACE_ALTERNATE_SETTING .

Types

	Name	Description
	USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE	USB Device Audio Function Driver Control Transfer Handle
	USB_DEVICE_AUDIO_EVENT_HANDLER	USB Device Audio Event Handler Function Pointer Type.
	USB_DEVICE_AUDIO_EVENT_RESPONSE	USB Device Audio Function Driver Event Callback Response Type
	USB_DEVICE_AUDIO_INDEX	USB Device Audio Function Driver Index
	USB_DEVICE_AUDIO_TRANSFER_HANDLE	USB Device Audio Function Driver Transfer Handle Definition.

File Name

usb_device_audio.h

Company

Microchip Technology Incorporated

5.9.3 USB Device Layer Library

5.9.3.1 Introduction

Introduces the MPLAB Harmony USB Device Layer

Description

The USB device layer library is part of the USB device stack that is available for the Microchip family of microcontrollers. This library has a dependency on the USB device driver to interact with the USB peripheral and therefore cannot operate independently. Within the USB device stack, the USB device layer is basically responsible for enumeration and performing control transfers. The USB device library implementation adheres to *USB Device Framework* of chapter-9 of USB specification 2.0.

In the USB device stack, the device layer features the following:

- Supports both USB Full-Speed and Hi-Speed operation
- Based on a modular and event driven architecture
- Supports the PIC32MX and PIC32MZ families of microcontrollers
- Supports composite USB devices
- Contains function drivers to implement the following type of device classes:
 - CDC-ACM
 - MSD
 - Generic
 - Audio
 - HID
- Supports non-blocking operation and is RTOS friendly
- Designed to integrate readily with other Harmony Middleware
- Supports both interrupt and polling operation

5.9.3.2 Release Notes

MPLAB Harmony Version: v0.70b **USB Device Layer Library Version :** 0.01 Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

- Limited USB chapter 9 testing performed on HID and MSD function drivers.
- The stack has not been tested for USB Interoperability.
- The USB Device Stack has not been tested with a Real Time Operating System.
- Attach/Detach behavior has been tested in a limited capacity.
- The device stack only supports interrupt mode operation. Polled mode operation will be added in a future release.
- While running the device stack on PIC32MZ microcontroller, the stack requires 3 seconds to initialize. This is due to a hardware issue with PIC32MZ USB module.
- Device stack operation has been tested with Windows XP and Windows 7 OS based PC USB Host.

5.9.3.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

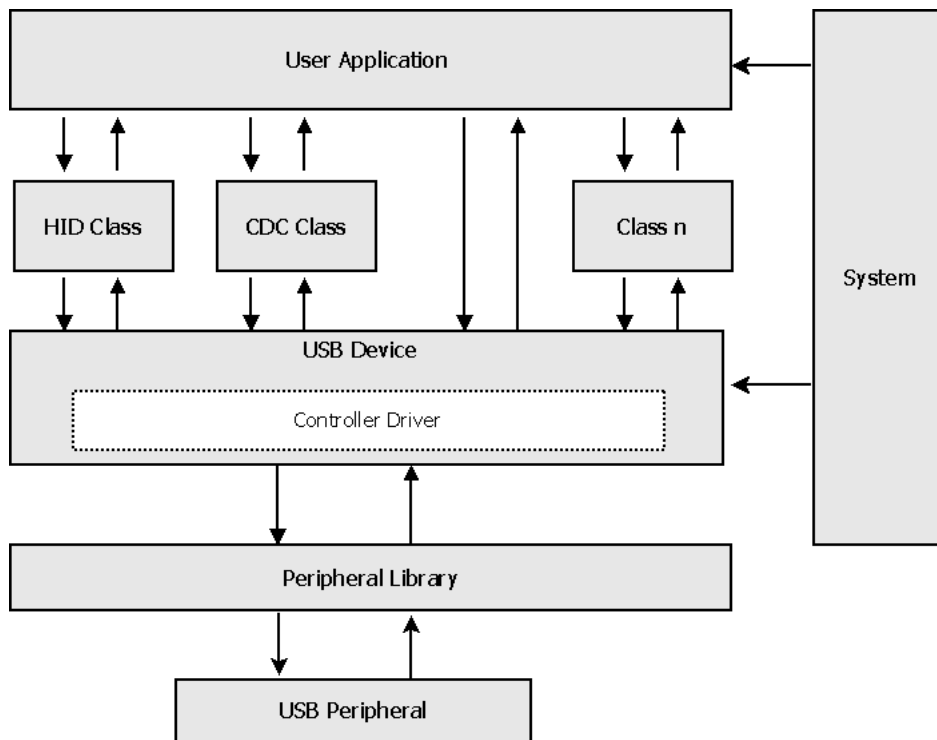
5.9.3.4 Using the Library

5.9.3.4.1 Abstraction Model

Describes the Abstraction Model of the USB Device Layer.

Description

The block diagram shows USB device layer interaction with USB controller driver, function drivers, user application and the system.



USB Device Layer Software Abstraction Block Diagram

System Interaction

The system is responsible for initializing and de-initializing the device layer. It is also responsible for calling the USB device layer task routine.

Function driver interaction

The USB device layer interacts with a function driver for following reasons.

- Device layer initializes the function driver when device is configured by the host. This can happen when host issues a set configuration request to the device. The device layer initializes only those function drivers that are valid for the selected configuration.
- Device layer de-initializes the function driver when host issues a bus reset or when device is detached from the host or when host unconfigures the device by setting configuration value to 0.
- The function driver task routines are run by the device layer. This means that function driver tasks runs at the same priority as device layer task.
- The device layer forwards class/interface specific setup requests from host to function drivers for processing. The function drivers can use device layer APIs to read data stage from endpoint 0 or write data stage to endpoint 0.

All of the above interactions are initiated by the device layer and hence it is required for a function driver to register a set of standard APIs with the device layer for initializing/de-initializing the function driver, for handling control transfers and for running the task routines. Registering of these callback functions with the device layer is compile time option and is done using function driver registration table. Function driver registration is explained in the subsequent sections of this help file.

User application (client) interaction

User application clients can register a callback function with the device layer to get USB device events. Apart from device events, the clients can interact with USB device layer to know other status like USB speed and remote wakeup status.

5.9.3.4.2 Library Overview

Provides an overview of the USB Device Layer Library

Description

The USB device layer mainly interacts with system, its clients and function drivers as shown in [USB Device Layer Software Abstraction Block Diagram](#). Hence the interfaces provided by the USB device layer can be broadly be classified as shown in the below table. Please refer to section "Library Interface" for more information on APIs, data types and constants.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, de-initialization, re-initialization and task functions
Client Core Functions	Provides function to register callbacks, mechanism to pass events to clients and functions to know the status.
Driver Interaction Functions	Provides function driver level interfaces for handling event callbacks and performing control transfers

5.9.3.5 How the Library Works

5.9.3.5.1 Files to Include

Describes which files to include in project while using the USB Device Layer

Description

Table 1 shows the files that must be included in the project in order to use the USB Device Layer. These files are located in the framework folder of the MPLAB Harmony installation. It is assumed that the USB Driver files are already included in the project.

Filename	Description
\\framework\\usb\\src\\dynamic\\usb_device.c	Contains the USB Device Layer Implementation.
\\framework\\usb\\ usb_device.h	Must included in every source file that needs to invoke USB Device Layer API.
system_config.h	User created file which contains the USB Device Layer Configuration MACors.

Table 1: Files to be included in USB Device Project

5.9.3.5.2 Initializing the Library

Describes how the USB Device Layer must be initialized

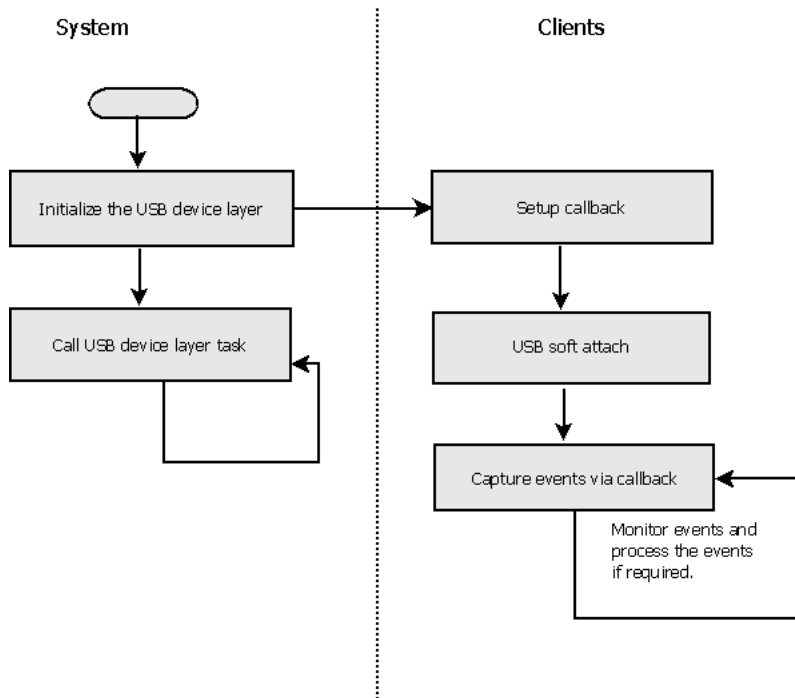
Description

Following are the components that a user must initialize and register with the USB device stack for the proper operation of the stack.

- Descriptors
- Master descriptor table
- Function registration table

The subsequent sections of this help file explains how these components must be initialized using code examples.

The flow chart shows the steps that has to be followed by the system and clients to initialize the USB device layer. The system initializes the USB device layer. After system initializes the device layer, clients can open handle to device layer, setup callback into the device layer and start capturing the events from the device layer.



Flowchart showing the sequence for initializing the USB device layer

The following code example shows the system side initialization sequence.

Example:

```

SYS_MODULE_OBJ usbDeviceObj;
DRV_HANDLE usbDevHandle;

// System module initialization
deviceLayerInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
// Identifies peripheral (PLIB-level) ID
deviceLayerInit.usbID = USB_ID_1;
// Boolean flag: true -> Stop USB module in Idle Mode
deviceLayerInit.stopInIdle = false;
// Boolean flag: true -> Suspend USB in Sleep Mode
deviceLayerInit.suspendInSleep = false;
// Interrupt Source for USB module
deviceLayerInit.interruptSource = INT_SOURCE_USB_1;
// Number of function drivers registered to this instance of the
USB device layer
deviceLayerInit.registeredFuncCount = 1;
// Function driver table registered to this instance of the USB device layer
deviceLayerInit.registeredFunctions = funcRegistrationTable;
// Pointer to USB Descriptor structure
deviceLayerInit.usbMasterDescriptor = &usbMasterDescriptor;
// USB operation speed.
deviceLayerInit.deviceSpeed = USB_SPEED_HIGH;

// USB device initialization
usbDeviceObj = USB_DEVICE_Initialize(USB_DEVICE_INDEX_0, &deviceLayerInit);
  
```



```

if (SYS_MODULE_OBJ_INVALID == usbDriverObj)
{
    // Handle error
}

while(1)
{
    // Call device layer task
    USB_DEVICE_Tasks( usbDeviceObj );
}

```

The following code example shows how a client can open an handle to device layer and use the same.

Example:

```

void clientInitialize( void )
{
    DRV_HANDLE usbDevHandle;

    // Open the device layer.
    usbDevHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0 );

    if(DRV_HANDLE_INVALID == usbDevHandle)
    {
        // Handle error
    }

    // Register a callback with device layer to get event notification (for end point 0)
    USB_DEVICE_EventCallbackSet(usbDevHandle, usbDeviceEventCallback); // Where
usbDeviceEventCallback is a callback function                                // that must be
provided by the client
}

void usbDeviceEventCallback( USB_DEVICE_EVENTS events, USB_DEVICE_EVENT_DATA * eventData )
{
    // Handle all device layer events here.
    switch( event )
    {
        case USB_DEVICE_EVENT_RESET:
        case USB_DEVICE_EVENT_DECONFIGURED:
            /* Bus reset is detected or device is unconfigured */
            /* TODO: Add user code here. */

            break;

        case USB_DEVICE_EVENT_CONFIGURED:
            if( eventData->eventConfigured.configurationValue == 1 )
            {
                /* The device is in configured state */
                /* TODO: Add user code here. */
            }
            break;

        case USB_DEVICE_EVENT_SUSPENDED:
            /* Device is suspended */
            /* TODO: Add user code here. */

            break;

        case USB_DEVICE_EVENT_RESUMED:
        case USB_DEVICE_EVENT_ATTACHED:
        case USB_DEVICE_EVENT_DETACHED:
        case USB_DEVICE_EVENT_ERROR:
        default:
            break;
    }
}

```

```
    }
}
```

5.9.3.5.2.1 Descriptors

A USB device reports its attributes to host using descriptors that are defined in chapter 9 of USB Specification, Revision 2.0. The user must initialize device descriptor, configurations descriptor, string descriptors and device qualifier descriptor using static tables. If the device supports dual speeds then there can be two sets of these descriptors, one for full speed and another for high speed. These descriptors are then registered to the USB device stack using master descriptor table.

The following code example shows how the descriptors are initialized in `system_config.c` file using static tables. Refer to Master Descriptor Table to know how to register these descriptors into USB device stack.

```
/* Example USB descriptors for a MSD device */

/* Device Descriptor Table */
const USB_DEVICE_DESCRIPTOR deviceDescriptor =
{
    0x12, // Size of this descriptor in bytes
    USB_DESCRIPTOR_DEVICE, // DEVICE descriptor type
    0x0110, // USB Spec Release Number in BCD format
    0x00, // Class Code
    0x00, // Subclass code
    0x00, // Protocol code
    USB_DEVICE_EP0_BUFFER_SIZE, // Max packet size for EP0, see usbcfg.h
    0x04D8, // Vendor ID
    0x0009, // Product ID: mass storage device demo
    0x0001, // Device release number in BCD format
    0x01, // Manufacturer string index
    0x02, // Product string index
    0x03, // Device serial number string index
    0x01 // Number of possible configurations
};

/* Configurations Descriptor Table */
const uint8_t configDescriptor[] =
{
    /* Configuration Descriptor */
    9, // Size of this descriptor in bytes
    USB_DESCRIPTOR_CONFIGURATION, // CONFIGURATION descriptor type
    0x20, 0x00, // Total length of data for this cfg
    1, // Number of interfaces in this cfg
    1, // Index value of this configuration
    0, // Configuration string index
    USB_ATTRIBUTE_DEFAULT | USB_ATTRIBUTE_SELF_POWERED, // Attributes, see usbdefs_std_dsc.h
    50, // Max power consumption (2X mA)

    /* Interface Descriptor */
    9, // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE, // INTERFACE descriptor type
    0, // Interface Number
    0, // Alternate Setting Number
    2, // Number of endpoints in this intf
    USB_DEVICE_MSD_INTF, // Class code
    USB_DEVICE_MSD_INTF_SUBCLASS, // Subclass code
    USB_DEVICE_MSD_PROTOCOL, // Protocol code
    0, // Interface string index

    /* Endpoint Descriptor */
    7,
    USB_DESCRIPTOR_ENDPOINT,
    0x01 | USB_EP_DIRECTION_IN,
    USB_TRANSFER_TYPE_BULK,
    MSD_IN_EP_SIZE,
}
```

```

    0x00,
    0x00,

    7,
    USB_DESCRIPTOR_ENDPOINT,
    0x01 | USB_EP_DIRECTION_OUT,
    USB_TRANSFER_TYPE_BULK,
    MSD_OUT_EP_SIZE,
    0x00,
    0x00
};

/* String descriptors Table */

/* Language code string descriptor */
const struct
{
    uint8_t bLength;
    uint8_t bDscType;
    uint16_t string[1];
} sd000 =
{
    sizeof(sd000),
    USB_DESCRIPTOR_STRING,
    { 0x0409 }
};

/* Manufacturer string descriptor */
const struct
{
    const uint8_t bLength;
    uint8_t bDscType;
    uint16_t string[25];
} sd001 =
{
    sizeof(sd001),
    USB_DESCRIPTOR_STRING,
    {
        'M', 'i', 'c', 'r', 'o', 'c', 'h', 'i', 'p', ' ',
        'T', 'e', 'c', 'h', 'n', 'o', 'l', 'o', 'g', 'y', ' ', 'I', 'n', 'c', '.'
    }
};

/* Product string descriptor */
const struct
{
    const uint8_t bLength;
    uint8_t bDscType;
    uint16_t string[22];
} sd002 =
{
    sizeof(sd002),
    USB_DESCRIPTOR_STRING,
    {
        'S', 'i', 'm', 'p', 'l', 'e', ' ', 'M', 'S', 'D', ' ',
        'D', 'e', 'v', 'i', 'c', 'e', ' ', 'D', 'e', 'm', 'o'
    }
};

/* Serial number string descriptor. Note: This should be unique for each unit
built on the assembly line. Plugging in two units simultaneously with the
same serial number into a single machine can cause problems. Additionally, not
all hosts support all character values in the serial number string. The MSD
Bulk Only Transport (BOT) specs v1.0 restrict the serial number to consist only
of ASCII characters "0" through "9" and capital letters "A" through "F". */

```

```

const struct
{
    uint8_t bLength;
    uint8_t bDscType;
    uint16_t string[12];
} sd003 =
{
    sizeof(sd003), USB_DESCRIPTOR_STRING,
    {'1','2','3','4','5','6','7','8','9','9','9','9'}
};

```

5.9.3.5.2.2 Master Descriptor Table

Master Descriptor Table is a static table that holds collection of pointers that points to USB descriptors. User will have to initialize the master descriptor table and must register it with the USB device stack. During the device enumeration the stack uses these information from the Master Descriptor Table to respond to the setup requests from the host. Following code example shows how to set up the Master Descriptor Table. The Master Descriptor Table must then be registered with the USB device stack while initializing the USB device layer. Refer to Initializing the Data Structure to know how it can be registered with the USB device stack.

```

/* Array of string descriptors */
/* Refer to Descriptors section to see how the
   string descriptors sd000 to sd003 are defined */

const uint8_t *const USB_SD_Ptr[4]=
{
    (const uint8_t *const)&sd000,
    (const uint8_t *const)&sd001,
    (const uint8_t *const)&sd002,
    (const uint8_t *const)&sd003
};

/* Array of full speed config descriptors */
/* Refer to Descriptors section to see how the
   configurations descriptor configDescriptor[] is defined */

const uint8_t *const fullSpeedConfigDescSet[1] =
{
    (const uint8_t *const)&configDescriptor[0]
};

/* Initialize the master descriptor table */

const USB_MASTER_DESCRIPTOR usbMasterDescriptor =
{
    /* Low/Full speed device descriptor */
    /* Refer to Descriptors section to see how deviceDescriptor
       is initialized */
    (uint8_t *)&deviceDescriptor ,

    /* Total number of low/full speed configurations available */
    sizeof( fullSpeedConfigDescSet )/sizeof( uint8_t* ) ,

    /* Pointer to array of low/full speed configurations descriptors */
    ( USB_DEVICE_CONFIG_DESCS_PTR )&fullSpeedConfigDescSet[0] ,

    /* High speed device descriptor. Not supported in this example
       and is set to NULL.*/
    NULL,

    /* Total number of high speed configurations available.

```

```

    Zero for this example */
0,

/* Pointer to array of high speed configurations descriptors.
   Not supported in this example*/
NULL,

/* Total number of string descriptors available */
sizeof( USB_SD_Ptr )/sizeof( uint8_t* ),

/* Pointer to array of string descriptors */
(USB_DEVICE_STRING_DESCS_PTR)USB_SD_Ptr,

/* Pointer to full speed dev qualifier. Not supported in this
   example */
NULL,

/* Pointer to high speed dev qualifier. Not supported in this
   example*/
NULL,
};

```

5.9.3.5.2.3 Functions Registration Table

This section explains how function drivers can be registered with the USB device layer using Function Registration Table. Function Registration Table is a static table and must be configured at compile time. It contains mapping of function driver instance to USB speed and configuration. This table helps the USB device layer to initialize the appropriate function driver for a configuration value and device speed selected by the host.

Following code example shows how to build a function registration table.

```

/* This is a code example that registers MSD function
   driver to the USB device stack*/

#define MSD_INDEX 0

const USB_DEVICE_FUNC_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
        /* Speed: Indicating both full/high speeds here
           will make the device stack to load MSD function driver
           for both the speeds */
        USB_SPEED_FULL | USB_SPEED_HIGH,

        /* Configuration value */
        1,

        /* InterfaceNumber */
        0,

        /* Total number of interfaces in this driver */
        1,

        /* Instance Index of the MSD function driver that will be loaded
           for this configuration value and speed */
        MSD_INDEX,

        /* MSD initialization data structure. See MSD documentation for this structure */
        (void*)&msdInit,

        /* Pointer to the structure that contains collection of MSD driver function pointers.
           The USB device stack calls out these MSD functions appropriately */
        (USB_DEVICE_FUNCTION_DRIVER*) &msdFunctionDriver
    }
}

```

```
};
```

5.9.3.5.2.4 Initializing the Data Structure

This section explains about the USB device initialization data structure. Before initializing this data structure user must keep the descriptors table, master descriptor table and function registration table ready and initialized. This structure maps these static tables to an instance of USB device layer and provides a mechanism to register these static tables with the USB device layer. Pointer to this initialization structure is then passed to USB device layer using [USB_DEVICE_Initialize\(\)](#) function. Following code example shows how to initialize this structure and register the same with the USB device stack.

```
/* USB Device initialization data structure */

const USB_DEVICE_INIT usbDevInitData =
{
    /* System module initialization */
    {SYS_MODULE_POWER_RUN_FULL},

    /* Identifies peripheral (PLIB-level) ID */
    USB_ID_1,

    /* Stop in idle */
    false,

    /* Stop in sleep */
    false,

    /* Interrupt source */
    INT_SOURCE_USB_1,

    /* Number of functions registered to this instance of the
    USB device layer. See Function Registration Table section of this help file.
    We are registering on */
    (sizeof(funcRegistrationTable)/sizeof(USB_DEVICE_FUNC_REGISTRATION_TABLE)),

    /* Function driver table registered to this instance of the USB device layer*/
    (USB_DEVICE_FUNC_REGISTRATION_TABLE*)funcRegistrationTable,

    /* Pointer to Master Descriptor structure.
    See Master Descriptor Table section of this help file. */
    (USB_MASTER_DESCRIPTOR*)&usbMasterDescriptor
};
/* This is an example that shows how the device initialization structure
is registered with USB_DEVICE_Initialize() */

typedef struct
{
    /* device layer object returned by device layer init function */
    SYS_MODULE_OBJ      usbDevObject;

    /* Controller driver object returned by controller driver init function */
    SYS_MODULE_OBJ      usbCDObject;
} APP_DRV_OBJECTS;

typedef struct
{
    /* device layer handle returned by device layer open function */
    DRV_HANDLE          usbDevHandle;
} APP_DATA;

/* Application objects */
APP_DRV_OBJECTS appDrvObject;
```

```

/* Application data */
APP_DATA appData =
{
    /* device layer handle */
    DRV_HANDLE_INVALID
};

void SYS_Initialize ( void )
{
    /* Set up cache and wait states for
    * maximum performance. */
    SYSTEMConfigPerformance(80000000);

    /* Initialize the BSP */
    BSP_Initialize( );

    /* Initialize the USB device layer.
    Note the registration of initialization data structure with the USB
    device layer */
    appDrvObject.usbDevObject = USB_DEVICE_Initialize (USB_DEVICE_INDEX_0 ,
        ( SYS_MODULE_INIT* ) & usbDevInitData);

    /* check if the object returned by the device layer is valid */
    SYS_ASSERT((SYS_MODULE_OBJ_INVALID != appDrvObject.usbDevObject),
        "Invalid USB DEVICE object");

    /* open an instance of the device layer */
    appData.usbDevHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0, 0 );

    /* Register a callback with device layer to get event notification
    (for end point 0) */
    USB_DEVICE_EventCallBackSet(appData.usbDevHandle,
        APP_UsbDeviceEventCallBack);

    /* Enable USB device layer */
    USB_DEVICE_Attach(appData.usbDevHandle);

    /* Initialize the Application */
    APP_Initialize ( );

    /* Initialize the interrupt system */
    SYS_INT_Initialize();

    /* set priority for USB interrupt source */
    SYS_INT_VectorPrioritySet(INT_VECTOR_USB, INT_PRIORITY_LEVEL3);

    /* set sub-priority for USB interrupt source */
    SYS_INT_VectorSubprioritySet(INT_VECTOR_USB, INT_SUBPRIORITY_LEVEL3);

    /* Initialize the global interrupts */
    SYS_INT_Enable();
}

```

5.9.3.5.3 Device Events

Describes the USB Device Layer Events

Description

The device layer generates following events. The application clients can capture these events by registering a callback function into the device layer using [USB_DEVICE_EventCallBackSet\(\)](#) function.

Events	Description
USB_DEVICE_EVENT_RESET	USB bus reset occurred. This event is an indication to the application client that device layer has de-initialized the function drivers. For application client, this event means that USB device has moved to default state and any further communication with the function drivers must be stopped.
USB_DEVICE_EVENT_SUSPENDED	This event is an indication to the application client that device is suspended and it can put the device to power-down mode if required.
USB_DEVICE_EVENT_RESUMED	Event that indicates that device has resumed from suspended state.
USB_DEVICE_EVENT_ERROR	This event is an indication to the application client that an error occurred on the USB bus.
USB_DEVICE_EVENT_SOF	This event is generated for every new start of frame. Application client can use this SOF event for general time based house keeping activities.
USB_DEVICE_EVENT_DETACHED	This event is an indication to the application client that the device is detached from the host and device layer is about to de-initialize the function drivers. For application client, this event means that it has to stop any further interactions with the function driver.
USB_DEVICE_EVENT_DECONFIGURED	This event is generated when the USB host unconfigures the device by configuring the device to configuration value 0.
USB_DEVICE_EVENT_CONFIGURED	This event is generated when the device is configured by the host. The device layer loads all the function drivers applicable to that configuration and application client can interact with function drivers.
USB_DEVICE_EVENT_ATTACHED	Device is attached to the USB, but is not powered.

The following code example shows how the application clients can register callback function with the to capture these events

Example:

```

void SYS_Initialize( void )
{
    /* Add user initialization code here */

    /* Register a callback with device layer to get event notification (for end point 0) */
    USB_DEVICE_EventCallBackSet(appData.usbDevHandle, APP_UsbDeviceEventCallBack);

    /* Enable the device */
    USB_DEVICE_Attach(appData.usbDevHandle);
}

void APP_UsbDeviceEventCallBack( USB_DEVICE_EVENT event, USB_DEVICE_EVENT_DATA * pEventData )
{
    switch( event )
    {
        case USB_DEVICE_EVENT_RESET:
        case USB_DEVICE_EVENT_DECONFIGURED:
            /* USB device is reset or device is deconfigured. */
            /* Add user code here */

            break;

        case USB_DEVICE_EVENT_CONFIGURED:
            /* Device is configured */
            /* Check the configuration value */

            if(pEventData->eventConfigured.configurationValue == 1)

```



```

    {
        /* Host selected configuration value 1 */
    }
    break;

case USB_DEVICE_EVENT_SUSPENDED:
    /* Add user's code here to handle USB bus suspend event */

    break;

case USB_DEVICE_EVENT_RESUMED:
case USB_DEVICE_EVENT_ERROR:
case USB_DEVICE_EVENT_SOF:
case USB_DEVICE_EVENT_DETACHED:
case USB_DEVICE_EVENT_ATTACHED:
default:
    break;
}
}

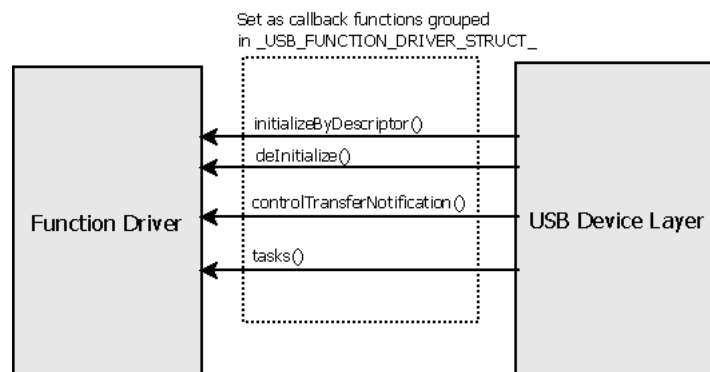
```

5.9.3.5.4 Developing Vendor-specific Function Drivers

This topic describes the development of vendor-specific function drivers.

Description

This section is important for the users who wish to develop their own function drivers. This section covers the interface between the function driver and USB device layer. The framework explains how the USB device layer manages function driver through a set of callbacks.



Framework showing the interaction between Function Driver and Device Layer

The USB device layer manages the function driver using a set of standard callout functions. A function driver must implement these callout functions as described in the following paragraphs. The pointer to these callout functions must be packed in [USB_DEVICE_FUNCTION_DRIVER](#) structure and pointer to this structure must be registered with the USB device layer using function registration table.

InitializeByDescriptor()

Device layer initializes a function driver by calling this function when host issues a set configuration request. This function is called multiple times by the device layer for every descriptor that is found under the interface owned by the function driver. The function driver must implement appropriate logic to configure itself for each descriptor. This function is called from within the interrupt context. Therefore the function driver design must ensure that the execution time of this function is short.

The example code for `initializeByDescriptor()` is shown below.

```

/*****

```

Function:

```

void USB_DEVICE_MYCLASS_InitializeByDescriptorType ( SYS_MODULE_INDEX iFunctionDriver,
                                                    SYS_MODULE_INDEX iDriver,
                                                    SYS_MODULE_INDEX iUsbDevice,
                                                    void* funcDriverInit ,
                                                    uint8_t* pConfigDesc )

```

Summary:

USB Device MYCLASS function driver initialization. The device layer calls this function to initialize the function driver based on the descriptor found (from the configurations descriptor).

Precondition:

None.

Parameters:

iFunctionDriver : Function driver instance index.
usbDeviceHandle : USB device handle for communicating with the USB device layer.
funcDriverInit : Pointer to function driver init data structure. User can have his own function driver init data structure and can register the same in the function registration table.
intfNumber : The interface number the descriptor identified by descriptorType belongs to.
altSetting : The alternate setting of the descriptor that is identified by descriptorType belongs to.
pDescriptor : Pointer to the descriptor. This can point to interface descriptor or endpoint descriptor or any class specific descriptor based on the descriptorType.

Returns:

None.

Remarks:

None

```

*/

```

```

void USB_DEVICE_MYCLASS_InitializeByDescriptorType(SYS_MODULE_INDEX iFunctionDriver,
DRV_HANDLE usbDeviceHandle,
                                                    void* funcDriverInit, uint8_t intfNumber, uint8_t
altSetting,
                                                    uint8_t descriptorType, uint8_t * pDescriptor)
{
    USB_ENDPOINT endpoint;
    USB_ENDPOINT_DESCRIPTOR endpointDescriptor

    switch(descriptorType )
    {
        case USB_DESCRIPTOR_ENDPOINT:
            // Get the endpoint
            endpointDescriptor = (USB_ENDPOINT_DESCRIPTOR *)pDescriptor;

            // Enable endpoint.
            USB_DEVICE_EndpointEnable(usbDeviceHandle, endpointDescriptor->endpoint,
                                     endpointDescriptor->transferType,
                                     endpointDescriptor->wMaxPacketSize);

            break;

        case USB_DESCRIPTOR_INTERFACE:
            // Do interface specific initialization here.

```

```

        break;

    default:
        // Handle class specific descriptor.
        break;
}

```

DeInitialize()

Device layer de-initializes a function driver by calling this callout function when host issues a USB bus reset or when host deconfigures the device by selecting configuration value as 0. In the DeInitialize() function, user must implement functionalities to close endpoints and cancel the IRPs owned by that particular function driver. This callout function is called from within the interrupt context. Therefore the design must ensure that the execution time of this callout function is short.

The example code for DeInitialize() is shown below.

```

/*****
Function:
    void USB_DEVICE_MYCLASS_DeInitialize( SYS_MODULE_INDEX iFunctionDriver )

Summary:
    Deinitializes an instance of the MYCLASS.

Description:

Parameters:
    iFunctionDriver          - function driver index

Returns:
    None.

*/

static void USB_DEVICE_MYCLASS_DeInitialize( SYS_MODULE_INDEX iFunctionDriver )
{
    /* Cancel all IRPs on the endpoint/s used by
       MYCLASS function driver */
    USB_DEVICE_IRPCancelAll( usbDeviceHandle, endpoint );

    /* Disable endpoints used by this function driver */
    USB_DEVICE_EndpointDisable( usbDeviceHandle, endpoint );
}

```

ControlTransferNotification()

Only standard setup packets whose recipient is device/endpoint is supported by the device layer. All other types of setup packet targeted to interface/class is forwarded to appropriate function driver using "controlTransferNotification()" callout function. Setup packets with "other" as recipient is discarded by the device layer and a STALL is generated on the control endpoint. In this callout function, the function driver has to parse the setup packet. If the request is supported, function driver can initiate data stage by using [USB_DEVICE_ControlSend](#)/[USB_DEVICE_ControlReceive](#) functions. If the request is not supported, function driver can stall the control endpoints by using [USB_DEVICE_ControlStatus](#) function. This callout function is called from within the interrupt context. Therefore the design must ensure that the execution time of this callout function is short. This is not a mandatory callout function and can be set to NULL if a function driver does not support class specific setup request.

```

/*****
Function:
    void USB_DEVICE_MYCLASS_ControlTransferHandler(
        USB_DEVICE_CONTROL_TRANSFER_HANDLE controlHandle,
        SYS_MODULE_INDEX iFunctionDriver,
        USB_DEVICE_CONTROL_TRANSFER_EVENT controlEvent,

```

```
void * pEventData )
```

Summary:

Handles all MYCLASS related control transfers.

Description:

Parameters:

*controlHandle - Control Handle
iFunctionDriver - Function driver index
controlEvent - Control transfer event
pEventData - Pointer to event related data*

Returns:

None.

```
*/
```

```
void USB_DEVICE_MYCLASS_ControlTransferHandler(
    USB_DEVICE_CONTROL_TRANSFER_HANDLE controlHandle,
    SYS_MODULE_INDEX iFunctionDriver,
    USB_DEVICE_CONTROL_TRANSFER_EVENT controlEvent,
    USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA * pEventData )
{
    USB_DEVICE_HID_INSTANCE *hidThisInstance;
    SETUP_PKT * setupPkt = pEventData->setupRequest;
    USB_DEVICE_HID_EVENT_DATA eventData;

    hidThisInstance = &gUsbDeviceHidInstance[iHID] ;

    if( controlEvent == USB_DEVICE_CONTROL_SETUP_REQUEST )
    {
        /* Setup request is received */

        if( setupPkt->bmRequestType & SETUP_RECIPIENT_INTERFACE)
        {
            /* Handle interface specific requests here */
            /* Use USB_DEVICE_ControlSend, USB_DEVICE_ControlReceive or
               USB_DEVICE_ControlStatus functions appropriately to respond to
               setup packet */

        }
        else if( ( setupPkt->bmRequestType & SETUP_TYPE_CLASS ) &&
                 ( hidThisInstance->appCallBack != NULL ) )
        {
            /* Handle class specific requests here */
            /* Use USB_DEVICE_ControlSend, USB_DEVICE_ControlReceive or
               USB_DEVICE_ControlStatus functions appropriately to respond to
               setup packet */

        }
    }
    else if ( controlEvent == USB_DEVICE_CONTROL_DATA_RECEIVED )
    {
        /* Data stage is received. Handle the data */
        /* Use USB_DEVICE_ControlStatus functions appropriately to respond to
           data stage */

    }
    else if ( controlEvent == USB_DEVICE_CONTROL_DATA_SENT )
    {
        /* Data stage is sent */
        /* This is just a success event. If required inform the same to
           application clients here. */

    }
}
```

Tasks()

All function driver tasks are called from within the device layer tasks. Device layer can call function driver task routine using this function driver provided callout function. Within this callout function, a function driver can implement its task specific routines. This is not a mandatory callback function. Set this callout function to NULL if it is not supported by the function driver.

```

/*****
Function:
    void USB_DEVICE_MYCLASS_Tasks ( SYS_MODULE_INDEX iFunctionDriver )

Summary:
    This function handles the main MSD state machine.

Description:

Precondition:
    None.

Parameters:
    iFunctionDriver    : Function driver index.

Returns:
    None.

Remarks:
    None
*/
void USB_DEVICE_MYCLASS_Tasks ( SYS_MODULE_INDEX iFunctionDriver )
{
    /* Perform all function driver tasks here */
}

```

Registering the callout functions

To register the above callout functions with the device layer, create a static table of type [USB_DEVICE_FUNCTION_DRIVER](#) structure with pointers to callout functions. The structure must then be registered to USB device layer using function registration table. Here is the code snippet that shows the same.

```

/* Make a table of USB_DEVICE_FUNCTION_DRIVER with pointers to callout functions */
const USB_DEVICE_FUNCTION_DRIVER myclassDriver = {
    /* MYCLASS init function */
    .initializeByDescriptor = USB_DEVICE_MYCLASS_InitializeByDescriptorType ,

    /* MYCLASS de-init function */
    .deInitialize = USB_DEVICE_MYCLASS_Deinitialization ,

    /* MYCLASS set-up packet handler */
    .controlTransferNotification = USB_DEVICE_MYCLASS_ControlTransferHandler ,

    /* MYCLASS tasks function */
    .tasks = USB_DEVICE_MYCLASS_Tasks
};

/* Using the function registration table register MYCLASS driver with USB device stack */
const USB_DEVICE_FUNC_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
        /* Speed
        USB_SPEED_FULL | USB_SPEED_HIGH,

        /* Configuration value
        1,

        /* interface number of MYCLASS

```

```

    0,

    // number of interfaces in MYCLASS
    1,

    // instance index of MYCLASS
    0,

    // MYCLASS data init structure. This structure is class specific data structure
    // that will be passed as funcDriverInit parameter to
    // USB_DEVICE_MYCLASS_InitializeByDescriptorType()
    &myclassInit,

    // My class driver structure having pointers to all the callouts
    &myclassDriver
};
}

```

5.9.3.5.5 Library Configuration

Describes how to configure the USB Device Library

Macros

	Name	Description
	USB_DEVICE_MAX_CLIENTS	Sets the maximum possible number of clients an instance of the USB device can open using USB_DEVICE_Open .
	USB_DEVICE_MAX_FUNCTION_DRIVER	Sets the maximum number of function drivers at a time that are supported by an instance of the USB device layer.
	USB_DEVICE_MAX_INSTANCES	Sets the maximum possible number of instances of the USB device that can be instantiated by using USB_DEVICE_Initialize() routine.

Description

The application designer must specify the following configuration parameters while using the USB Device Library. The configuration macros that implement these parameters must be located in the system_config.h file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

Configuration Macro Name	Description	Comments
USB_DEVICE_MAX_INSTANCES	Sets the maximum possible number of instances of the USB device that can be instantiated by using USB_DEVICE_Initialize() routine.	In case of microcontrollers with more than one USB peripheral, the value of this constant can be increased to support more than one instances of USB device layer. The static implementation supports only one instance. Setting the value of this constant to > 1 has no effect on static implementations. Only in dynamic implementations of the USB device layer this value can be set > 1. The USB device layer has to support at least one instance. Therefore, ensure that the value of this constant is set to > 0. Increasing the instance count consumes RAM and can lead to performance degradation.
USB_DEVICE_MAX_CLIENTS	Sets the maximum possible number of clients an instance of the USB device can open using USB_DEVICE_Open .	If multiple clients need USB device layer services, user can set the value of this constant to > 1. The value of this macro must not be set to zero. Each instance of the USB device layer must support at least one client. Therefore, set the value to at least 1. The static single client implementation of the USB device layer supports only one client. Therefore, increasing the value of this constant > 1 has no effect in static single client implementation. Increasing the client count consumes RAM and can lead to performance degradation.

5.9.3.5.5.1 USB_DEVICE_MAX_CLIENTS Macro

C

```
#define USB_DEVICE_MAX_CLIENTS 1
```

Description

If multiple clients need USB device layer services, user can set the value of this constant to > 1.

Remarks

The value of this macro must not be set to zero. Each instance of the USB device layer must support at least one client. Therefore, set the value to at least 1.

The static single client implementaion of the USB device layer supports only one client. Therefore, increasing the value of this constant > 1 has no effect in static single client implementation.

Increase the client count consumes RAM and can lead to performance degradation.

5.9.3.5.5.2 USB_DEVICE_MAX_FUNCTION_DRIVER Macro

C

```
#define USB_DEVICE_MAX_FUNCTION_DRIVER 3
```

Description

This constant sets the maximum number of function drivers that are loaded by a USB device instance for a configuration set by the USB host.

Remarks

None.

5.9.3.5.5.3 USB_DEVICE_MAX_INSTANCES Macro

C

```
#define USB_DEVICE_MAX_INSTANCES 1
```

Description

In case of microcontrollers with more than one USB peripheral, the value of this constant can be increased to support more than one instances of USB device layer.

Remarks

The static implementation supports only one instance. Setting the value of this constant to > 1 has no effect on static implementations. Only in dynamic implementations of the USB device layer this value can be set > 1.

The USB device layer has to support at least one instance. Therefore, ensure that the value of this constant is set to > 0.

Increasing the instance count consumes RAM and can lead to performance degradation.

5.9.3.6 Library Interface

Data Types and Constants

	Name	Description
	USB_DEVICE_INDEX_0	USB device layer index definitions.
	USB_DEVICE_INDEX_1	This is macro USB_DEVICE_INDEX_1.

	USB_DEVICE_INDEX_2	This is macro USB_DEVICE_INDEX_2.
	USB_DEVICE_INDEX_3	This is macro USB_DEVICE_INDEX_3.
	USB_DEVICE_INDEX_4	This is macro USB_DEVICE_INDEX_4.
	USB_DEVICE_INDEX_5	This is macro USB_DEVICE_INDEX_5.
	USB_DEVICE_INDEX_COUNT	Number of valid USB Device Layer indices
	USB_DEVICE_CALLBACK	Pointer to a USB Device Layer callback function data type for bus events.
	USB_DEVICE_CONFIG_DESCS_PTR	Pointer to an array that contains pointer to configuration descriptors.
	USB_DEVICE_CONTROL_STATUS	This enumerated data-type identifies the status stage of control transfer.
	USB_DEVICE_CONTROL_TRANSFER_CALLBACK	
	USB_DEVICE_CONTROL_TRANSFER_EVENT	This datatype defines the different control transfer events.
	USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA	USB device control transfer event data.
	USB_DEVICE_CONTROL_TRANSFER_HANDLE	Data type of USB device control transfer handle.
	USB_DEVICE_CONTROL_TRANSFER_RESULT	Enumerated data type identifying results of a control transfer.
	USB_DEVICE_EVENT	Enumerated data-type identifying the bus events that has occurred in the USB device layer.
	USB_DEVICE_EVENT_DATA	Data associated with USB bus events.
	USB_DEVICE_EVENT_DATA_CONFIGURED	Data-type that holds the data related to USB device configured event.
	USB_DEVICE_FUNC_REGISTRATION_TABLE	A function driver has to be registered with the USB device layer using this structure.
	USB_DEVICE_FUNCTION_DRIVER	A function driver has to expose standard APIs to device layer using following structure.
	USB_DEVICE_HANDLE	Data type for USB device handle.
	USB_DEVICE_INIT	This structure has to be initialized by the system/application and must be passed as parameter to USB_DEVICE_Initialize() .
	USB_DEVICE_POWER_STATE	Enumerated data type that identifies if the device is self powered or bus powered .
	USB_DEVICE_REMOTE_WAKEUP_STATUS	Enumerated data type that identifies if the remote wakeup status of the device.
	USB_DEVICE_STATE	Standard USB device states as described in Chapter-9 of USB 2.0 Specification.
	USB_DEVICE_STRING_DESCS_PTR	Pointer to an array that contains pointer to string descriptors.
	USB_MASTER_DESCRIPTOR	Global USB descriptor structure containing pointers to standard USB descriptor structures.
	USB_DEVICE_ENDPOINT_TABLE_SIZE	This is macro USB_DEVICE_ENDPOINT_TABLE_SIZE.
	USB_DEVICE_HANDLE_INVALID	Macro that defines the value of invalid device handle.

Client Core Functions

	Name	Description
	USB_DEVICE_Open	Opens the specified USB device layer instance and returns a handle to it.
	USB_DEVICE_ControlReceive	Receives data stage of the control transfer from host to device.
	USB_DEVICE_ControlSend	Sends data stage of the control transfer from device to host.
	USB_DEVICE_Attach	This function will attach the device to the USB.

	USB_DEVICE_Detach	This function will detach the device from the USB.
	USB_DEVICE_Status	Provides the current status of the USB device layer
	USB_DEVICE_ClientStatus	Gets the current client-specific status of the USB device layer.
	USB_DEVICE_ResumeStart	This function will start the resume signalling.
	USB_DEVICE_ResumeStop	This function will stop the resume signalling.
	USB_DEVICE_GetDeviceSpeed	Informs the client of the current operation speed of the USB bus.
	USB_DEVICE_GetDeviceState	Returns the current state of the USB device.
	USB_DEVICE_GetConfigurationValue	Informs the client of the current USB device configuration set by the USB host.
	USB_DEVICE_RemoteWakeupIsEnabled	Gets the "Remote wakeup" status of the device.
	USB_DEVICE_PowerStateSet	Sets power state of the device.

Driver Interaction Functions

	Name	Description
	USB_DEVICE_ControlStatus	Initiates status stage of the control transfer.
	USB_DEVICE_EndpointEnable	This function enables a endpoint for the specified direction and endpoint size.
	USB_DEVICE_EndpointIsEnabled	This function returns the enable/ disable status of the specified endpoint and direction.
	USB_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	USB_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	USB_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	USB_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	USB_DEVICE_IRPSubmit	This function submits a I/O Request Packet (IRP) for processing to the USB Driver.

System Interaction Functions

	Name	Description
	USB_DEVICE_Deinitialize	Deinitializes the specified instance of the USB device layer.
	USB_DEVICE_Close	Closes an opened handle to an instance of the USB device layer.
	USB_DEVICE_Initialize	Creates and initializes an instance of the USB device layer.
	USB_DEVICE_ControlEventCallBackSet	Client can register its call-back function into the device layer to get control transfer events.
	USB_DEVICE_Reinitialize	Reinitializes the USB device layer
	USB_DEVICE_EndpointDisable	This function disables an endpoint.
	USB_DEVICE_EventCallBackSet	Client can register its call-back function into the device layer.
	USB_DEVICE_Tasks	USB Device layer calls all other function driver tasks in this function. It also generates and forwards events to its clients.

Description

This section describes the Application Programming Interface (API) functions of the USB device layer library

Refer to each section for a detailed description.

5.9.3.6.1 System Interaction Functions

5.9.3.6.1.1 USB_DEVICE_Deinitialize Function

C

```
void USB_DEVICE_Deinitialize(  
    SYS_MODULE_OBJ usbDeviceObj  
);
```

Description

This function deinitializes the specified instance of the USB device layer, disabling its operation (and any hardware) and invalidates all of the internal data.

Preconditions

Function [USB_DEVICE_Initialize](#) must have been called before calling this routine and a valid [SYS_MODULE_OBJ](#) must have been returned.

Parameters

Parameters	Description
object	USB device layer object handle, returned by USB_DEVICE_Initialize

Returns

None.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Example

5.9.3.6.1.2 USB_DEVICE_Close Function

C

```
void USB_DEVICE_Close(  
    USB_DEVICE_HANDLE usbDeviceHandle  
);
```

Description

This function closes an opened handle to an instance of the USB device layer, invalidating the handle.

Preconditions

The [USB_DEVICE_Initialize](#) function must have been called for the specified device layer instance. [USB_DEVICE_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from USB_DEVICE_Open

Returns

None

Remarks

After calling this routine, the handle passed in "usbDevHandle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling [USB_DEVICE_Open\(\)](#) before the client may use the device layer again.

Example

```

USB_DEVICE_HANDLE usbDeviceHandle;

// Before opening a handle, USB device must have been initialized
// by calling USB_DEVICE_Initialize().
usbDeviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0 );

if(USB_DEVICE_HANDLE_INVALID == usbDeviceHandle)
{
    //Failed to open handle.
}

.....
.....
// User's code
.....
.....
// Close handle
USB_DEVICE_Close( usbDevHandle );

```

5.9.3.6.1.3 USB_DEVICE_Initialize Function**C**

```

SYS_MODULE_OBJ USB_DEVICE_Initialize(
    const SYS_MODULE_INDEX instanceIndex,
    const SYS_MODULE_INIT * const init
);

```

Description

This function initializes an instance of USB device layer, making it ready for clients to open and use it. The number of instances is limited by the value of macro [USB_DEVICE_MAX_INSTANCES](#).

Preconditions

None.

Parameters

Parameters	Description
instanceIndex	In case of microcontrollers with multiple USB peripherals, user can create multiple instances of USB device layer. Parameter instanceIndex identifies this instance.
init	Pointer to a data structure containing any data necessary to initialize the USB device layer

Returns

If successful, returns a valid handle to a device layer object. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#).

Remarks

This routine must be called before any other USB Device Layer routine is called and after the initialization of USB Device Driver. This routine should only be called once during system initialization.

Example

```

USB_DEVICE_INIT deviceLayerInit;
SYS_MODULE_OBJ usbDeviceObj;

// System module initialization
deviceLayerInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
// Identifies peripheral (PLIB-level) ID
deviceLayerInit.usbID = USB_ID_1;
// Boolean flag: true -> Stop USB module in Idle Mode
deviceLayerInit.stopInIdle = false;

```

```

// Boolean flag: true -> Suspend USB in Sleep Mode
deviceLayerInit.suspendInSleep = false;
// Interrupt Source for USB module
deviceLayerInit.interruptSource = INT_SOURCE_USB_1;
// Number of function drivers registered to this instance of the
USB device layer
deviceLayerInit.registeredFuncCount = 1;
// Function driver table registered to this instance of the USB device layer
deviceLayerInit.registeredFunctions = funcRegistrationTable;
// Pointer to USB Descriptor structure
deviceLayerInit.usbMasterDescriptor = &usbMasterDescriptor;

// USB device initialization
usbDeviceObj = USB_DEVICE_Initialize(USB_DEVICE_INDEX_0, &deviceLayerInit);

if (SYS_MODULE_OBJ_INVALID == usbDeviceObj)
{
    // Handle error
}

```

5.9.3.6.1.4 USB_DEVICE_ControlEventCallbackSet Function

C

```

USB_ERROR USB_DEVICE_ControlEventCallbackSet(
    USB_DEVICE_HANDLE usbDeviceHandle,
    const USB_DEVICE_CONTROL_TRANSFER_CALLBACK callBackFunc
);

```

Description

The USB Device Layer notifies the control transfer events to the client by calling callBackFunc.

Preconditions

The device layer must have been initialized by calling [USB_DEVICE_Initialize](#) and a valid handle to the instance must have been obtained by calling [USB_DEVICE_Open](#).

Parameters

Parameters	Description
usbDeviceHandle	Pointer to the device layer handle that is returned from USB_DEVICE_Open
callBackFunc	Pointer to the control transfer event handler. The device layer notifies the client about control transfer event by calling this function.

Returns

Returns USB_ERROR_NONE if successful.

Remarks

None.

Example

5.9.3.6.1.5 USB_DEVICE_Reinitialize Function

C

```

void USB_DEVICE_Reinitialize(
    SYS_MODULE_OBJ usbDeviceObj,
    const SYS_MODULE_INIT * const init
);

```

Description

This function reinitializes the USB device layer.

Preconditions

USB device driver and USB device layer must have been initialized.

Parameters

Parameters	Description
usbDeviceObj	Driver object handle, returned by USB_DEVICE_Initialize
init	Pointer to a data structure containing any data necessary to reinitialize the USB device layer.

Returns

None

Remarks

This function can be called multiple times to reinitialize the USB device layer.

This operation reinitializes all the module variables of the USB device layer associated with the instance specified by the parameter "usbDeviceObj".

Example

```

USB_DEVICE_INIT deviceLayerInit;

// System module initialization
deviceLayerInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
// Identifies peripheral (PLIB-level) ID
deviceLayerInit.usbID = USB_ID_1;
// Boolean flag: true -> Stop USB module in Idle Mode
deviceLayerInit.stopInIdle = false;
// Boolean flag: true -> Suspend USB in Sleep Mode
deviceLayerInit.suspendInSleep = false;
// Interrupt Source for USB module
deviceLayerInit.interruptSource = INT_SOURCE_USB_1;
// Number of function drivers registered to this instance of the
USB device layer
deviceLayerInit.registeredFuncCount = 1;
// Function driver table registered to this instance of the USB device layer
deviceLayerInit.registeredFunctions = funcRegistrationTable;
// Pointer to USB Descriptor structure
deviceLayerInit.usbMasterDescriptor = &usbMasterDescriptor;

// USB device initialization
usbDeviceObj = USB_DEVICE_Initialize(USB_DEVICE_INDEX_0, &deviceLayerInit);

if (SYS_MODULE_OBJ_INVALID == usbDeviceObj)
{
}

// Do something here.

// Re-initialize if required.
// System module initialization
deviceLayerInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
// Identifies peripheral (PLIB-level) ID
deviceLayerInit.usbID = USB_ID_1;
// Boolean flag: true -> Stop USB module in Idle Mode
deviceLayerInit.stopInIdle = false;
// Boolean flag: true -> Suspend USB in Sleep Mode
deviceLayerInit.suspendInSleep = false;
// Interrupt Source for USB module
deviceLayerInit.interruptSource = INT_SOURCE_USB_1;
// Number of function drivers registered to this instance of the

```

```

USB device layer
deviceLayerInit.registeredFuncCount = 1;
// Function driver table registered to this instance of the USB device layer
deviceLayerInit.registeredFunctions = funcRegistrationTable;
// Pointer to USB Descriptor structure
deviceLayerInit.usbMasterDescriptor = &usbMasterDescriptor;

USB_DEVICE_ReInitialize(USB_DEVICE_INDEX_0, &deviceLayerInit);

```

5.9.3.6.1.6 USB_DEVICE_EndpointDisable Function

C

```

USB_ERROR USB_DEVICE_EndpointDisable(
    USB_DEVICE_HANDLE usbDeviceHandle,
    USB_ENDPOINT endpointAndDirection
);

```

Description

This function disables an endpoint. If the endpoint type is control type then both directions are disabled. For non-control endpoints, the function disables one direction at a time.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
endpointAndDirection	Specifies the endpoint and direction.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

This function must not be called by the application clients. Application has no access to modify the endpoint features.

Example

```

// This code snippet shows an example of how to disable
// a control endpoint. Note that the direction parameter is ignored.
// For a control endpoint, both the directions are disabled.

handle = USB_DEVICE_Open(USB_DEVICE_INDEX_0);
USB_ENDPOINT ep;

ep.endpoint = 0;
ep.direction = USB_DATA_DIRECTION_DEVICE_TO_HOST;

USB_DEVICE_EndpointDisable(handle, ep );

// This code snippet shows an example of how to disable a BULK IN
// endpoint

USB_ENDPOINT ep;

ep.endpoint = 1;
ep.direction = USB_DATA_DIRECTION_DEVICE_TO_HOST;

USB_DEVICE_EndpointDisable(handle, ep );

```

5.9.3.6.1.7 USB_DEVICE_EventCallbackSet Function

C

```
USB_ERROR USB_DEVICE_EventCallbackSet(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    const USB_DEVICE_CALLBACK callBackFunc  
);
```

Description

The USB Device Layer notifies the event to the client by calling callBackFunc.

Preconditions

The device layer must have been initialized by calling [USB_DEVICE_Initialize](#) and a valid handle to the instance must have been obtained by calling [USB_DEVICE_Open](#).

Parameters

Parameters	Description
usbDeviceHandle	Pointer to the device layer handle that is returned from USB_DEVICE_Open
callBackFunc	Pointer to the call back function. The device layer calls notifies the client about bus event by calling this function.

Returns

Returns USB_ERROR_NONE if successful.

Remarks

None.

Example

5.9.3.6.1.8 USB_DEVICE_Tasks Function

C

```
void USB_DEVICE_Tasks(  
    SYS_MODULE_OBJ object  
);
```

Description

This function must be periodically called by the user application. The USB Device layer calls all other function driver tasks in this function. It also generates and forwards events to its clients.

Preconditions

Device layer must have been initialized by calling [USB_DEVICE_Initialize](#).

Parameters

Parameters	Description
devLayerObj	Pointer to the Device Layer Object that is returned from USB_DEVICE_Initialize

Returns

none.

Remarks

This function must be called only after the device layer is initialized by calling function [USB_DEVICE_Initialize](#).

Example

5.9.3.6.2 Client Core Functions

5.9.3.6.2.1 USB_DEVICE_Open Function

C

```
USB_DEVICE_HANDLE USB_DEVICE_Open(  
    const SYS_MODULE_INDEX instanceIndex,  
    const DRV_IO_INTENT intent  
);
```

Description

This function opens the USB device layer instance specified by instance index and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the USB device layer. The number of handles a client can open is limited by the value set to [USB_DEVICE_MAX_CLIENTS](#).

Preconditions

This function must be called after USB device driver initialization and after the initialization of USB Device Layer.

Parameters

Parameters	Description
instanceIndex	USB device layer instance index
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Returns

If successful, returns a valid handle to a device layer object. Otherwise, it returns [SYS_MODULE_OBJ_INVALID](#).

Remarks

This routine must be called after USB device driver initialization and after the initialization of USB Device Layer. This routine should be called only once during system initialization.

Example

```
USB_DEVICE_HANDLE usbDeviceHandle;  
  
// Before opening a handle, USB device must have been initialized  
// by calling USB_DEVICE_Initialize().  
usbDeviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0, DRV_IO_INTENT_BLOCKING );  
  
if(USB_DEVICE_HANDLE_INVALID == usbDeviceHandle)  
{  
    //Failed to open handle.  
}
```

5.9.3.6.2.2 USB_DEVICE_ControlReceive Function

C

```
USB_DEVICE_CONTROL_TRANSFER_RESULT USB_DEVICE_ControlReceive(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    USB_DEVICE_CONTROL_TRANSFER_HANDLE controlXferHandle,  
    void * data,  
    size_t length  
);
```

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
controlXferHandle	Control transfer handle that is returned by bus event callback function.
data	Pointer to buffer that holds data.
length	Size in bytes.

Returns

USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED - If control transfer is failed due to host aborting the previous control transfer.

USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS - Control endpoint is successfully armed with data buffer.

Example**5.9.3.6.2.3 USB_DEVICE_ControlSend Function****C**

```
USB_DEVICE_CONTROL_TRANSFER_RESULT USB_DEVICE_ControlSend(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    USB_DEVICE_CONTROL_TRANSFER_HANDLE controlXferHandle,  
    void * data,  
    size_t length  
);
```

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
controlXferHandle	Control transfer handle that is returned by the in bus event callback function.
data	Pointer to buffer that holds data.
length	Size in bytes.

Returns

USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED - If control transfer is failed due to host aborting the previous control transfer.

USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS - Control endpoint is successfully armed with data buffer.

Example**5.9.3.6.2.4 USB_DEVICE_Attach Function****C**

```
void USB_DEVICE_Attach(  
    USB_DEVICE_HANDLE usbDeviceHandle  
);
```

Description

This function will attach the device to the USB. It does this by enabling the pull up resistors on the D+ or D- lines. This function should be called when the USB device layer is ready to receive communication from the host (typically after all initialization is complete).

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	Client's USB device layer handle (returned from USB_DEVICE_Open)

Returns

None.

Remarks

None.

Example

```

USB_DEVICE_INIT deviceLayerInit;
SYS_MODULE_OBJ usbDeviceObj;
USB_DEVICE_HANDLE usbDeviceHandle;

// System module initialization
deviceLayerInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
// Identifies peripheral (PLIB-level) ID
deviceLayerInit.usbID = USB_ID_1;
// Boolean flag: true -> Stop USB module in Idle Mode
deviceLayerInit.stopInIdle = false;
// Boolean flag: true -> Suspend USB in Sleep Mode
deviceLayerInit.suspendInSleep = false;
// Interrupt Source for USB module
deviceLayerInit.interruptSource = INT_SOURCE_USB_1;
// Number of function drivers registered to this instance of the
USB device layer
deviceLayerInit.registeredFuncCount = 1;
// Function driver table registered to this instance of the USB device layer
deviceLayerInit.registeredFunctions = funcRegistrationTable;
// Pointer to USB Descriptor structure
deviceLayerInit.usbMasterDescriptor = &usbMasterDescriptor;

// USB device initialization
usbDeviceObj = USB_DEVICE_Initialize(USB_DEVICE_INDEX_0, &deviceLayerInit);

if (SYS_MODULE_OBJ_INVALID == usbDeviceObj)
{
    // Handle error
}

// Get an handle to the USB device layer.
usbDeviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0,
                                   DRV_IO_INTENT_BLOCKING );

if(USB_DEVICE_HANDLE_INVALID == usbDeviceHandle)
{
    // Failed to open handle.
    // Handle error.
}

// Now, connect device to USB
USB_DEVICE_Attach(usbDeviceHandle);

```

5.9.3.6.2.5 USB_DEVICE_Detach Function**C**

```
void USB_DEVICE_Detach(
```

```
    USB_DEVICE_HANDLE usbDeviceHandle  
);
```

Description

This function will detach the device from the USB. It does this by disabling the pull up resistors on the D+ or D- lines. This function should be called when the application wants to disconnect the device from the bus (typically to implement a soft detach or switch to host mode operation).

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	Client's driver handle (returned from USB_DEVICE_Open)

Returns

None.

Remarks

None.

Example

```
USB_DEVICE_HANDLE usbDeviceHandle;  
  
// Detach the device from the USB  
USB_DEVICE_Detach( usbDeviceHandle );
```

5.9.3.6.2.6 USB_DEVICE_Status Function

C

```
SYS_STATUS USB_DEVICE_Status(  
    SYS_MODULE_OBJ object  
);
```

Description

This function provides the current status of the USB device layer.

Preconditions

The [USB_DEVICE_Initialize](#) function must have been called before calling this function.

Parameters

Parameters	Description
object	Driver object handle, returned from USB_DEVICE_Initialize

Returns

SYS_STATUS_READY - Indicates that the device is busy with a previous system level operation and cannot start another

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the device is ready to [accept](#) new operations.

SYS_STATUS_BUSY - Indicates that the device is busy with a previous system level operation and cannot start another

SYS_STATUS_UNINITIALIZED - Indicates that the device has never been initialized

SYS_STATUS_ERROR - Indicates that the device is in an error state

Any value less than `SYS_STATUS_ERROR` is also an error state.

None.

Example

```
SYS_MODULE_OBJ    object;    // Returned from DRV_USB_Initialize
SYS_STATUS        status;

status = USB_DEVICE_Status(object);

if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

5.9.3.6.2.7 USB_DEVICE_ClientStatus Function

C

```
DRV_CLIENT_STATUS USB_DEVICE_ClientStatus(
    USB_DEVICE_HANDLE usbDevHandle
);
```

Description

This function gets the client-specific status of the USB device layer associated with the specified handle.

Preconditions

The [USB_DEVICE_Initialize](#) function must have been called. [USB_DEVICE_Open](#) must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open instance handle, returned from USB_DEVICE_Open

Returns

A value of enum type [DRV_CLIENT_STATUS](#) describing the current status of the USB device layer.

Remarks

None.

Example

5.9.3.6.2.8 USB_DEVICE_ResumeStart Function

C

```
void USB_DEVICE_ResumeStart(
    USB_DEVICE_HANDLE usbDeviceHandle
);
```

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	Client's driver handle (returned from USB_DEVICE_Open)

Returns

None.

Remarks

None.

Example**5.9.3.6.2.9 USB_DEVICE_ResumeStop Function****C**

```
void USB_DEVICE_ResumeStop(  
    USB_DEVICE_HANDLE usbDeviceHandle  
)  
;
```

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	Client's driver handle (returned from USB_DEVICE_Open)

Returns

None.

Remarks

None.

Example**5.9.3.6.2.10 USB_DEVICE_GetDeviceSpeed Function****C**

```
USB_SPEED USB_DEVICE_GetDeviceSpeed(  
    USB_DEVICE_HANDLE usbDeviceHandle  
)  
;
```

Description

The USB device stack supports both high speed and full speed operations. This function returns the current operation speed of the USB bus.

Preconditions

The USB device layer must have been initialized and a valid handle to USB device layer must have been opened.

Parameters

Parameters	Description
usbDeviceHandle	Pointer to device layer handle that is returned from USB_DEVICE_Open

Returns

USB_SPEED_LOW - USB module is at low speed

USB_SPEED_FULL - USB module is at full speed

USB_SPEED_HIGH - USB module is at high speed

Remarks

This function must be called only after the device layer is initialized and opened by calling [USB_DEVICE_Initialize](#) and [USB_DEVICE_Open](#).

Example

5.9.3.6.2.11 USB_DEVICE_GetDeviceState Function

C

```
USB_DEVICE_STATE USB_DEVICE_GetDeviceState(  
    USB_DEVICE_HANDLE usbDeviceHandle  
);
```

Description

This function returns the current state of the USB device, as described in Chapter 9 of USB 2.0 Specification.

Preconditions

The USB device layer must have been initialized and opened before calling this function.

Parameters

Parameters	Description
usbDeviceHandle	Pointer to the device layer handle that is returned from USB_DEVICE_Open

Returns

USB_DEVICE_STATE_DETACHED - Device is not in any of the known states

USB_DEVICE_STATE_ATTACHED - Device is attached to the USB, but is not powered

USB_DEVICE_STATE_POWERED - Device is attached to the USB and powered, but has not been reset

USB_DEVICE_STATE_DEFAULT - Device is attached to the USB and powered and has been reset, but has not been assigned a unique address

USB_DEVICE_STATE_ADDRESS - Device is attached to the USB, powered, has been reset, and a unique device address has been assigned

USB_DEVICE_STATE_CONFIGURED - Device is attached to the USB, powered, has been reset, has a unique address, is configured, and is not suspended

USB_DEVICE_STATE_SUSPENDED - Device is, at minimum, attached to the USB and is powered and has not seen bus activity for 3 ms. The device is still in addressed state.

Remarks

This function must be called only after the device layer is initialized and opened by calling [USB_DEVICE_Initialize](#) and [USB_DEVICE_Open](#).

Example

```
USB_DEVICE_STATE usbDevState;  
  
// Get USB Device State.  
usbDevState = USB_DEVICE_GetDeviceState( usbDeviceHandle );  
  
switch(usbDevState)  
{  
    case USB_DEVICE_STATE_ATTACHED:  
        // Add code here  
        break;  
  
    case USB_DEVICE_STATE_POWERED:  
        // Add code here  
        break;  
  
    default:
```

```
        break;  
    }
```

5.9.3.6.2.12 USB_DEVICE_GetConfigurationValue Function

C

```
uint8_t USB_DEVICE_GetConfigurationValue(  
    USB_DEVICE_HANDLE usbDeviceHandle  
);
```

Description

This function returns the current active USB device configuration.

Preconditions

The USB Device Layer must have been initialized and opened before calling this function.

Parameters

Parameters	Description
usbDeviceHandle	Pointer to the Device Layer Handle that is returned from USB_DEVICE_Open

Returns

Present active configuration.

Remarks

This function must be called only after the device layer is initialized and opened by calling [USB_DEVICE_Initialize](#) and [USB_DEVICE_Open](#).

Example

5.9.3.6.2.13 USB_DEVICE_RemoteWakeupIsEnabled Function

C

```
USB_DEVICE_REMOTE_WAKEUP_STATUS USB_DEVICE_RemoteWakeupIsEnabled(  
    USB_DEVICE_HANDLE usbDeviceHandle  
);
```

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .

Returns

USB_DEVICE_REMOTE_WAKEUP_ENABLED - Remote wakeup is enabled. USB_DEVICE_REMOTE_WAKEUP_DISABLED - Remote wakeup is disabled.

Example

5.9.3.6.2.14 USB_DEVICE_PowerStateSet Function

C

```
void USB_DEVICE_PowerStateSet(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    USB_DEVICE_POWER_STATE powerState  
);
```

Description

Application clients can use this function to set the power state of the device. A USB device can be bus powered or self powered.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
powerState	USB_DEVICE_POWER_STATE_BUS_POWERED/ USB_DEVICE_POWER_STATE_SELF_POWERED

Returns

None.

Example

5.9.3.6.3 Driver Interaction Functions

5.9.3.6.3.1 USB_DEVICE_ControlStatus Function

C

```
USB_DEVICE_CONTROL_TRANSFER_RESULT USB_DEVICE_ControlStatus(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    USB_DEVICE_CONTROL_TRANSFER_HANDLE controlTransferHandle,  
    USB_DEVICE_CONTROL_STATUS status  
);
```

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
controlTransferHandle	Control transfer handle that is returned by bus event callback function.
status	USB_DEVICE_CONTROL_STATUS_SEND_ZLP/ USB_DEVICE_CONTROL_STATUS_STALL

Returns

USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED - If control transfer is failed due to host aborting the previous control transfer.

USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS - Control endpoint is successfully armed with data buffer.

Example

5.9.3.6.3.2 USB_DEVICE_EndpointEnable Function

C

```
USB_ERROR USB_DEVICE_EndpointEnable(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    USB_ENDPOINT endpointAndDirection,  
    USB_TRANSFER_TYPE transferType,  
    uint16_t endpointSize  
);
```



```
);
```

Description

This function enables a endpoint for the specified direction and endpoint size. The function will enable the endpoint for communication in one direction at a time. It must be called twice if the endpoint is required to communicate in both the directions, except for control endpoints. If the endpoint type is a control endpoint, the endpoint is always bi-directional and the function needs to be called only once. The size of the endpoint must match the wMaxPacketSize reported in the endpoint descriptor for this endpoint. The function does not check if the endpoint is already in use. It is the client's responsibility to make sure that a endpoint is not accidentally re-used.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
endpointAndDirection	Specifies the endpoint and direction.
transferType	Should be USB_TRANSFER_TYPE_CONTROL for control endpoint, USB_TRANSFER_TYPE_BULK for bulk endpoint, USB_TRANSFER_TYPE_INTERRUPT for interrupt endpoint and USB_TRANSFER_TYPE_ISOCHRONOUS for isochronous endpoint.
endpointSize	Maximum size (in bytes) of the endpoint as reported in the endpoint descriptor.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

None.

Example

```
// This code snippet shows an example of how to enable endpoint
// 0 for control transfers. Note that for a control endpoint, the
// direction parameter is ignored. A control endpoint is always
// bi-directional. Endpoint size is 64 bytes.

USB_ENDPOINT ep;

handle = USB_DEVICE_Open(USB_DEVICE_INDEX_0);
ep.endpoint = 0;
ep.direction = USB_DATA_DIRECTION_DEVICE_TO_HOST;

USB_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_CONTROL, 64);

// This code snippet shows an example of how to set up a endpoint
// for BULK IN transfer. For an IN transfer, data moves from device
// to host. In this example, endpoint 1 is enabled. The maximum
// packet size is 64.

USB_ENDPOINT ep;

ep.endpoint = 1;
ep.direction = USB_DATA_DIRECTION_DEVICE_TO_HOST;

USB_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);

// If endpoint 1 must also be set up for BULK OUT, then the enable
// function must be called again, as shown here.
```

```
ep.endpoint = 1;
ep.direction = USB_DATA_DIRECTION_HOST_TO_DEVICE;

USB_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);
```

5.9.3.6.3.3 USB_DEVICE_EndpointIsEnabled Function

C

```
bool USB_DEVICE_EndpointIsEnabled(
    USB_DEVICE_HANDLE usbDeviceHandle,
    USB_ENDPOINT endpointAndDirection
);
```

Description

This function returns the enable/ disable status of the specified endpoint and direction.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
endpointAndDirection	Specifies the endpoint and direction.

Returns

Returns true if the endpoint is enabled, false otherwise.

Remarks

This function must not be called by the application clients. Application has no access to read the endpoint features.

Example

```
// This code snippet shows an example of how the
// USB_DEVICE_EndpointIsEnabled() function can be used to obtain the
// status of the endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep.endpoint = 1;
ep.direction = USB_DATA_DIRECTION_DEVICE_TO_HOST;

if(DRV_USB_ENDPOINT_STATE_DISABLED ==
    USB_DEVICE_EndpointIsEnabled(handle, ep))
{
    // Endpoint is disabled. Enable endpoint.

    USB_DEVICE_EndpointEnable(handle, ep, USB_ENDPOINT_TYPE_BULK, 64);
}
```

5.9.3.6.3.4 USB_DEVICE_EndpointIsStalled Function

C

```
bool USB_DEVICE_EndpointIsStalled(
    USB_DEVICE_HANDLE usbDeviceHandle,
    USB_ENDPOINT endpoint
);
```

Description

This function returns the stall status of the specified endpoint and direction.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
endpointAndDirection	Specifies the endpoint and direction.

Returns

Returns true if endpoint is stalled, false otherwise.

Remarks

This function must not be called by the application clients. Application has no access to read the endpoint features.

Example

```
// This code snippet shows an example of how the  
// USB_DEVICE_EndpointIsStalled() function can be used to obtain the  
// stall status of the endpoint 1 and IN direction.
```

```
USB_ENDPOINT ep;  
  
ep.endpoint = 1;  
ep.direction = USB_DATA_DIRECTION_DEVICE_TO_HOST;  
  
if(true == USB_DEVICE_EndpointIsStalled (handle, ep))  
{  
    // Endpoint stall is enabled. Clear the stall.  
    USB_DEVICE_EndpointStallClear(handle, ep);  
}
```

5.9.3.6.3.5 USB_DEVICE_EndpointStall Function**C**

```
USB_ERROR USB_DEVICE_EndpointStall(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    USB_ENDPOINT endpointAndDirection  
);
```

Description

This function stalls an endpoint in the specified direction.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
endpointAndDirection	Specifies the endpoint and direction.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

This function must not be called by the application clients. Application has no access to modify the endpoint features.

Example

```
// This code snippet shows an example of how to stall an endpoint. In  
// this case , endpoint 1 IN direction is stalled.
```

```
USB_ENDPOINT ep;  
  
ep.endpoint = 1;  
ep.direction = USB_DATA_DIRECTION_DEVICE_TO_HOST;  
  
USB_DEVICE_EndpointStall(handle, ep);
```

5.9.3.6.3.6 USB_DEVICE_EndpointStallClear Function**C**

```
USB_ERROR USB_DEVICE_EndpointStallClear(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    USB_ENDPOINT endpointAndDirection  
);
```

Description

This function clears the stall on an endpoint in the specified direction.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
endpointAndDirection	Specifies the endpoint and direction.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

This function must not be called by the application clients. Application has no access to modify the endpoint features.

Example

```
// This code snippet shows an example of how to clear a stall. In this  
// example. the stall on endpoint 1 IN direction is cleared.
```

```
USB_ENDPOINT ep;  
  
ep.endpoint = 1;  
ep.direction = USB_DATA_DIRECTION_DEVICE_TO_HOST;  
  
USB_DEVICE_EndpointStallClear(handle, ep);
```

5.9.3.6.3.7 USB_DEVICE_IRPCancelAll Function**C**

```
USB_ERROR USB_DEVICE_IRPCancelAll(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    USB_ENDPOINT endpointAndDirection  
);
```

Description

This function cancels all IRPs that are queued and in progress at the specified endpoint. The driver checks if IRPs are to be cancelled at two instances; when it has called the IRP callback and is ready to process the next IRP in the queue and when it is ready to process the next transaction in the the current IRP. It is recommended that this function be called in the IRP callback of the IRP that just completed as this ensures that an IRP in progress is not cancelled. Cancelling an IRP that is progress may cause disturbance to the USB host firmware. Cancelling the IRP from any other location, other than a IRP callback could cause an IRP that is in progress to get cancelled.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
endpointAndDirection	Specifies the endpoint and direction.

Returns

USB_ERROR_NONE - The endpoint was successfully enabled. USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Remarks

This function must not be called by the application clients. Application has no access modify the IRPs.

Example

// This code snippet shows an exampl of how to cancel all IRPs.

```
void MyIRPCallback(USB_DEVICE_IRP * irp)
{
    // Check if this is setup command

    if(irp->status == USB_DEVICE_IRP_STATUS_SETUP)
    {
        if(IsSetupCommandSupported(irp->data) == false)
        {
            // This means that this setup command is not
            // supported. Stall the endpoint.

            USB_DEVICE_IRPCancelAll(handle, ep);
            USB_DEVICE_EndpointStall(handle, ep);
        }
    }
}
```

5.9.3.6.3.8 USB_DEVICE_IRPSubmit Function**C**

```
USB_ERROR USB_DEVICE_IRPSubmit(
    USB_DEVICE_HANDLE usbDeviceHandle,
    USB_ENDPOINT endpointAndDirection,
    USB_DEVICE_IRP * irp
);
```

Description

This function submits a I/O Request Packet (IRP) for processing to the USB Driver. The IRP allows a client to [send](#) and receive data from the USB Host. The data will sent or received through the specified endpoint. The direction of the data transfer is indicated by the direction flag in the endpointAndDirection structure. Submitting an IRP arms the endpoint to either [send](#) data to

or receive data from the host. If an IRP is under progress on the endpoint, then the subsequent IRP submit operation will result in the IRPs getting queued. The contents of the IRP should not be changed till the IRP has been processed.

Particular attention should be paid to the size parameter of IRP. The following should be noted:

- The size parameter while sending data to the host can be less than, greater than, equal to or be an exact multiple of maximum packet size for the endpoint. The maximum packet size for the endpoint determines the number of transactions required to process the IRP.
- If the size parameter while sending data to the host is less than maximum packet size, the transfer will complete in one transaction.
- If the size parameter while sending data to the host is greater than maximum packet size, the IRP will be processed in multiple transactions.
- If the size parameter while sending data to the host is equal to or an exact multiple of the maximum packet size, the client can optionally ask the driver to **send** a Zero Length packet by specifying the USB_DEVICE_IRP_FLAG_SEND_ZLP flag as the flag parameter.
- The size parameter while receiving data from the host must be an exact multiple of the maximum packet size of the endpoint. If this is not the case, the driver will return a USB_DEVICE_IRP_SUBMIT_RESULT_INVALID_SIZE result. If while processing the IRP, the driver receives less than maximum packet size or a ZLP from the host, the driver considers the IRP as processed. The size parameter at this point contains the actual amount of data received from the host.

Preconditions

Client handle should be valid.

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open() .
endpointAndDirection	Specifies the endpoint and direction.
irp	Pointer to the USB_DEVICE_IRP structure.

Returns

USB_ERROR_NONE - if the IRP was submitted successful. USB_ERROR_IRP_SIZE_INVALID - if the size parameter of the IRP is not correct. USB_ERROR_IRP_QUEUE_FULL - if the driver IRP queue is full.

Remarks

This function must not be called by the application clients. Application must not submit IRP directly.

Example

*// The following code snippet shows an example of how to schedule a
// IRP to send data from device to host. Assume that max packet size
// is 64 and endpoint is 1.*

```

USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_DATA_DIRECTION_DEVICE_TO_HOST;
ep.endpoint = 1;

irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_NONE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (USB_DEVICE_IRPSubmit(handle, ep, irp)
    != USB_ERROR_NONE)
{

```

```

    // This means there was an error.
}
else
{
    // The status of the IRP can be checked.
    while(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Wait or run a task routine.
    }
}

// The following code snippet shows how the client can request
// the driver to send a ZLP when the size is an exact multiple of
// end point size.

irp.data = myDataBufferToSend;
irp.size = 128;
irp.flags = USB_DEVICE_IRP_FLAG_SEND_ZLP;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

// Note that while receiving data from the host, the size should
// be an exact multiple maximum packet size of the endpoint. In the
// example below, the USB_DEVICE_IRPSubmit() function will
// return a USB_DEVICE_IRP_SUBMIT_RESULT_INVALID_SIZE value.

ep.direction = USB_DATA_DIRECTION_HOST_TO_DEVICE;
ep.endpoint = 1;

irp.data = myDataBufferToSend;
irp.size = 60; // THIS SIZE IS NOT CORRECT
irp.flags = USB_DEVICE_IRP_FLAG_NONE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

```

5.9.3.6.4 Data Types and Constants

5.9.3.6.4.1 USB_DEVICE_INDEX_0 Macro

C

```
#define USB_DEVICE_INDEX_0 0
```

Description

USB Device Layer Index Numbers

These constants provide USB device layer index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the [USB_DEVICE_Initialize](#) and [USB_DEVICE_Open](#) routines to identify the device layer instance in use.

5.9.3.6.4.2 USB_DEVICE_INDEX_1 Macro

C

```
#define USB_DEVICE_INDEX_1 1
```

Description

This is macro USB_DEVICE_INDEX_1.

5.9.3.6.4.3 USB_DEVICE_INDEX_2 Macro

C

```
#define USB_DEVICE_INDEX_2 2
```

Description

This is macro USB_DEVICE_INDEX_2.

5.9.3.6.4.4 USB_DEVICE_INDEX_3 Macro

C

```
#define USB_DEVICE_INDEX_3 3
```

Description

This is macro USB_DEVICE_INDEX_3.

5.9.3.6.4.5 USB_DEVICE_INDEX_4 Macro

C

```
#define USB_DEVICE_INDEX_4 4
```

Description

This is macro USB_DEVICE_INDEX_4.

5.9.3.6.4.6 USB_DEVICE_INDEX_5 Macro

C

```
#define USB_DEVICE_INDEX_5 5
```

Description

This is macro USB_DEVICE_INDEX_5.

5.9.3.6.4.7 USB_DEVICE_INDEX_COUNT Macro

C

```
#define USB_DEVICE_INDEX_COUNT _USB_DEVICE_EXISTS
```

Description

USB Device Layer Module Index Count

This constant identifies number of valid USB device layer indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

5.9.3.6.4.8 USB_DEVICE_CALLBACK Type

C

```
typedef void (* USB_DEVICE_CALLBACK)(USB_DEVICE_EVENT event, USB_DEVICE_EVENT_DATA *  
eventData);
```


Description

USB Device Layer Callback Function Pointer

This is the data type of the callback function that will be called by the USB device layer when there is a bus event from USB device layer.

Remarks

A USB Device Layer callback function must have the following function signature:

```
void MyCallBack ( USB\_DEVICE\_EVENT event, USB\_DEVICE\_EVENT\_DATA * eventData );
```

Where, "event" indicates an event on the USB device layer, and "MyCallBack" can be any name desired as the function is called through the pointer.

5.9.3.6.4.9 USB_DEVICE_CONFIG_DESCS_PTR Type

C

```
typedef uint8_t** USB\_DEVICE\_CONFIG\_DESCS\_PTR;
```

Description

Configuration descriptors pointer

5.9.3.6.4.10 USB_DEVICE_CONTROL_STATUS Enumeration

C

```
typedef enum {
    USB\_DEVICE\_CONTROL\_STATUS\_OK,
    USB\_DEVICE\_CONTROL\_STATUS\_ERROR
} USB\_DEVICE\_CONTROL\_STATUS;
```

Description

Control transfer status stage

Members

Members	Description
USB_DEVICE_CONTROL_STATUS_OK	Control transfer is supported. Send ZLP in the status stage.
USB_DEVICE_CONTROL_STATUS_ERROR	Control transfer is not supported. Stall control endpoint.

Remarks

Also see, [USB_DEVICE_ControlStatus](#)

5.9.3.6.4.11 USB_DEVICE_CONTROL_TRANSFER_CALLBACK Type

C

```
typedef void (* USB\_DEVICE\_CONTROL\_TRANSFER\_CALLBACK)(USB\_DEVICE\_CONTROL\_TRANSFER\_HANDLE
handle, USB\_DEVICE\_CONTROL\_TRANSFER\_EVENT event, USB\_DEVICE\_CONTROL\_TRANSFER\_EVENT\_DATA *
eventData);
```

Description

USB Device Layer control transfer callback Function Pointer

This is the data type of the callback function that will be called by the USB device layer when there is a control transfer event from USB device layer.

Remarks

A USB Device Layer callback function must have the following function signature:

```
void MyCallBack ( USB\_DEVICE\_EVENT event, USB_DEVICE );
```

Where, "event" indicates an event on the USB device layer, and "MyCallBack" can be any name desired as the function is called through the pointer.

5.9.3.6.4.12 USB_DEVICE_CONTROL_TRANSFER_EVENT Enumeration**C**

```
typedef enum {
    USB_DEVICE_CONTROL_TRANSFER_ABORTED,
    USB_DEVICE_CONTROL_DATA_RECEIVED,
    USB_DEVICE_CONTROL_SETUP_REQUEST,
    USB_DEVICE_CONTROL_DATA_SENT = 3
} USB_DEVICE_CONTROL_TRANSFER_EVENT;
```

Description

USB Device control transfer events

Members

Members	Description
USB_DEVICE_CONTROL_TRANSFER_ABORTED	Previous control transfer was aborted.
USB_DEVICE_CONTROL_DATA_RECEIVED	Control transfer data stage was completed
USB_DEVICE_CONTROL_SETUP_REQUEST	A setup packet was received and control transfer is in setup stage
USB_DEVICE_CONTROL_DATA_SENT = 3	Control transfer data stage transmit is complete

Remarks

None.

5.9.3.6.4.13 USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA Union**C**

```
typedef union {
    USB_SETUP_PACKET * setupRequest;
} USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA;
```

Description

USB Device control transfer event data.

Members

Members	Description
USB_SETUP_PACKET * setupRequest;	Data associated with USB_DEVICE_CONTROL_SETUP_REQUEST

Remarks

None.

5.9.3.6.4.14 USB_DEVICE_CONTROL_TRANSFER_HANDLE Type**C**

```
typedef uintptr_t USB_DEVICE_CONTROL_TRANSFER_HANDLE;
```

Description

Data type of control transfer handle.

This is the data type of the handle that must be used by the application client for all control transfers.

Remarks

Also see, [USB_DEVICE_ControlSend](#) [USB_DEVICE_ControlReceive](#) [USB_DEVICE_ControlStatus](#)

5.9.3.6.4.15 USB_DEVICE_CONTROL_TRANSFER_RESULT Enumeration**C**

```
typedef enum {  
    USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED,  
    USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS  
} USB_DEVICE_CONTROL_TRANSFER_RESULT;
```

Description

These enumerated values are the possible return values for control transfer operation.

Members

Members	Description
USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED	Control transfer failed. This could be because the control transfer handle is no more valid since the control transfer was aborted by host by sending a new setup packet
USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS	Control transfer was successful

Remarks

Also see, [USB_DEVICE_ControlSend](#) [USB_DEVICE_ControlReceive](#) [USB_DEVICE_ControlStatus](#)

5.9.3.6.4.16 USB_DEVICE_EVENT Enumeration**C**

```
typedef enum {  
    USB_DEVICE_EVENT_RESET,  
    USB_DEVICE_EVENT_SUSPENDED,  
    USB_DEVICE_EVENT_RESUMED,  
    USB_DEVICE_EVENT_ERROR,  
    USB_DEVICE_EVENT_SOF,  
    USB_DEVICE_EVENT_DETACHED,  
    USB_DEVICE_EVENT_DECONFIGURED,  
    USB_DEVICE_EVENT_CONFIGURED,  
    USB_DEVICE_EVENT_ATTACHED  
} USB_DEVICE_EVENT;
```

Description

Datatype that identifies the event that is active in the USB device layer.

Members

Members	Description
USB_DEVICE_EVENT_RESET	USB bus reset occurred. This event is an indication to the application client that device layer is about to de-initialize the function drivers. For application client, this event means that it has to close any open handles to function drivers
USB_DEVICE_EVENT_SUSPENDED	This event is an indication to the application client that device is suspended and it can put the device to power-down mode if required.
USB_DEVICE_EVENT_RESUMED	This event indicates that device has resumed from suspended state.

USB_DEVICE_EVENT_ERROR	This event is an indication to the application client that an error occurred on the USB bus

5.9.3.6.4.17 USB_DEVICE_EVENT_DATA Union

C

```
typedef union {
    USB_DEVICE_EVENT_DATA_CONFIGURED eventConfigured;
} USB_DEVICE_EVENT_DATA;
```

Members

Members	Description
USB_DEVICE_EVENT_DATA_CONFIGURED eventConfigured;	Data related to configured event

Remarks

Also see [USB_DEVICE_CALLBACK](#)

5.9.3.6.4.18 USB_DEVICE_EVENT_DATA_CONFIGURED Structure

C

```
typedef struct {
    uint8_t configurationValue;
    USB_SPEED speed;
} USB_DEVICE_EVENT_DATA_CONFIGURED;
```

Members

Members	Description
uint8_t configurationValue;	Configuration value selected by the host
USB_SPEED speed;	USB speed at which the device is connected to host

Remarks

Also see [USB_DEVICE_EVENT_DATA](#) [USB_DEVICE_CALLBACK](#)

5.9.3.6.4.19 USB_DEVICE_FUNC_REGISTRATION_TABLE Structure

C

```
typedef struct {
    USB_SPEED speed;
    uint8_t configurationValue;
    uint8_t interfaceNumber;
    uint8_t numberOfInterfaces;
    SYS_MODULE_INDEX funcDriverIndex;
    void* funcDriverInit;
    USB_DEVICE_FUNCTION_DRIVER* driver;
} USB_DEVICE_FUNC_REGISTRATION_TABLE;
```

Description

Global USB Device function registration structure

Members

Members	Description
USB_SPEED speed;	Type of speed (high, full or low speed)
uint8_t configurationValue;	Configuration Value to which the function driver has to be tied

uint8_t interfaceNumber;	Interface number to which this function driver has to be tied
uint8_t numberOfInterfaces;	Number of interfaces used by the function
SYS_MODULE_INDEX funcDriverIndex;	Function driver instance index
void* funcDriverInit;	Pointer to a structure that contains function driver initialization data
USB_DEVICE_FUNCTION_DRIVER* driver;	Pointer to a standard structure that exposes function driver APIs to USB device layer

5.9.3.6.4.20 USB_DEVICE_FUNCTION_DRIVER Structure

C

```
typedef struct {
    void (* initializeByDescriptor)(SYS_MODULE_INDEX funcDriverIndex, USB_DEVICE_HANDLE
usbDeviceHandle, void* funcDriverInit, uint8_t interfaceNumber, uint8_t alternateSetting,
uint8_t descriptorType, uint8_t * pDescriptor);
    void (* deInitialize)(SYS_MODULE_INDEX funcDriverIndex);
    void (* controlTransferNotification)(USB_DEVICE_CONTROL_TRANSFER_HANDLE controlHandle,
SYS_MODULE_INDEX index, USB_DEVICE_CONTROL_TRANSFER_EVENT controlEvent,
USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA * controlEventData);
    void (* tasks)(SYS_MODULE_INDEX funcDriverIndex);
} USB_DEVICE_FUNCTION_DRIVER;
```

Description

USB function driver structure

All function drivers (including vendor specific ones) must provide callback functions to USB device layer in the format specified by this structure. The USB device layer calls these callback functions at the time of appropriate event.

Members

Members	Description
void (* initializeByDescriptor)(SYS_MODULE_INDEX funcDriverIndex, USB_DEVICE_HANDLE usbDeviceHandle, void* funcDriverInit, uint8_t interfaceNumber, uint8_t alternateSetting, uint8_t descriptorType, uint8_t * pDescriptor);	Initialize gets called by the Device layer when it receives set configuration event. The device layer will initialize a function driver for every descriptor. Based on the descriptor type the function driver has to initialize itself.
void (* deInitialize)(SYS_MODULE_INDEX funcDriverIndex);	deInit gets called when the device layer detects a device detach, change in configuration or on USB bus reset.
void (* controlTransferNotification)(USB_DEVICE_CONTROL_TRANSFER_HANDLE controlHandle, SYS_MODULE_INDEX index, USB_DEVICE_CONTROL_TRANSFER_EVENT controlEvent, USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA * controlEventData);	This function will be called by the device layer when there is an interface specific setup packet request
void (* tasks)(SYS_MODULE_INDEX funcDriverIndex);	Function driver Tasks

Remarks

Even the vendor specific function drivers must provide callback functions in this format.

5.9.3.6.4.21 USB_DEVICE_HANDLE Type

C

```
typedef uintptr_t USB_DEVICE_HANDLE;
```

Description

Data type for USB device handle.

The data type of the handle that is returned from [USB_DEVICE_Open](#) function.

Remarks

None.

5.9.3.6.4.22 USB_DEVICE_INIT Structure

C

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    unsigned int usbID;
    bool stopInIdle;
    bool suspendInSleep;
    INT_SOURCE interruptSource;
    void * endpointTable;
    uint16_t registeredFuncCount;
    USB_DEVICE_FUNC_REGISTRATION_TABLE * registeredFunctions;
    USB_MASTER_DESCRIPTOR * usbMasterDescriptor;
    USB_SPEED deviceSpeed;
} USB_DEVICE_INIT;
```

Description

USB Device Initialization Structure

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
unsigned int usbID;	Identifies peripheral (PLIB-level) ID
bool stopInIdle;	Boolean flag: true -> Stop USB module in Idle Mode
bool suspendInSleep;	Boolean flag: true -> Suspend USB in Sleep Mode
INT_SOURCE interruptSource;	Interrupt Source for USB module
void * endpointTable;	Endpoint Table Buffer
uint16_t registeredFuncCount;	Number of function drivers registered to this instance of the USB device layer
USB_DEVICE_FUNC_REGISTRATION_TABLE * registeredFunctions;	Function driver table registered to this instance of the USB device layer
USB_MASTER_DESCRIPTOR * usbMasterDescriptor;	Pointer to USB Descriptor structure
USB_SPEED deviceSpeed;	Speed at which this device speed should operate

Remarks

Also see, [USB_DEVICE_Initialization](#)

5.9.3.6.4.23 USB_DEVICE_POWER_STATE Enumeration

C

```
typedef enum {
    USB_DEVICE_POWER_STATE_BUS_POWERED,
    USB_DEVICE_POWER_STATE_SELF_POWERED
} USB_DEVICE_POWER_STATE;
```

Description

Power state

Members

Members	Description
USB_DEVICE_POWER_STATE_BUS_POWERED	Device is bus powered
USB_DEVICE_POWER_STATE_SELF_POWERED	Device is self powered

Remarks

Also see, [USB_DEVICE_PowerStateSet](#)

5.9.3.6.4.24 USB_DEVICE_REMOTE_WAKEUP_STATUS Enumeration**C**

```
typedef enum {  
    USB_DEVICE_REMOTE_WAKEUP_DISABLED,  
    USB_DEVICE_REMOTE_WAKEUP_ENABLED  
} USB_DEVICE_REMOTE_WAKEUP_STATUS;
```

Description

Remote wakeup status

Members

Members	Description
USB_DEVICE_REMOTE_WAKEUP_DISABLED	Remote wakeup is disabled
USB_DEVICE_REMOTE_WAKEUP_ENABLED	Remote wakeup is enabled

Remarks

Also see, [USB_DEVICE_RemoteWakeupsEnabled](#)

5.9.3.6.4.25 USB_DEVICE_STATE Enumeration**C**

```
typedef enum {  
    USB_DEVICE_STATE_DETACHED,  
    USB_DEVICE_STATE_ATTACHED,  
    USB_DEVICE_STATE_POWERED,  
    USB_DEVICE_STATE_DEFAULT,  
    USB_DEVICE_STATE_ADDRESSED,  
    USB_DEVICE_STATE_CONFIGURED,  
    USB_DEVICE_STATE_SUSPENDED  
} USB_DEVICE_STATE;
```

Description

USB device states as described in chapter-9 of USB 2.0 specification

This data type identifies the USB Device States.

Members

Members	Description
USB_DEVICE_STATE_DETACHED	Device is not in any of the known USB states
USB_DEVICE_STATE_ATTACHED	Device is in attached state
USB_DEVICE_STATE_POWERED	Device is in powered state
USB_DEVICE_STATE_DEFAULT	Device is in default state
USB_DEVICE_STATE_ADDRESSED	Device is in addressed state
USB_DEVICE_STATE_CONFIGURED	Device is in configured state

USB_DEVICE_STATE_SUSPENDED	Device is in suspended state

Remarks

The USB specification doesn't define the state of a device when it is detached from the USB. The USB_DEVICE_STATE_DETACHED is not the standard state, but is required to indicate the user, that the device is not in any of the known states.

5.9.3.6.4.26 USB_DEVICE_STRING_DESCS_PTR Type**C**

```
typedef uint8_t** USB_DEVICE_STRING_DESCS_PTR;
```

Description

String descriptors pointer

5.9.3.6.4.27 USB_MASTER_DESCRIPTOR Structure**C**

```
typedef struct {
    uint8_t* ptrDeviceDescriptor;
    uint8_t configDescriptorCount;
    USB_DEVICE_CONFIG_DESCS_PTR ptrConfigDescriptor;
    uint8_t* ptrHighSpeedDeviceDescriptor;
    uint8_t highSpeedConfigDescriptorCount;
    USB_DEVICE_CONFIG_DESCS_PTR ptrHighSpeedConfigDescriptor;
    uint8_t stringDescCount;
    USB_DEVICE_STRING_DESCS_PTR ptrStringDesc;
    uint8_t* ptrFullSpeedDeviceQualifier;
    uint8_t* ptrHighSpeedDeviceQualifier;
} USB_MASTER_DESCRIPTOR;
```

Description

Global USB Descriptor Structure.

Members

Members	Description
uint8_t* ptrDeviceDescriptor;	Pointer to standard device descriptor (for low/full speed)
uint8_t configDescriptorCount;	Total number configurations available (for low/full speed)
USB_DEVICE_CONFIG_DESCS_PTR ptrConfigDescriptor;	Pointer to array of configurations descriptor pointers (for low/full speed)
uint8_t* ptrHighSpeedDeviceDescriptor;	Pointer to array of high speed standard Device descriptor. Assign this to NULL if not supported.
uint8_t highSpeedConfigDescriptorCount;	Total number of high speed configurations available. Set this to zero if not supported
USB_DEVICE_CONFIG_DESCS_PTR ptrHighSpeedConfigDescriptor;	Pointer to array of high speed configurations descriptor pointers. Set this to NULL if not supported
uint8_t stringDescCount;	Total number of string descriptors available (common to all speeds)
USB_DEVICE_STRING_DESCS_PTR ptrStringDesc;	Pointer to array of string Descriptor pointers (common to all speeds)
uint8_t* ptrFullSpeedDeviceQualifier;	Pointer to full speed device_qualifier descriptor. Device responds with this descriptor when it is operating at high speed
uint8_t* ptrHighSpeedDeviceQualifier;	Pointer to high speed device_qualifier descriptor. Device responds with this descriptor when it is operating at full speed

5.9.3.6.4.28 USB_DEVICE_ENDPOINT_TABLE_SIZE Macro

C

```
#define USB_DEVICE_ENDPOINT_TABLE_SIZE (DRV_USB_ENDPOINTS_NUMBER *  
DRV_USB_ENDPOINT_TABLE_ENTRY_SIZE)
```

Description

This is macro USB_DEVICE_ENDPOINT_TABLE_SIZE.

5.9.3.6.4.29 USB_DEVICE_HANDLE_INVALID Macro

C

```
#define USB_DEVICE_HANDLE_INVALID ((USB_DEVICE_HANDLE)(-1))
```

Description

USB Device Layer invalid handle

5.9.3.7 Files

Files

Name	Description
usb_device.h	USB Device Layer Interface Header
usb_device_config_template.h	USB device configuration template header file.

Description

5.9.3.7.1 usb_device.h

USB Device Layer Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB device layer.

Enumerations

	Name	Description
	USB_DEVICE_CONTROL_STATUS	This enumerated data-type identifies the status stage of control transfer.
	USB_DEVICE_CONTROL_TRANSFER_EVENT	This datatype defines the different control transfer events.
	USB_DEVICE_CONTROL_TRANSFER_RESULT	Enumerated data type identifying results of a control transfer.
	USB_DEVICE_EVENT	Enumerated data-type identifying the bus events that has occurred in the USB device layer.
	USB_DEVICE_POWER_STATE	Enumerated data type that identifies if the device is self powered or bus powered .
	USB_DEVICE_REMOTE_WAKEUP_STATUS	Enumerated data type that identifies if the remote wakeup status of the device.
	USB_DEVICE_STATE	Standard USB device states as described in Chapter-9 of USB 2.0 Specification.

Functions

	Name	Description
	<u>USB_DEVICE_Attach</u>	This function will attach the device to the USB.
	<u>USB_DEVICE_ClientStatus</u>	Gets the current client-specific status of the USB device layer.
	<u>USB_DEVICE_Close</u>	Closes an opened handle to an instance of the USB device layer.
	<u>USB_DEVICE_ControlEventCallBackSet</u>	Client can register its call-back function into the device layer to get control transfer events.
	<u>USB_DEVICE_ControlReceive</u>	Receives data stage of the control transfer from host to device.
	<u>USB_DEVICE_ControlSend</u>	Sends data stage of the control transfer from device to host.
	<u>USB_DEVICE_ControlStatus</u>	Initiates status stage of the control transfer.
	<u>USB_DEVICE_Deinitialize</u>	Deinitializes the specified instance of the USB device layer.
	<u>USB_DEVICE_Detach</u>	This function will detach the device from the USB.
	<u>USB_DEVICE_EndpointDisable</u>	This function disables an endpoint.
	<u>USB_DEVICE_EndpointEnable</u>	This function enables a endpoint for the specified direction and endpoint size.
	<u>USB_DEVICE_EndpointIsEnabled</u>	This function returns the enable/ disable status of the specified endpoint and direction.
	<u>USB_DEVICE_EndpointIsStalled</u>	This function returns the stall status of the specified endpoint and direction.
	<u>USB_DEVICE_EndpointStall</u>	This function stalls an endpoint in the specified direction.
	<u>USB_DEVICE_EndpointStallClear</u>	This function clears the stall on an endpoint in the specified direction.
	<u>USB_DEVICE_EventCallBackSet</u>	Client can register its call-back function into the device layer.
	<u>USB_DEVICE_GetConfigurationValue</u>	Informs the client of the current USB device configuration set by the USB host.
	<u>USB_DEVICE_GetDeviceSpeed</u>	Informs the client of the current operation speed of the USB bus.
	<u>USB_DEVICE_GetDeviceState</u>	Returns the current state of the USB device.
	<u>USB_DEVICE_Initialize</u>	Creates and initializes an instance of the USB device layer.
	<u>USB_DEVICE_IRPCancelAll</u>	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	<u>USB_DEVICE_IRPSubmit</u>	This function submits a I/O Request Packet (IRP) for processing to the USB Driver.
	<u>USB_DEVICE_Open</u>	Opens the specified USB device layer instance and returns a handle to it.
	<u>USB_DEVICE_PowerStateSet</u>	Sets power state of the device.
	<u>USB_DEVICE_Reinitialize</u>	Reinitializes the USB device layer
	<u>USB_DEVICE_RemoteWakeupIsEnabled</u>	Gets the "Remote wakeup" status of the device.
	<u>USB_DEVICE_ResumeStart</u>	This function will start the resume signalling.
	<u>USB_DEVICE_ResumeStop</u>	This function will stop the resume signalling.
	<u>USB_DEVICE_Status</u>	Provides the current status of the USB device layer
	<u>USB_DEVICE_Tasks</u>	USB Device layer calls all other function driver tasks in this function. It also generates and forwards events to its clients.

Macros

	Name	Description
	<u>USB_DEVICE_ENDPOINT_TABLE_SIZE</u>	This is macro USB_DEVICE_ENDPOINT_TABLE_SIZE.
	<u>USB_DEVICE_HANDLE_INVALID</u>	Macro that defines the value of invalid device handle.
	<u>USB_DEVICE_INDEX_0</u>	USB device layer index definitions.

	USB_DEVICE_INDEX_1	This is macro USB_DEVICE_INDEX_1.
	USB_DEVICE_INDEX_2	This is macro USB_DEVICE_INDEX_2.
	USB_DEVICE_INDEX_3	This is macro USB_DEVICE_INDEX_3.
	USB_DEVICE_INDEX_4	This is macro USB_DEVICE_INDEX_4.
	USB_DEVICE_INDEX_5	This is macro USB_DEVICE_INDEX_5.
	USB_DEVICE_INDEX_COUNT	Number of valid USB Device Layer indices

Structures

	Name	Description
	USB_DEVICE_EVENT_DATA_CONFIGURED	Data-type that holds the data related to USB device configured event.
	USB_DEVICE_FUNC_REGISTRATION_TABLE	A function driver has to be registered with the USB device layer using this structure.
	USB_DEVICE_FUNCTION_DRIVER	A function driver has to expose standard APIs to device layer using following structure.
	USB_DEVICE_INIT	This structure has to be initialized by the system/application and must be passed as parameter to USB_DEVICE_Initialize() .
	USB_MASTER_DESCRIPTOR	Global USB descriptor structure containing pointers to standard USB descriptor structures.

Types

	Name	Description
	USB_DEVICE_CALLBACK	Pointer to a USB Device Layer callback function data type for bus events.
	USB_DEVICE_CONFIG_DESCS_PTR	Pointer to an array that contains pointer to configuration descriptors.
	USB_DEVICE_CONTROL_TRANSFER_CALLBACK	
	USB_DEVICE_CONTROL_TRANSFER_HANDLE	Data type of USB device control transfer handle.
	USB_DEVICE_HANDLE	Data type for USB device handle.
	USB_DEVICE_STRING_DESCS_PTR	Pointer to an array that contains pointer to string descriptors.

Unions

	Name	Description
	USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA	USB device control transfer event data.
	USB_DEVICE_EVENT_DATA	Data associated with USB bus events.

File Name

usb_device.h

Company

Microchip Technology Inc.

5.9.3.7.2 usb_device_config_template.h

USB Device Layer Compile Time Options

This file contains USB device layer compile time options (macros) that are to be configured by the user. This file is a template file and must be used as an example only. This file must not be directly included in the project.

Macros

	Name	Description
	<u>USB_DEVICE_MAX_CLIENTS</u>	Sets the maximum possible number of clients an instance of the USB device can open using <u>USB_DEVICE_Open</u> .
	<u>USB_DEVICE_MAX_FUNCTION_DRIVER</u>	Sets the maximum number of function drivers at a time that are supported by an instance of the USB device layer.
	<u>USB_DEVICE_MAX_INSTANCES</u>	Sets the maximum possible number of instances of the USB device that can be instantiated by using <u>USB_DEVICE_Initialize()</u> routine.

File Name

usb_device_config_template.h

Company

Microchip Technology Inc.

5.9.4 USB Device CDC Library

5.9.4.1 Introduction

Introduces the MPLAB Harmony USB Device CDC Library

Description

The MPLAB Harmony USB Device CDC library provides a high-level abstraction of the Universal Serial Bus (USB) Communications Device Class (CDC). This function driver is a part of the MPLAB Harmony USB Device stack. The library offers a convenient C language interface and supports revision 1.2 of the USB CDC specification release.

The USB Communications Device Class (or USB CDC) is a composite Universal Serial Bus device class. The communications device class is primarily used for

- Telecommunication Devices (analog modems, ISDN terminal adapters, digital telephones, and analog phones)
- Networking Devices (ADSL modems, cable modems and Ethernet cross-over cables).

USB CDC specification, and associated subclass specifications, do not attempt to redefine existing standards for connection and control of communications services. The Communications Class defines mechanisms for a device and host to identify which existing protocols to use. Where possible, existing data formats are used and the transport of these formats are merely enabled by the USB through the definition of the appropriate descriptors, interfaces, and requests.

The MPLAB Harmony USB Device CDC library (also referred to as CDC function driver in this document) offers services to the application to interact and respond to the host requests.

5.9.4.2 Release Notes

MPLAB Harmony Version: v0.70b **USB Device CDC Library Version :** 0.01 Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.9.4.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.9.4.4 Library Architecture

5.9.4.4.1 Abstraction Model

Provides an Architectural Overview of the CDC Function Driver.

Description

The CDC Function Driver offers services to a USB CDC device to communicate with the host by abstracting the USB specification details. It must be used along with the USB Device layer and USB controller to communicate with the USB host. Figure 1 shows a block diagram representation of the MPLAB Harmony USB Architecture and where the USB CDC Function Driver is placed.

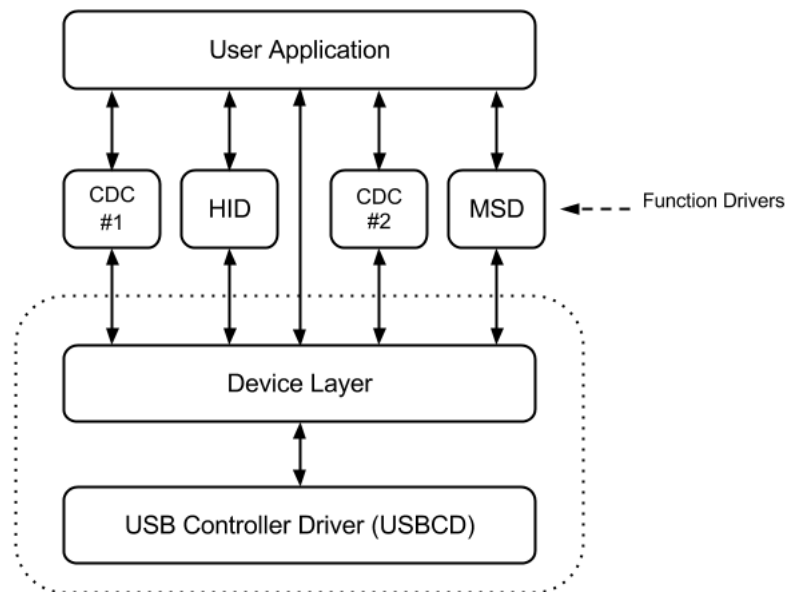


Figure 1: USB CDC Function Driver

The USB controller driver takes the responsibility of managing the USB peripheral on the device. The USB device layer handles the device enumeration etc. The USB Device layer forwards all CDC specific control transfers to the CDC Function Driver. The CDC Function Driver ACM sub-layer interprets the control transfers and requests application's intervention through event handlers and well defined set of API. The application must register a event handler with the CDC function driver in the Device Layer Set Configuration Event. While the application must respond to the CDC ACM events, it can do this either in the event handler or out of it. The application interacts directly with the CDC Function Driver to [send](#)/receive data and to [send](#) serial state notifications.

As per the CDC specification, a USB CDC device is a collection of following interfaces.

- Communication Interface (Device Management) on endpoint 0
- Communication Interface (Notification) on an interrupt endpoint. This is optional
- Data Interface (either a bulk or isochronous endpoint). This is optional.

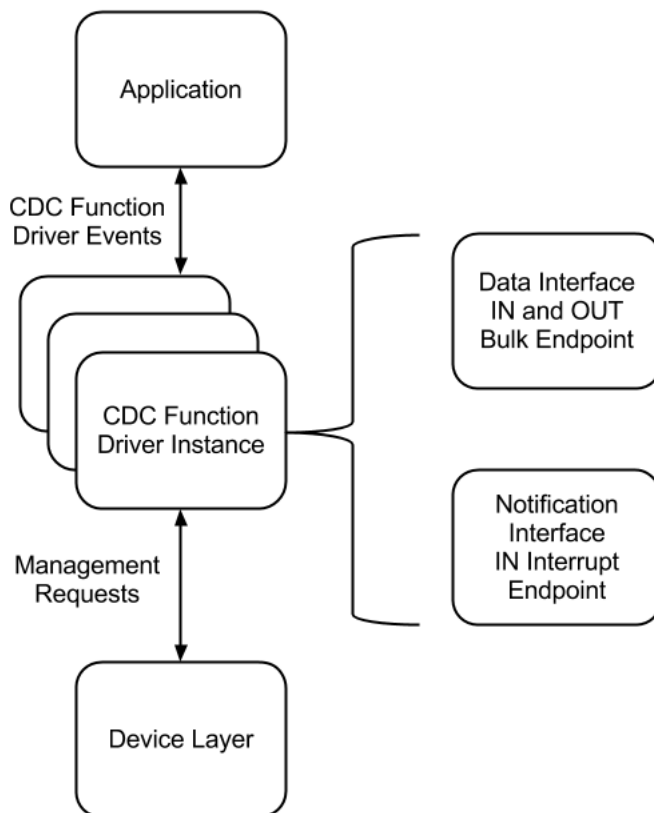


Figure 2: CDC Function Driver Architecture

Figure 2 shows the architecture of the CDC Function Driver. The device management on end point zero is handled by the device library(class specific requests are routed to the CDC function driver by USB Device Layer). So an instance of CDC function driver actually consists of a data interface and a notification interface. The library is implemented in two .c files. The `usb_device_cdc.c` file implements the CDC data and serial state notification. The `usb_device_cdc_acm.c` file implements the control transfer interpretation and and event generation.

5.9.4.4.2 Abstract Control Model (ACM)

Describes the various ACM commands supported by this CDC Function Driver Implementation

Description

One of the basic supported models for communication by CDC is POTS (Plain Old Telephone Service). The POTS model is for devices that communicate via ordinary phone lines and generic COM port devices. The USB CDC specification refers to this basic model as PSTN (Public Switched Telephone Network).

Depending on the amount of data processing the device is responsible for POTS/PSTN is divided into several models. The processing of data can include modulation, demodulation, error correction and data compression.

Of the supported PSTN models, this CDC function driver implements Abstract Control Model (ACM). In the ACM the device handles modulation, demodulation and handles V.25ter (AT) commands. This model (ACM) also supports requests and notifications to get and set R-232 status, control and asynchronous port part parameters. Virtual COM port devices use Abstract Control Model.

The following sections describe the management requests and notifications supported by the CDC ACM function driver.

Management Requests:

The host requests/sends some information in the form of management requests on the bi-directional end-point 0. Table 1 shows the CDC specification ACM sub class management requests and how these request are handled by the CDC Function Driver.

Request Code	Required/Optional	Comments
SEND_ENCAPSULATED_COMMAND	Required	Implemented by CDC Function Driver ACM layer. This request is stalled.
GET_ENCAPSULATED_RESPONSE	Required	Implemented by CDC Function Driver ACM layer. This request is stalled.
SET_COMM_FEATURE	Optional	Not Implemented
GET_COMM_FEATURE	Optional	Not Implemented
CLEAR_COMM_FEATURE	Optional	Not Implemented
SET_LINE_CODING	Optional	Implemented by CDC Function Driver ACM layer. Requires application response
GET_LINE_CODING	Optional	Implemented by CDC Function Driver ACM layer. Requires application response
SET_CONTROL_LINE_STATE	Optional	Implemented by CDC Function Driver ACM layer. Requires application response
SEND_BREAK	Optional	Implemented by CDC Function Driver ACM layer. Requires application response

Table 1: CDC ACM management requests and CDC Function Driver support status.

5.9.4.5 Using the Library

5.9.4.5.1 Files to Include

Describes which files to include in project while using the CDC Function Driver

Description

Table 2 shows the files that must be included in the project in order to use the CDC Function Driver. These files are located in the framework folder of the MPLAB Harmony installation. It is assumed that the Device Layer files (and the files needed by the Device Layer) are already included in the project.

Filename	Description
\\framework\\usb\\src\\dynamic\\usb_device_cdc.c	Contains the CDC data and serial notification transfer functions implementation.
\\framework\\usb\\src\\dynamic\\usb_device_cdc_acm.c	Implements the control transfer interpretation and and event generation of CDC-ACM events.
\\framework\\usb\\ usb_device_cdc.h	Must included in every source file that needs to invoke CDC function driver API.
system_config.h	User created file which contains the CDC function driver configuration.

Table 2: Files to be included in USB CDC Device Project.

5.9.4.5.2 Library Configuration

Describes how to configure the CDC Function Driver

Description

The application designer must specify the following configuration parameters while using the CDC Function Driver. The configuration macros that implement these parameters must be located in the system_config.h file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

Configuration Macro Name	Description	Comments
USB_DEVICE_CDC_INSTANCES_NUMBER	Number of CDC Function Driver Instances in the application	This macro defines the number of instances of the CDC Function Driver. For example, if the application needs to implement two instances of the CDC Function Driver (to create two COM ports) on one USB Device, the macro should be set to 2. Note that implementing a USB Device that features multiple CDC interfaces requires appropriate USB configuration descriptors.
USB_DEVICE_CDC_QUEUE_SIZE	CDC Function Driver Buffer Combined Queue size	This macro defines the number of entries in all queues in all instances of the CDC function driver. This value can be obtained by adding up the read and write queue sizes of each CDC Function driver instance. In a simple single instance USB CDC device application, that does not require buffer queuing and serial state notification, the USB_DEVICE_CDC_QUEUE_SIZE macro can be set to 2. Consider a case with two CDC function driver instances, CDC 1 has a read queue size of 2 and write queue size of 3, CDC 2 has a read queue size of 4 and write queue size of 1, this macro should be set to 10 (2 + 3 + 4 + 1).

5.9.4.5.3 Library Initialization

Describes how the CDC Function Driver is initialized.

Description

The CDC function driver instance for a USB device configuration is initialized by the Device Layer when the configuration is set by the host. This process does not require application intervention. Each instance of the CDC function driver should be registered with the Device layer through the Device Layer Function Driver Registration Table. The CDC function driver does not require a initialization data structure. The funcDriverInit member of the function driver registration table entry for the CDC function driver instance should be set to NULL. The cdcFuncDriver object is a global object provided by the CDC function driver and points to the CDC function driver - Device Layer interface functions which are required by the Device Layer. The code snippet below shows an example of how multiple instances of CDC Function driver can be registered with the Device Layer.

```

/* This code snippet shows an example of how two CDC function
 * driver instances can be registered with the Device Layer
 * via the Device Layer Function Driver Registration Table.
 * In this case Device Configuration 1 consists of two CDC
 * function driver instances. */

/* Define the CDC initialization data structure for CDC instance 0.
 * Set read queue size to 2 and write queue size to 3 */

const USB_DEVICE_CDC_INIT cdcInit0 = {.queueSizeRead = 2, .queueSizeWrite = 3};

```

```

/* Define the CDC initialization data structure for CDC instance 1.
 * Set read queue size to 4 and write queue size to 1 */

const USB_DEVICE_CDC_INIT cdcInit1 = {.queueSizeRead = 4, .queueSizeWrite = 1};

const USB_DEVICE_FUNC_REGISTRATION_TABLE funcRegistrationTable[2] =
{
    /* This is the first instance of the CDC Function Driver */
    {
        .speed = USB_SPEED_FULL|USB_SPEED_HIGH, // Supported speed
        .configurationValue = 1, // To be initialized for Configuration 1
        .interfaceNumber = 0, // Starting interface number.
        .numberOfInterfaces = 2, // Number of interfaces in this instance
        .funcDriverIndex = 0, // Function Driver instance index is 0
        .funcDriverInit = &cdcInit0, // Function Driver initialization data structure
        .driver = &cdcFuncDriver // Pointer to Function Driver - Device Layer interface
    },
    /* This is the second instance of the CDC Function Driver */
    {
        .speed = USB_SPEED_FULL|USB_SPEED_HIGH, // Supported speed
        .configurationValue = 1, // To be initialized for Configuration 1
        .interfaceNumber = 2, // Starting interface number.
        .numberOfInterfaces = 2, // Number of interfaces in this instance
        .funcDriverIndex = 1, // Function Driver instance index is 1
        .funcDriverInit = &cdcInit1, // Function Driver initialization data structure
        .driver = &cdcFuncDriver // Pointer to Function Driver - Device Layer interface
    },
};

```

5.9.4.5.4 Event Handling

Describes CDC function driver event handler registration and event handling.

Description

Registering a CDC Function Driver Event Handler:

While creating a USB CDC Device based application, an event handler must be registered with the Device Layer (the Device Layer Event Handler) and every CDC function driver instance (CDC Function Driver Event Handler). The CDC function driver event handler receives CDC and CDC ACM events. This event handler should be registered before the USB device layer acknowledges the SET CONFIGURATION request from the USB Host. In order to ensure this, the event handler should be set in the USB_DEVICE_EVENT_CONFIGURED event that is generated by the device layer. While registering the CDC function driver event handler, the CDC function driver allows the application to also pass a application context object. This object gets associated with the instance of the CDC function driver and is returned by the CDC function driver when a CDC function driver event occurs. The code example below shows an example of how this can be done.

```

/* This a sample Application Device Layer Event Handler
 * Note how the CDC Function Driver event handler APP_USBDeviceCDCEventHandler()
 * is registered in the USB_DEVICE_EVENT_CONFIGURED event. The appData
 * object that is passed in the USB_DEVICE_CDC_EventHandlerSet()
 * function will be returned as the userData parameter in the
 * when the APP_USBDeviceCDCEventHandler() function is invoked */

void APP_USBDeviceEventCallBack ( USB_DEVICE_EVENT event,
                                USB_DEVICE_EVENT_DATA * eventData )
{
    switch ( event )
    {
        case USB_DEVICE_EVENT_RESET:
        case USB_DEVICE_EVENT_DECONFIGURED:

```

```

    // USB device is reset or device is deconfigured.
    // This means that USB device layer is about to deinitalize
    // all function drivers.

    break;

case USB_DEVICE_EVENT_CONFIGURED:

    /* check the configuration */
    if ( eventData->eventConfigured.configurationValue == 1)
    {

        /* Register the CDC Device application event handler here.
        * Note how the appData object pointer is passed as the
        * user data */

        USB_DEVICE_CDC_EventHandlerSet(USB_DEVICE_CDC_INDEX_0,
            APP_USBDeviceCDCEventHandler, (uintptr_t)&appData);

        /* mark that set configuration is complete */
        appData.isConfigured = true;

    }
    break;

case USB_DEVICE_EVENT_SUSPENDED:

    break;

case USB_DEVICE_EVENT_RESUMED:
case USB_DEVICE_EVENT_ATTACHED:
case USB_DEVICE_EVENT_DETACHED:
case USB_DEVICE_EVENT_ERROR:
default:
    break;
}
}

```

CDC Function Driver Events:

The CDC Function Driver generates events that the application must respond. Some of these events are management requests communicated through control transfers. The application must therefore complete the control transfer. Based on the generated event, the application may be required to:

- Respond with a [USB_DEVICE_CDC_ControlSend\(\)](#) function which completes the data stage of a Control Read Transfer.
- Respond with a [USB_DEVICE_CDC_ControlReceive\(\)](#) function which provisions the data stage of a Control Write Transfer.
- Respond with a [USB_DEVICE_CDC_ControlStatus\(\)](#) function which completes the handshake stage of the Control Transfer. The application can either STALL or Acknowledge the handshake stage via the [USB_DEVICE_CDC_ControlStatus\(\)](#) function.
- Analyze the pData member of the event handler and check for event specific data. This data member should be type cast to an event specific data type. The table below shows the event and the data type to use while type casting. Note that the pData member is not required for all events

CDC Function Driver Event	Related pData type
USB_DEVICE_CDC_EVENT_SET_LINE_CODING	Not Applicable
USB_DEVICE_CDC_EVENT_GET_LINE_CODING	Not Applicable
USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE	USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE *

USB_DEVICE_CDC_EVENT_SEND_BREAK	USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK *
USB_DEVICE_CDC_EVENT_WRITE_COMPLETE	USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE *
USB_DEVICE_CDC_EVENT_READ_COMPLETE	USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE *
USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE	USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE *
USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT	Not Applicable
USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	Not Applicable

The possible CDC Function Driver events are described here along with the required application response, event specific data and likely follow up function driver event:

1. USB_DEVICE_CDC_EVENT_SET_LINE_CODING

Application Response: This event occurs when the host issues a SET LINE CODING command. The application must provide a [USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING](#) data structure to the device layer to receive the line coding data that the host will provide. The application must provide the buffer by calling the [USB_DEVICE_CDC_ControlReceive\(\)](#) function either in the event handler or in the application (outside of the event handler function). The application must use the controlTransferHandle parameter provided in the event while calling [USB_DEVICE_CDC_ControlReceive\(\)](#) function. The application can use the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (pData): The pData parameter will be NULL.

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. This indicates that the data was received successfully. The application must either acknowledge or stall the handshake stage of the control transfer by calling [USB_DEVICE_ControlStatus\(\)](#) function with USB_DEVICE_CONTROL_STATUS_OK or USB_DEVICE_CONTROL_STATUS_ERROR flag respectively.

2. USB_DEVICE_CDC_EVENT_GET_LINE_CODING

Application Response: This event occurs when the host issues a GET LINE CODING command. The application must provide a [USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING](#) data structure to the device layer that contains the line coding data that to be provided to the host. The application must provide the buffer by calling the [USB_DEVICE_CDC_ControlSend\(\)](#) function either in the event handler or in the application (outside of the event handler function). The application must use the controlTransferHandle parameter provided in the event while calling [USB_DEVICE_CDC_ControlSend\(\)](#) function. The size of the buffer is indicated by the length parameter. The application can use the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (pData): The pData parameter will be NULL.

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT event. This indicates that the data was sent to the host successfully. The application must acknowledge the handshake stage of the control transfer by calling [USB_DEVICE_CDC_ControlStatus\(\)](#) function with USB_DEVICE_CONTROL_STATUS_OK flag.

3. USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE

Application Response: This event occurs when the host issues a SET CONTROL LINE STATE command. . The application can then use the [USB_DEVICE_CDC_ControlStatus\(\)](#) function to indicate acceptance or rejection of the command. The [USB_DEVICE_CDC_ControlStatus\(\)](#) function can be called from the event handler or in the application (out of the event handler context). The application should use the controlTransferHandle while calling the [USB_DEVICE_ControlStatus\(\)](#) function.

Event Specific Data (pData): The application must interpret the pData parameter as a pointer to a [USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE](#) data type that contains the control line state data.

Likely Follow-up event: None.

4. USB_DEVICE_CDC_EVENT_SEND_BREAK

Application Response: This event occurs when the host issues a SEND BREAK command. The application can then use the [USB_DEVICE_CDC_ControlStatus\(\)](#) function to indicate acceptance or rejection of the command. The [USB_DEVICE_CDC_ControlStatus\(\)](#) function can be called from the event handler or in the application (out of the event handler context). The application should use the controlTransferHandle while calling the [USB_DEVICE_CDC_ControlStatus\(\)](#) function.

Event Specific Data (pData): The application must interpret the pData parameter as a pointer to a [USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK](#) data type that contains the break duration data.

Likely Follow-up event: None.

5. USB_DEVICE_CDC_EVENT_WRITE_COMPLETE

Application Response: This event occurs when a write operation scheduled by calling the [USB_DEVICE_CDC_Write\(\)](#) function has completed. This event does not require the application to respond with any function calls..

Event Specific Data (pData): The pData member in the event handler will point to [USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE](#) type.

Likely Follow-up event: None.

6. USB_DEVICE_CDC_EVENT_READ_COMPLETE

Application Response: This event occurs when a read operation scheduled by calling the [USB_DEVICE_CDC_Read\(\)](#) function has completed. This event does not require the application to respond with any function calls..

Event Specific Data (pData): The pData member in the event handler will point to [USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE](#) type.

Likely Follow-up event: None.

7. USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE

Application Response: This event occurs when a serial state notification [send](#) scheduled by calling the [USB_DEVICE_CDC_SerialStateNotificationSend\(\)](#) function has completed. This event does not require the application to respond with any function calls..

Event Specific Data (pData): The pData member in the event handler will point to [USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE](#) type.

Likely Follow-up event: None.

8. USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT

Application Response: This event occurs when the data stage of a control read transfer has completed in response to the [USB_DEVICE_CDC_ControlSend\(\)](#) function (in the USB_DEVICE_CDC_EVENT_GET_LINE_CODING event). The application must acknowledge the handshake stage of the control transfer by calling [USB_DEVICE_CDC_ControlStatus\(\)](#) function with USB_DEVICE_CONTROL_STATUS_OK flag.

Event Specific Data (pData): The pData parameter will be NULL.

Likely Follow-up event: None.

9. USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED

Application Response: This event occurs when the data stage of a control write transfer has completed in response to the `USB_DEVICE_CDC_ControlReceive()` function (in the `USB_DEVICE_CDC_EVENT_SET_LINE_CODING` event). The application must either acknowledge or stall the handshake stage of the control transfer by calling `USB_DEVICE_CDC_ControlStatus()` function with `USB_DEVICE_CONTROL_STATUS_OK` or `USB_DEVICE_CONTROL_STATUS_ERROR` flag respectively.

Event Specific Data (pData): The pData parameter will be NULL.

Likely Follow-up event: None.

CDC Function Driver Event Handling:

The following code snippet shows an example event handling scheme. The application always returns from the event handler with a `USB_DEVICE_CDC_EVENT_RESPONSE_NONE` value.

```
// This code example shows all CDC Function Driver possible events
// and a possible scheme for handling these events. In this case
// event responses are not deferred.

USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING_DATA setLineCodingData;
USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING_DATA getLineCodingData;
USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_DATA controlLineStateData;
USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK_DATA breakData;

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    uintptr_t controlTransferHandle,
    void * data,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_SET_LINE_CODING:
            // In this case, the application should read the line coding
            // data that is sent by the host.

            USB_DEVICE_CDC_ControlReceive(usbDeviceHandle, controlTransferHandle
                                           &setLineCodingData,
                                           sizeof(USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING));

            break;

        case USB_DEVICE_CDC_EVENT_GET_LINE_CODING:
            // In this case, the application should send the line coding
            // data to the host.

            USB_DEVICE_CDC_ControlSend(usbDeviceHandle, controlTransferHandle
                                       &getLineCodingData,
                                       sizeof(USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING));

            break;

        case USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE:
            // In this case, pData will point to a
            // USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE type data object that
            // will contain the control line state data sent by the host.
```

```

        controllLineStateData.dtr =
            ((USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE *)pData)->dtr;
        controllLineStateData.carrier =
            ((USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE *)pData)->carrier;

        USB_DEVICE_CDC_ControlStatus(usbDeviceHandle, controlTransferHandle,
                                     USB_DEVICE_CONTROL_STATUS_OK);

        break;

    case USB_DEVICE_CDC_EVENT_SEND_BREAK:

        // In this case, pData will point to a
        // USB_DEVICE_CDC_EVENT_SEND_BREAK_DATA type data object that
        // will contain the break duration sent by the host.

        breakData.duration = ((USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK *)pData)->duration;

        USB_DEVICE_CDC_ControlStatus(usbDeviceHandle, controlTransferHandle,
                                     USB_DEVICE_CONTROL_STATUS_OK);

        break;

    case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT:
    case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:

        // This means that the data stage is complete. The data in
        // setLineCodingData is valid or data in getLineCodingData was
        // sent to the host. The application can now decide whether it
        // supports this data. It is not mandatory to do this in the
        // event handler.

        USB_DEVICE_ControlStatus(usbDeviceHandle, controlTransferHandle,
                                 USB_DEVICE_CONTROL_STATUS_OK);

    case USB_DEVICE_CDC_EVENT_WRITE_COMPLETE:

        // This means USB_DEVICE_CDC_Write() operation completed.
        // The pData member will point to a
        // USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE type of data.

        break;

    case USB_DEVICE_CDC_EVENT_READ_COMPLETE:

        // This means USB_DEVICE_CDC_Read() operation completed.
        // The pData member will point to a
        // USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE type of data.

        break;

    case USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE:

        // This means USB_DEVICE_CDC_SerialStateNotification() operation
        // completed. The pData member will point to a
        // USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE type of data.

        break;

    default:
        break;
}

return(USB_DEVICE_CDC_EVENT_RESPONSE_NONE);
}

```

Refer to the [USB_DEVICE_CDC_EVENT](#) enumeration for more details on each event.

5.9.4.5.5 Sending Data

Describes how to [send](#) data to the CDC Host.

Description

The application may need to [send](#) data or serial state notification to the USB CDC Host. This is done by using the [USB_DEVICE_CDC_Write\(\)](#) and [USB_DEVICE_CDC_SerialStateNotificationSend\(\)](#) functions respectively.

1. Sending data to the USB Host:

The application can [send](#) data to the host by using the [USB_DEVICE_CDC_Write\(\)](#) function. This function returns a transfer handler that allows the application to track the read request. The request is completed when the host as requested for the data. The completion of the write transfer is indicated by USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event. A write request could fail if the function driver instance transfer queue is full.

The [USB_DEVICE_CDC_Write\(\)](#) function also allows the application to [send](#) data to the host without ending the transfer. This is done by specifying the USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_PENDING flag. The application can use this option when the data to be sent is not readily available or when the application is memory constrained. The combination of the transfer flag and the transfer size affects how the function driver sends the data to the host.

- a. If size is a multiple of maxPacketSize (the IN endpoint size) and flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function will append a Zero Length Packet (ZLP) to complete the transfer.
- b. If size is a multiple of maxPacketSize and flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function will not append a ZLP and hence will not complete the transfer.
- c. If size is greater than but not a multiple of maxPacketSize and flags is set as USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules (length/maxPacketSize) packets and one packet for the residual data.
- d. If size is greater than but not a multiple of maxPacketSize and flags is set as USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function schedules (size/maxPacketSize) packets only.
- e. If size is less than maxPacketSize and flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules one packet.
- f. If size is less than maxPacketSize and flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function returns an error code and sets the transferHandle parameter to [USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID](#).

The following code snippet explains this in more detail:

```
// Below is a set of examples showing various conditions trying to
// send data with the USB_DEVICE_CDC_Write() command.
//
// This assumes that driver was opened successfully.
// Assume maxPacketSize is 64.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_RESULT writeRequestHandle;
USB_DEVICE_CDC_INDEX instance;

//-----
// In this example we want to send 34 bytes only.

writeRequestResult = USB_DEVICE_CDC_Write(instance,
                                         &transferHandle, data, 34,
```



```

        USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//-----
// In this example we want to send 64 bytes only.
// This will cause a ZLP to be sent.

writeRequestResult = USB_DEVICE_CDC_Write(instance,
        &transferHandle, data, 64,
        USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//-----
// This example will return an error because size is less
// than maxPacketSize and the flag indicates that more
// data is pending.

writeRequestResult = USB_DEVICE_CDC_Write(instanceHandle,
        &transferHandle, data, 32,
        USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING);

//-----
// In this example we want to place a request for a 70 byte transfer.
// The 70 bytes will be sent out in a 64 byte transaction and a 6 byte
// transaction completing the transfer.

writeRequestResult = USB_DEVICE_CDC_Write(instanceHandle,
        &transferHandle, data, 70,
        USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//-----
// In this example we want to place a request for a 70 bytes to be sent
// but that we don't end the transfer as more data is coming. 64 bytes
// of the 70 will be sent out and the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE
// with 64 bytes. This indicates that the extra 6 bytes weren't
// sent because it would cause the end of the transfer. Thus the
// user needs add these 6 bytes back to the buffer for the next group
// of data that needs to be sent out.

writeRequestResult = USB_DEVICE_CDC_Write(instanceHandle,
        &transferHandle, data, 70,
        USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

// The completion of the write request will be indicated by the
// USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event.

```

2. Sending a serial state notification:

The application can **send** a Serial State Notification by using the `USB_DEVICE_CDC_SerialStateSend()` function. his function

returns a transfer handler that allows the application to track the read request. The request is completed when the host as requested for the data. The completion of the transfer is USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE event. A request could fail if the function driver transfer queue is full. The following code snippet shows an example of how this can be done.

```
USB_DEVICE_CDC_INDEX instanceIndex;
USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA notificationData;

// This application function could possibly update the notificationData
// data structure.

APP_UpdateNotificationData(&notificationData);

// Now send the updated notification data to the host.

result = USB_DEVICE_CDC_SerialStateDataSend(instanceIndex, &transferHandle,
                                              &notificationData);

if(USB_DEVICE_CDC_RESULT_OK != result)
{
    // Error handling here
}
```

5.9.4.5.6 Receiving Data

Describes how the CDC device can read data from the Host.

Description

The application can receive data from the host by using the [USB_DEVICE_CDC_Read\(\)](#) function. This function returns a transfer handler that allows the application to track the read request. The request is completed when the host sends the required amount or less than required amount of data. The application must make sure that it allocates a buffer size that is at least the size or a multiple of the receive endpoint size. The return value of the function indicates the success of the request. A read request could fail if the function driver transfer queue is full. The completion of the read transfer is USB_DEVICE_CDC_EVENT_READ_COMPLETE event. The request completes based on the amount of the data that was requested and size of the transaction initiated by the host.

- If the size parameter is not a multiple of maxPacketSize or is 0, the function returns [USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID](#) in transferHandle and returns USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_SIZE_INVALID as a return value.
- If the size parameter is a multiple of maxPacketSize and the host [send](#) less than maxPacketSize data in any transaction, the transfer completes and the function driver will issue a USB_DEVICE_CDC_EVENT_READ_COMPLETE event along with the USB_DEVICE_CDC_EVENT_READ_COMPLETE_DATA data structure.
- If the size parameter is a multiple of maxPacketSize and the host sends maxPacketSize amount of data, and total data received does not exceed size, then the function driver will wait for the next packet.

The following code snippet shows an example if the [USB_DEVICE_CDC_Read\(\)](#) function:

```
// Shows an example of how to read. This assumes that
// driver was opened successfully.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_RESULT readRequestResult;
USB_DEVICE_CDC_HANDLE instanceHandle;

readRequestResult = USB_DEVICE_CDC_Read(instanceHandle,
                                         &transferHandle, data, 128);

if(USB_DEVICE_CDC_RESULT_OK != readRequestResult)
```

```

{
    //Do Error handling here
}

// The completion of the read request will be indicated by the
// USB_DEVICE_CDC_EVENT_READ_COMPLETE event.

```

5.9.4.6 Library Interface

Data Types and Constants

	Name	Description
	<u>USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE</u>	USB Device CDC Function Driver Control Transfer Handle
	<u>USB_DEVICE_CDC_EVENT</u>	USB Device CDC Function Driver Control Transfer Status
	<u>USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING</u>	USB Device CDC Event Get Line Coding Data Type.
	<u>USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE</u>	USB Device CDC Function Driver Read and Write Complete Event Data.
	<u>USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK</u>	USB Device CDC Event Set Break Data Type.
	<u>USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE</u>	USB Device CDC Function Driver Read and Write Complete Event Data.
	<u>USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE</u>	USB Device CDC Event Set Line Coding Data Type.
	<u>USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING</u>	USB Device CDC Event Set Line Coding Data Type.
	<u>USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE</u>	USB Device CDC Function Driver Read and Write Complete Event Data.
	<u>USB_DEVICE_CDC_EVENT_HANDLER</u>	USB Device CDC Event Handler Function Pointer Type.
	<u>USB_DEVICE_CDC_EVENT_RESPONSE</u>	USB Device CDC Function Driver Event Callback Response Type
	<u>USB_DEVICE_CDC_INDEX</u>	USB Device CDC Function Driver Index
	<u>USB_DEVICE_CDC_INIT</u>	USB Device CDC Function Driver Initialization Data Structure
	<u>USB_DEVICE_CDC_RESULT</u>	USB Device CDC Function Driver USB Device CDC Result enumeration.
	<u>USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA</u>	USB Device Serial State Notification Data Type.
	<u>USB_DEVICE_CDC_TRANSFER_FLAGS</u>	USB Device CDC Function Driver Transfer Flags

	USB_DEVICE_CDC_TRANSFER_HANDLE	USB Device CDC Function Driver Transfer Handle Definition.
	USB_DEVICE_CDC_EVENT_RESPONSE_NONE	USB Device CDC Function Driver Event Handler Response Type None.
	USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID	USB Device CDC Function Driver Invalid Transfer Handle Definition.

Functions

	Name	Description
	USB_DEVICE_CDC_ControlReceive	This function allows the application to respond to the CDC function driver specific control transfer requests. It allows the application to receive data from the host in the data stage of the control transfer.
	USB_DEVICE_CDC_ControlSend	This function allows the application to respond to the CDC function driver specific control transfer requests. It allows the application to send data to the host in the data stage of the control transfer.
	USB_DEVICE_CDC_ControlStatus	This function allows the application to complete the status stage of the the CDC Function Driver specific control transfer request.
	USB_DEVICE_CDC_EventHandlerSet	This function registers a event handler for the specified CDC function driver instance.
	USB_DEVICE_CDC_Read	This function requests a data read from the USB Device CDC Function Driver Layer.
	USB_DEVICE_CDC_SerialStateNotificationSend	This function schedules a request to send serial state notification to the host.
	USB_DEVICE_CDC_Write	This function requests a data write to the USB Device CDC Function Driver Layer.

Description

5.9.4.6.1 Functions

5.9.4.6.1.1 USB_DEVICE_CDC_ControlReceive Function

C

```

USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_ControlReceive(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * data,
    size_t size
);

```

Description

This function allows the application to respond to the CDC function driver specific control transfer requests. It allows the application to receive data from the host in the data stage of the control transfer. For example, the [USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING](#) event requires a control transfer response. The function can be called in the CDC Function Driver event handler or can be called outside the event handler. Calling this function outside the event handler defers the response to the event. This allows the application to prepare the data buffer out of the event handler context, especially if the data buffer to receive the data is not readily available. Note however, that there are timing

considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device. The application must use the control transfer handle that was generated along with the event as the controlTransferHandle.

Preconditions

This function should only be called in response to a CDC function driver event that requires a control transfer response.

Parameters

Parameters	Description
instance	USB Device CDC Function Driver instance.
controlTransferHandle	Control Transfer handle for the control transfer.
data	Data buffer to receive the data stage of the control transfer.
size	size (in bytes) of the data to sent.

Returns

USB_DEVICE_CDC_RESULT_ERROR_CONTROL_TRANSFER_FAILED - The request was not successful because the USB host cancelled the control transfer.

USB_DEVICE_CDC_RESULT_OK - The request was successful.

Remarks

None.

Example

```
// -----
// In this example, the application responds to the SET LINE CODING
// with in the event handler. The application uses the
// USB_DEVICE_CDC_ControlReceive() function to do this.
// -----

USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING setLineCodingData;
USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle;

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_SET_LINE_CODING:

            // In this case, the application should receive the line coding
            // data from the host. The application must use the
            // USB_DEVICE_CDC_ControlReceive() function to receive the data.

            USB_DEVICE_CDC_ControlReceive(instanceIndex, controlTransferHandle
                                           &setLineCodingData,
                                           sizeof(USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING));

            break;
    }
}

// -----
// In this example, the application defers the response to the
// event. This is done by calling the USB_DEVICE_CDC_ControlReceive()
```

```

// function after the event handler exits.
// -----

USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING setLineCodingData;
USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE thisControlTransferHandle;
bool sendLineCodingToHost;

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_SET_LINE_CODING:

            // In this case, the application should send the line coding
            // data to the host. The application must responds to the
            // event outside the event handler because the data buffer is
            // not available.

            thisControlTransferHandle = controlTransferHandle;
            getLineCodingFromHost = true;

            break;
    }
}

if(getLineCodingFromHost)
{
    // The application allocates a buffer using an arbitrary function
    // Note that this happens after the event handler exits.

    setLineCoding = AllocateMemoryBuffer();

    USB_DEVICE_CDC_ControlReceive(instanceIndex, thisControlTransferHandle
                                &setLineCodingData,
                                sizeof(USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING));
}

```

5.9.4.6.1.2 USB_DEVICE_CDC_ControlSend Function

C

```

USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_ControlSend(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * data,
    size_t size
);

```

Description

This function allows the application to respond to the CDC function driver specific control transfer requests. It allows the application to **send** data to the host in the data stage of the control transfer. For example, the USB_DEVICE_CDC_EVENT_GET_LINE_CODING event requires a control transfer response. The function can be called in the CDC Function Driver event handler or can be called outside the event handler. Calling this function outside the event handler defers the response to the event. This allows the application to prepare the response data out of the event handler context, especially if the data is not readily available. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the

device. The application must use the control transfer handle that was generated along with the event as the controlTransferHandle.

Preconditions

This function should only be called in response to a CDC function driver event that requires a control transfer response.

Parameters

Parameters	Description
instance	USB Device CDC Function Driver instance.
controlTransferHandle	Control Transfer handle for the control transfer.
data	Data that represents the data stage of the control transfer.
size	size (in bytes) of the data to sent.

Returns

USB_DEVICE_CDC_RESULT_ERROR_CONTROL_TRANSFER_FAILED - The request was not successful because the USB host cancelled the control transfer.

USB_DEVICE_CDC_RESULT_OK - The request was successful.

Remarks

None.

Example

```
// -----
// In this example, the application responds to the GET LINE CODING
// with in the event handler. The application uses the
// USB_DEVICE_CDC_ControlSend() function to do this.
// -----

USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING getLineCodingData;
USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle;

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_GET_LINE_CODING:

            // In this case, the application should send the line coding
            // data to the host. The application must use the
            // USB_DEVICE_CDC_ControlSend() function to send the data.

            USB_DEVICE_CDC_ControlSend(instanceIndex, controlTransferHandle
                                         &getLineCodingData,
                                         sizeof(USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING));

            break;
    }
}

// -----
// In this example, the application defers the response to the
// event. This is done by calling the USB_DEVICE_CDC_ControlSend()
// function after the event handler exits.
// -----
```

```

USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING getLineCodingData;
USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE thisControlTransferHandle;
bool sendLineCodingToHost;

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_GET_LINE_CODING:

            // In this case, the application should send the line coding
            // data to the host. The application does not call
            // USB_DEVICE_CDC_ControlSend() function
            // in the event handler, because it does not have the line
            // coding data ready.

            thisControlTransferHandle = controlTransferHandle;
            sendLineCodingToHost = true;

            break;
    }
}

// This is outside the event handler.

if(sendLineCodingToHost)
{
    // The application fetches the line coding from the
    // an EEPROM.

    GetLineCodingFromEEPROM(&getLineCoding);

    USB_DEVICE_CDC_ControlSend(instanceIndex, thisControlTransferHandle
                                &getLineCodingData,
                                sizeof(USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING));
}

```

5.9.4.6.1.3 USB_DEVICE_CDC_ControlStatus Function

C

```

USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_ControlStatus(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    USB_DEVICE_CDC_CONTROL_STATUS status
);

```

Description

This function allows the application to complete the status stage of the the CDC Function Driver specific control transfer request. The application can either **accept** the data stage or stall it. Calling this function with status set to USB_DEVICE_CDC_CONTROL_STATUS_OK will acknowledge the status stage of the control transfer. The control transfer can be stalled by setting the status parameter to USB_DEVICE_CDC_CONTROL_STATUS_ERROR. The function can be called in the CDC Function Driver event handler or can be called outside the event handler. Calling this function outside the event handler defers the response to the event. This allows the application to analyze the event response outside the event handler. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device. The application must use the control transfer

handle that was generated along with the event as the controlTransferHandle. The application must be aware of events and associated control transfers that do or do not require data stages. Incorrect usage of the USB_DEVICE_CDC_ControlStatus() function could cause the device function to be non-compliant.

Preconditions

This function should only be called in response to a CDC function driver event that requires a control transfer response.

Parameters

Parameters	Description
instance	USB Device CDC Function Driver instance.
controlTransferHandle	Control Transfer handle for the control transfer.
status	Set to USB_DEVICE_CDC_CONTROL_STATUS_OK to acknowledge the control transfer. Set to USB_DEVICE_CDC_CONTROL_STATUS_ERROR to stall the transfer.

Returns

USB_DEVICE_CDC_RESULT_ERROR_CONTROL_TRANSFER_FAILED - The request was not successful because the USB host cancelled the control transfer.

USB_DEVICE_CDC_RESULT_OK - The request was successful.

Remarks

None.

Example

```
// -----
// In this code example, the application responds to the the
// SEND_BREAK event in the event handler.
// -----

USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK * sendBreakData;

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_SEND_BREAK:

            // The application can check if the break duration is supported.

            sendBreakData = (USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK *)pData;

            if(IsBreakDurationSupported(sendBreakData->duration))
            {
                USB_DEVICE_CDC_ControlStatus(instanceIndex,
                    controlTransferHandle, USB_DEVICE_CDC_CONTROL_STATUS_OK);
            }
            else
            {
                // If the break duration is not supported the application
                // can stall it.

                USB_DEVICE_CDC_ControlStatus(instanceIndex,
                    controlTransferHandle, USB_DEVICE_CDC_CONTROL_STATUS_ERROR);
            }
        }
    }
}
```

```

        }
        break;
    }
}

// -----
// In this example, the application defers the response to the
// SEND_BREAK event. The application responds to the event
// after the event handler exits.
// -----

USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE thisControlTransferHandle;
USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK * sendBreakData;
uint16_t breakDuration;
bool sendBreakEvent;

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_SEND_BREAK:

            // Get the break duration.

            sendBreakData = (USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK *)pData;
            breakDuration = sendBreakData->duration;
            sendBreakEvent = true;
            break;

    }
}

if(sendBreakData)
{
    // The application checks if the duration is supported
    // Note that this occurs after the event handler has exited.

    if(IsBreakDurationSupported(breakDuration))
    {
        USB_DEVICE_CDC_ControlStatus(instanceIndex,
            controlTransferHandle, USB_DEVICE_CDC_CONTROL_STATUS_OK);
    }
    else
    {
        // If the break duration is not supported the application
        // can stall it.

        USB_DEVICE_CDC_ControlStatus(instanceIndex,
            controlTransferHandle, USB_DEVICE_CDC_CONTROL_STATUS_ERROR);
    }
}
}

```

5.9.4.6.1.4 USB_DEVICE_CDC_EventHandlerSet Function

C

```

USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_EventHandlerSet(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT_HANDLER eventHandler,

```

```
    uintptr_t context
);
```

Description

This function registers a event handler for the specified CDC function driver instance. This function should be called by the client when it receives a SET CONFIGURATION event from the device layer. A event handler must be registered for function driver to respond to function driver specific commands. If the event handler is not registered, the device layer will stall function driver specific commands and the USB device may not function.

Preconditions

This function should be called when the function driver has been initialized as a result of a set configuration.

Parameters

Parameters	Description
instance	Instance of the CDC Function Driver.
eventHandler	A pointer to event handler function.
context	Application specific context that is returned in the event handler.

Returns

USB_DEVICE_CDC_RESULT_OK - The operation was successful
 USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
 USB_DEVICE_CDC_RESULT_ERROR_PARAMETER_INVALID - The eventHandler parameter is NULL

Remarks

None.

Example

```
// This code snippet shows an example registering an event handler. Here
// the application specifies the context parameter as a pointer to an
// application object (appObject) that should be associated with this
// instance of the CDC function driver.

USB_DEVICE_CDC_RESULT result;

USB_DEVICE_CDC_EVENT_RESPONSE APP_USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex ,
    USB_DEVICE_CDC_EVENT event ,
    USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE controlTransferHandle
    void* pData, uintptr_t context
)
{
    // Event Handling comes here

    switch(event)
    {
        ...
    }

    return(USB_DEVICE_CDC_EVENT_RESPONSE_NONE);
}

result = USB_DEVICE_CDC_EventHandlerSet ( 0 ,&APP_EventHandler, (uintptr_t) &appObject);
if(USB_DEVICE_CDC_RESULT_OK != result)
{
    SYS_ASSERT ( false , "invalid event handler function" );
}
```

5.9.4.6.1.5 USB_DEVICE_CDC_Read Function

C

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_Read(  
    USB_DEVICE_CDC_INDEX instanceIndex,  
    USB_DEVICE_CDC_TRANSFER_HANDLE * transferHandle,  
    void * data,  
    size_t size  
);
```

Description

This function requests a data read from the USB Device CDC Function Driver Layer. The function places a requests with driver, the request will get serviced as data is made available by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_CDC_EVENT_READ_COMPLETE event. The amount of data read and the transfer handle associated with the request is returned along with the event in the pData parameter of the event handler. The transfer handle expires when event handler for the USB_DEVICE_CDC_EVENT_READ_COMPLETE exits. If the read request could not be accepted, the function returns an error code and transferHandle will contain the value [USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID](#).

If the size parameter is not a multiple of maxPacketSize or is 0, the function returns [USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID](#) in transferHandle and returns an error code as a return value. If the size parameter is a multiple of maxPacketSize and the host send less than maxPacketSize data in any transaction, the transfer completes and the function driver will issue a USB_DEVICE_CDC_EVENT_READ_COMPLETE event along with the USB_DEVICE_CDC_EVENT_READ_COMPLETE_DATA data structure. If the size parameter is a multiple of maxPacketSize and the host sends maxPacketSize amount of data, and total data received does not exceed size, then the function driver will wait for the next packet.

Preconditions

The function driver should have been configured.

Parameters

Parameters	Description
instance	USB Device CDC Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_CDC_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the read request was successful.
data	pointer to the data buffer where read data will be stored.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.

Returns

USB_DEVICE_CDC_RESULT_OK - The read request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The write request could not be added.

USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_SIZE_INVALID - The specified transfer size was not a mulitple of endpoint size or is 0.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Remarks

None.

Example

```
// Shows an example of how to read. This assumes that
// driver was opened successfully.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_RESULT readRequestResult;
USB_DEVICE_CDC_HANDLE instanceHandle;

readRequestResult = USB_DEVICE_CDC_Read(instanceHandle,
                                         &transferHandle, data, 128);

if(USB_DEVICE_CDC_RESULT_OK != readRequestResult)
{
    //Do Error handling here
}

// The completion of the read request will be indicated by the
// USB_DEVICE_CDC_EVENT_READ_COMPLETE event.
```

5.9.4.6.1.6 USB_DEVICE_CDC_SerialStateNotificationSend Function**C**

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_SerialStateNotificationSend(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_TRANSFER_HANDLE * transferHandle,
    USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA * notificationData
);
```

Description

This function places a request to **send** serial state notification data to the host. The function will place the request with the driver, the request will get serviced when the data is requested by the USB host. A handle to the request is returned in the `transferHandle` parameter. The termination of the request is indicated by the `USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE` event. The amount of data transmitted and the transfer handle associated with the request is returned along with the event in the `serialStateNotificationCompleteData` member of `pData` parameter of the event handler. The transfer handle expires when the event handler for the `USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE` event exits. If the **send** request could not be accepted, the function returns an error code and `transferHandle` will contain the value `USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID`.

Preconditions

The function driver should have been configured

Parameters

Parameters	Description
<code>instance</code>	USB Device CDC Function Driver instance.
<code>transferHandle</code>	Pointer to a output only variable that will contain transfer handle.
<code>notificationData</code>	<code>USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA</code> type of notification data to be sent to the host.

Returns

`USB_DEVICE_CDC_RESULT_OK` - The request was successful. `transferHandle` contains a valid transfer handle.

`USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL` - Internal request queue is full. The request could not be added.

`USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_NOT_CONFIGURED` - The specified instance is not configured yet.

`USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID` - The specified instance was not provisioned in the application

and is invalid.

Remarks

None.

Example

```
USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA notificationData;

// This application function could possibly update the notificationData
// data structure.

APP_UpdateNotificationData(&notificationData);

// Now send the updated notification data to the host.

result = USB_DEVICE_CDC_SerialStateNotificationSend
(instanceIndex, &transferHandle, &notificationData);

if(USB_DEVICE_CDC_RESULT_OK != result)
{
    // Error handling here. The transferHandle will contain
    // USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID in this case.
}
```

5.9.4.6.1.7 USB_DEVICE_CDC_Write Function

C

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_Write(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_TRANSFER_HANDLE * transferHandle,
    const void * data,
    size_t size,
    USB_DEVICE_CDC_TRANSFER_FLAGS flags
);
```

Description

This function requests a data write to the USB Device CDC Function Driver Layer. The function places a requests with driver, the request will get serviced as data is requested by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event. The amount of data written and the transfer handle associated with the request is returned along with the event in writeCompleteData member of the pData parameter in the event handler. The transfer handle expires when event handler for the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE exits. If the read request could not be accepted, the function returns an error code and transferHandle will contain the value [USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID](#).

The behavior of the write request depends on the flags and size parameter. If the application intends to [send](#) more data in a request, then it should use the USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING flag. If there is no more data to be sent in the request, the application must use the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE flag. This is explained in more detail here:

- If size is a multiple of maxPacketSize and flag is set as

USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function will append a Zero Length Packet (ZLP) to complete the transfer.

- If size is a multiple of maxPacketSize and flag is set as

USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function will not append a ZLP and hence will

not complete the transfer.

- If size is greater than but not a multiple of maxPacketSize and

flags is set as USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules (length/maxPacketSize) packets and one packet for the residual data.

- If size is greater than but not a multiple of maxPacketSize and flags is

set as USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function schedules (size/maxPacketSize) packets only.

- If size is less than maxPacketSize and flag is set as

USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules one packet.

- If size is less than maxPacketSize and flag is set as

USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function returns an error code and sets the transferHandle parameter to [USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID](#).

Completion of the write transfer is indicated by the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event. The amount of data written along with the transfer handle is returned along with the event.

Preconditions

The function driver should have been configured.

Parameters

Parameters	Description
instance	USB Device CDC Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_CDC_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the write request was successful.
data	pointer to the data buffer that contains the data to written.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.
flags	Flags that indicate whether the transfer should continue or end. Refer to the description for more details.

Returns

USB_DEVICE_CDC_RESULT_OK - The write request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The write request could not be added.

USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_SIZE_INVALID - The specified transfer size and flag parameter are invalid.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Remarks

None.

Example

```

// Below is a set of examples showing various conditions trying to
// send data with the Write() command.
//
// This assumes that driver was opened successfully.
// Assume maxPacketSize is 64.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_RESULT writeRequestHandle;
USB_DEVICE_CDC_INDEX instance;

//-----
// In this example we want to send 34 bytes only.

writeRequestResult = USB_DEVICE_CDC_Write(instance,
                                          &transferHandle, data, 34,
                                          USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//-----
// In this example we want to send 64 bytes only.
// This will cause a ZLP to be sent.

writeRequestResult = USB_DEVICE_CDC_Write(instance,
                                          &transferHandle, data, 64,
                                          USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//-----
// This example will return an error because size is less
// than maxPacketSize and the flag indicates that more
// data is pending.

writeRequestResult = USB_DEVICE_CDC_Write(instanceHandle,
                                          &transferHandle, data, 32,
                                          USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING);

//-----
// In this example we want to place a request for a 70 byte transfer.
// The 70 bytes will be sent out in a 64 byte transaction and a 6 byte
// transaction completing the transfer.

writeRequestResult = USB_DEVICE_CDC_Write(instanceHandle,
                                          &transferHandle, data, 70,
                                          USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//-----
// In this example we want to place a request for a 70 bytes to be sent but
// that we don't end the transfer as more data is coming. 64 bytes of the
// 70 will be sent out and the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE with 64
// bytes. This indicates that the extra 6 bytes weren't sent because it
// would cause the end of the transfer. Thus the user needs add these 6

```



```

// bytes back to the buffer for the next group of data that needs to be sent
// out.

writeRequestResult = USB_DEVICE_CDC_Write(instanceHandle,
                                          &transferHandle, data, 70,
                                          USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

// The completion of the write request will be indicated by the
// USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event.

```

5.9.4.6.2 Data Types and Constants

5.9.4.6.2.1 USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE Type

C

```
typedef USB_DEVICE_CONTROL_TRANSFER_HANDLE USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE;
```

Description

USB Device CDC Function Driver Control Transfer Handle

This is returned by the CDC function driver event handler and should be used by the application while responding to CDC function driver control transfer requests.

Remarks

None.

5.9.4.6.2.2 USB_DEVICE_CDC_EVENT Enumeration

C

```

typedef enum {
    USB_DEVICE_CDC_EVENT_SET_LINE_CODING,
    USB_DEVICE_CDC_EVENT_GET_LINE_CODING,
    USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE,
    USB_DEVICE_CDC_EVENT_SEND_BREAK,
    USB_DEVICE_CDC_EVENT_WRITE_COMPLETE,
    USB_DEVICE_CDC_EVENT_READ_COMPLETE,
    USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE,
    USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT,
    USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED
} USB_DEVICE_CDC_EVENT;

```

Description

USB Device CDC Function Driver Control Transfer Status

This flag is used along with the [USB_DEVICE_CDC_ControlStatus\(\)](#) function to indicate success or failure of a CDC class specific control transfer request.

Members

Members	Description
USB_DEVICE_CDC_EVENT_SET_LINE_CODING	This event occurs when the host issues a SET LINE CODING command. The application must provide a USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING data structure to the device layer to receive the line coding data that the host will provide. The application must provide the buffer by calling the USB_DEVICE_CDC_ControlReceive() function either in the event handler or in the application (outside the event handler function). The application must use the controlTransferHandle parameter provided in the event while calling USB_DEVICE_CDC_ControlReceive() function. The pData parameter will be NULL. The application can use the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event to track completion of the command.
USB_DEVICE_CDC_EVENT_GET_LINE_CODING	This event occurs when the host issues a GET LINE CODING command. The application must provide a USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING data structure to the device layer that contains the line coding data to be provided to the host. The application must provide the buffer by calling the USB_DEVICE_CDC_ControlSend() function either in the event handler or in the application (outside the event handler function). The application must use the controlTransferHandle parameter provided in the event while calling USB_DEVICE_CDC_ControlSend() function. The size of the buffer is indicated by the length parameter. The application can use the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.
USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE	This event occurs when the host issues a SET CONTROL LINE STATE command. The application must interpret the pData parameter as USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE pointer type. This data structure contains the control line state data. The application can then use the USB_DEVICE_CDC_ControlStatus() function to indicate acceptance or rejection of the command. The USB_DEVICE_CDC_ControlStatus() function can be called from the event handler or in the application (out of the event handler context). The application should use the controlTransferHandle while calling the USB_DEVICE_CDC_ControlStatus() function.

USB_DEVICE_CDC_EVENT_SEND_BREAK	This event occurs when the host issues a SEND BREAK command. The application must interpret the pData parameter as a <code>USB_DEVICE_CDC_EVENT_SEND_BREAK_DATA</code> pointer type. This data structure contains the break duration data. The application can then use the <code>USB_DEVICE_CDC_ControlStatus()</code> function to indicate acceptance or rejection of the command. The <code>USB_DEVICE_CDC_ControlStatus()</code> function can be called from the event handler or in the application (out of the event handler context). The application should use the <code>controlTransferHandle</code> while calling the <code>USB_DEVICE_CDC_ControlStatus()</code> function.
USB_DEVICE_CDC_EVENT_WRITE_COMPLETE	This event occurs when a write operation scheduled by calling the <code>USB_DEVICE_CDC_Write()</code> function has completed. The pData parameter should be interpreted as a <code>USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE</code> pointer type. This will contain the transfer handle associated with the completed write transfer and the amount of data written.
USB_DEVICE_CDC_EVENT_READ_COMPLETE	This event occurs when a read operation scheduled by calling the <code>USB_DEVICE_CDC_Read()</code> function has completed. The pData parameter should be interpreted as a <code>USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE</code> pointer type. This will contain the transfer handle associated with the completed read transfer and the amount of data read.
USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE	This event occurs when a serial state notification scheduled using the <code>USB_DEVICE_CDC_SerialDataSend()</code> function, was sent to the host. The pData parameter should be interpreted as a <code>USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE</code> pointer type and will contain the transfer handle associated with the completed <code>send</code> transfer and the amount of data sent.
USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT	This event occurs when the data stage of a control read transfer has completed. This event would occur after the application uses the <code>USB_DEVICE_CDC_ControlSend()</code> function to respond to the <code>USB_DEVICE_CDC_EVENT_GET_LINE_CODING</code> event.

USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	This event occurs when the data stage of a control write transfer has completed. This would occur after the application would respond with a USB_DEVICE_CDC_ControlReceive() function to the USB_DEVICE_CDC_EVENT_SET_LINE_CODING_EVENT and the data has been received.

Remarks

None.

5.9.4.6.2.3 USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING Type

C

```
typedef USB_CDC_LINE_CODING USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING;
```

Description

USB Device CDC Event Get Line Coding Data Type.

This type defines the data type of the data that the user needs to respond with as a result of a USB_DEVICE_CDC_EVENT_GET_LINE_CODING event. This data is sent to the host via the USB_DEVICE_ControlWrite() function.

The valid values for the members of this structure are defined in table 17 of the PSTN120.pdf document. This document is part of the CDC specification download available on the USB.org website: http://www.usb.org/developers/devclass_docs/CDC1.2_WMC1.1_012011.zip

Remarks

None.

5.9.4.6.2.4 USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE Structure

C

```
typedef struct {
    USB_DEVICE_CDC_TRANSFER_HANDLE handle;
    size_t length;
} USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE,
USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE;
```

Description

USB Device CDC Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_CDC_EVENT_READ_COMPLETE and USB_DEVICE_CDC_EVENT_WRITE_COMPLETE events.

Members

Members	Description

5.9.4.6.2.5 USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK Structure

C

```
typedef struct {
    uint16_t duration;
} USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK;
```

Description

USB Device CDC Event Set Break Data Type.

This defines the data type of the data generated to the CDC event handler on a USB_DEVICE_CDC_EVENT_SEND_BREAK event.

The SEND_BREAK command is described in section 6.3.13 of the PSTN120.pdf document. This document is part of the CDC specification [download available on the USB.org website:](http://www.usb.org/developers/devclass_docs/CDC1.2_WMC1.1_012011.zip)

Remarks

None.

5.9.4.6.2.6 USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE Structure

C

```
typedef struct {
    USB_DEVICE_CDC_TRANSFER_HANDLE handle;
    size_t length;
} USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE,
USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE;
```

Description

USB Device CDC Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_CDC_EVENT_READ_COMPLETE and USB_DEVICE_CDC_EVENT_WRITE_COMPLETE events.

Members

Members	Description

5.9.4.6.2.7 USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE Type

C

```
typedef USB_CDC_CONTROL_LINE_STATE USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE;
```

Description

USB Device CDC Event Set Control Line State Data Type.

This defines the data type of the data that should be provided to the function driver via the [USB_DEVICE_CDC_ControlReceive\(\)](#) function when the function driver generates USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE event.

Remarks

None.

5.9.4.6.2.8 USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING Type

C

```
typedef USB_CDC_LINE_CODING USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING;
```

Description

USB Device CDC Event Set Line Coding Data Type.

This defines the data type of the data generated to the CDC event handler on a USB_DEVICE_CDC_EVENT_SET_LINE_CODING event.

The valid values for the members of this structure are defined in table 17 of the PSTN120.pdf document. This document is part of the CDC specification download available on the USB.org website: http://www.usb.org/developers/devclass_docs/CDC1.2_WMC1.1_012011.zip

Remarks

None.

5.9.4.6.2.9 USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE Structure

C

```
typedef struct {
    USB_DEVICE_CDC_TRANSFER_HANDLE handle;
    size_t length;
} USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE,
USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE;
```

Description

USB Device CDC Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_CDC_EVENT_READ_COMPLETE and USB_DEVICE_CDC_EVENT_WRITE_COMPLETE events.

Members

Members	Description

5.9.4.6.2.10 USB_DEVICE_CDC_EVENT_HANDLER Type

C

```
typedef USB_DEVICE_CDC_EVENT_RESPONSE (* USB_DEVICE_CDC_EVENT_HANDLER)(USB_DEVICE_CDC_INDEX
instanceIndex, USB_DEVICE_CDC_EVENT event, USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE
controlTransferHandle, void * pData, uintptr_t context);
```

Description

USB Device CDC Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Device CDC Function Driver event handling callback function. The application must register a pointer to a CDC Function Driver events handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the CDC Function Driver. The function driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

instanceIndex - Instance index of the CDC Function Driver that generated the event.

event - Type of event generated.

controlTransferHandle - Control Transfer Handle for CDC function driver events that require application response. The application should use this handle when calling the USB CDC Device Control Transfer functions to respond to the events.

pData - This parameter should be type casted to a event specific pointer type based on the event that has occurred. Refer to the [USB_DEVICE_CDC_EVENT](#) enumeration description for more details.

context - Value identifying the context of the application that was registered along with the event handling function.

Remarks

None.

5.9.4.6.2.11 USB_DEVICE_CDC_EVENT_RESPONSE Type

C

```
typedef void USB_DEVICE_CDC_EVENT_RESPONSE;
```

Description

USB Device CDC Function Driver Event Handler Response Type

This is the return type of the CDC Function Driver event handler.

Remarks

None.

5.9.4.6.2.12 USB_DEVICE_CDC_INDEX Type

C

```
typedef uintptr_t USB_DEVICE_CDC_INDEX;
```

Description

USB Device CDC Function Driver Index

This uniquely identifies a CDC Function Driver instance.

Remarks

None.

5.9.4.6.2.13 USB_DEVICE_CDC_INIT Structure

C

```
typedef struct {
    size_t queueSizeRead;
    size_t queueSizeWrite;
    size_t queueSizeSerialStateNotification;
} USB_DEVICE_CDC_INIT;
```

Description

USB Device CDC Function Driver Initialization Data Structure

This data structure must be defined for every instance of the CDC function driver. It is passed to the CDC function driver, by the Device Layer, at the time of initialization. The funcDriverInit member of the Device Layer Function Driver registration table entry must point to this data structure for an instance of the CDC function driver.

Members

Members	Description

5.9.4.6.2.14 USB_DEVICE_CDC_RESULT Enumeration

C

```
typedef enum {
    USB_DEVICE_CDC_RESULT_OK,
    USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_SIZE_INVALID,
    USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL,
    USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID,
    USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_NOT_CONFIGURED,
    USB_DEVICE_CDC_RESULT_ERROR_PARAMETER_INVALID,
    USB_DEVICE_CDC_RESULT_ERROR_CONTROL_TRANSFER_FAILED
}
```

```
} USB_DEVICE_CDC_RESULT;
```

Description

USB Device CDC Function Driver USB Device CDC Result enumeration.

This enumeration lists the possible USB Device CDC Function Driver operation results. These values are returned by USB Device CDC Library functions.

Members

Members	Description
USB_DEVICE_CDC_RESULT_OK	The operation was successful

5.9.4.6.2.15 USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA Type**C**

```
typedef USB_CDC_SERIAL_STATE USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA;
```

Description

USB Device CDC Serial State Notification Data Type.

This defines the data type for the CDC Serial State. This data is sent to the host over the CDC notification interface.

Remarks

None.

5.9.4.6.2.16 USB_DEVICE_CDC_TRANSFER_FLAGS Enumeration**C**

```
typedef enum {  
    USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE,  
    USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING  
} USB_DEVICE_CDC_TRANSFER_FLAGS;
```

Description

USB Device CDC Transfer Flags

These flags are used to indicate status of the pending data while sending data to the host by using the [USB_DEVICE_CDC_Write\(\)](#) function.

Members

Members	Description
USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE	This flag indicates there is no further data to be sent in this transfer and that the transfer should end. If the size of the transfer is a multiple of the maximum packet size for related endpoint configuration, the function driver will send a zero length packet to indicate end of the transfer to the host.

USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING	This flag indicates there is more data to be sent in this transfer. If the size of the transfer is a multiple of the maximum packet size for the related endpoint configuration, the function driver will not send a zero length packet. If the size of the transfer is greater than (but not a multiple of) the maximum packet size, the function driver will only send maximum packet size amount of data. If the size of the transfer is greater than endpoint size but not an exact multiple of endpoint size, only the closest endpoint size multiple bytes of data will be sent. This flag should not be specified if the size of the transfer is less than maximum packet size.

Remarks

The relevance of the specified flag depends on the size of the buffer. Refer to the individual flag descriptions for more details.

5.9.4.6.2.17 USB_DEVICE_CDC_TRANSFER_HANDLE Type**C**

```
typedef uintptr_t USB_DEVICE_CDC_TRANSFER_HANDLE;
```

Description

USB Device CDC Function Driver Transfer Handle Definition

This definition defines a USB Device CDC Function Driver Transfer Handle. A Transfer Handle is owned by the application but its value is modified by the [USB_DEVICE_CDC_Write\(\)](#), [USB_DEVICE_CDC_Read\(\)](#) and the [USB_DEVICE_CDC_SerialStateDataSend](#) functions. The transfer handle is valid for the life time of the transfer and expires when the transfer related event had occurred.

Remarks

None.

5.9.4.6.2.18 USB_DEVICE_CDC_EVENT_RESPONSE_NONE Macro**C**

```
#define USB_DEVICE_CDC_EVENT_RESPONSE_NONE
```

Description

USB Device CDC Function Driver Event Handler Response None

This is the definition of the CDC Function Driver Event Handler Response Type none.

Remarks

Intentionally defined to be empty.

5.9.4.6.2.19 USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID Macro**C**

```
#define USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID ((USB_DEVICE_CDC_TRANSFER_HANDLE)(-1))
```

Description

USB Device CDC Function Driver Invalid Transfer Handle Definition

This definition defines a USB Device CDC Function Driver Invalid Transfer Handle. A Invalid Transfer Handle is returned by the [USB_DEVICE_CDC_Write\(\)](#), [USB_DEVICE_CDC_Read\(\)](#) and the [USB_DEVICE_CDC_SerialStateNotificationSend\(\)](#) functions

when the request was not successful.

Remarks

None.

5.9.4.7 Files

Files

Name	Description
usb_device_cdc.h	USB Device CDC Function Driver Interface

Description

5.9.4.7.1 usb_device_cdc.h

USB Device CDC Function Driver Interface

This file describes the USB Device CDC Function Driver interface.

Enumerations

	Name	Description
	USB_DEVICE_CDC_EVENT	USB Device CDC Function Driver Control Transfer Status
	USB_DEVICE_CDC_RESULT	USB Device CDC Function Driver USB Device CDC Result enumeration.
	USB_DEVICE_CDC_TRANSFER_FLAGS	USB Device CDC Function Driver Transfer Flags

Functions

	Name	Description
	USB_DEVICE_CDC_ControlReceive	This function allows the application to respond to the CDC function driver specific control transfer requests. It allows the application to receive data from the host in the data stage of the control transfer.
	USB_DEVICE_CDC_ControlSend	This function allows the application to respond to the CDC function driver specific control transfer requests. It allows the application to send data to the host in the data stage of the control transfer.
	USB_DEVICE_CDC_ControlStatus	This function allows the application to complete the status stage of the the CDC Function Driver specific control transfer request.
	USB_DEVICE_CDC_EventHandlerSet	This function registers a event handler for the specified CDC function driver instance.
	USB_DEVICE_CDC_Read	This function requests a data read from the USB Device CDC Function Driver Layer.
	USB_DEVICE_CDC_SerialStateNotificationSend	This function schedules a request to send serial state notification to the host.
	USB_DEVICE_CDC_Write	This function requests a data write to the USB Device CDC Function Driver Layer.

Macros

	Name	Description
	USB_DEVICE_CDC_EVENT_RESPONSE_NONE	USB Device CDC Function Driver Event Handler Response Type None.
	USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID	USB Device CDC Function Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE	USB Device CDC Function Driver Read and Write Complete Event Data.
	USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK	USB Device CDC Event Set Break Data Type.
	USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE	USB Device CDC Function Driver Read and Write Complete Event Data.
	USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE	USB Device CDC Function Driver Read and Write Complete Event Data.
	USB_DEVICE_CDC_INIT	USB Device CDC Function Driver Initialization Data Structure

Types

	Name	Description
	USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE	USB Device CDC Function Driver Control Transfer Handle
	USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING	USB Device CDC Event Get Line Coding Data Type.
	USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE	USB Device CDC Event Set Line Coding Data Type.
	USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING	USB Device CDC Event Set Line Coding Data Type.
	USB_DEVICE_CDC_EVENT_HANDLER	USB Device CDC Event Handler Function Pointer Type.
	USB_DEVICE_CDC_EVENT_RESPONSE	USB Device CDC Function Driver Event Callback Response Type
	USB_DEVICE_CDC_INDEX	USB Device CDC Function Driver Index
	USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA	USB Device Serial State Notification Data Type.
	USB_DEVICE_CDC_TRANSFER_HANDLE	USB Device CDC Function Driver Transfer Handle Definition.

Company

Microchip Technology Incorporated

FileName: usb_device_cdc.h

5.9.5 USB Generic Device Library

5.9.5.1 Introduction

Introduces the MPLAB Harmony USB Generic Device Library.

Description

The USB Generic Device library provides a framework to develop custom or vendor class USB device applications. The library supports all USB transfer types (control, bulk, interrupt, isochronous) along with endpoint management functions. The USB device generic library implementation supports USB specification 2.0. Following figure illustrates how Generic Device Library fits into the MPLAB Harmony USB Device Stack architecture.

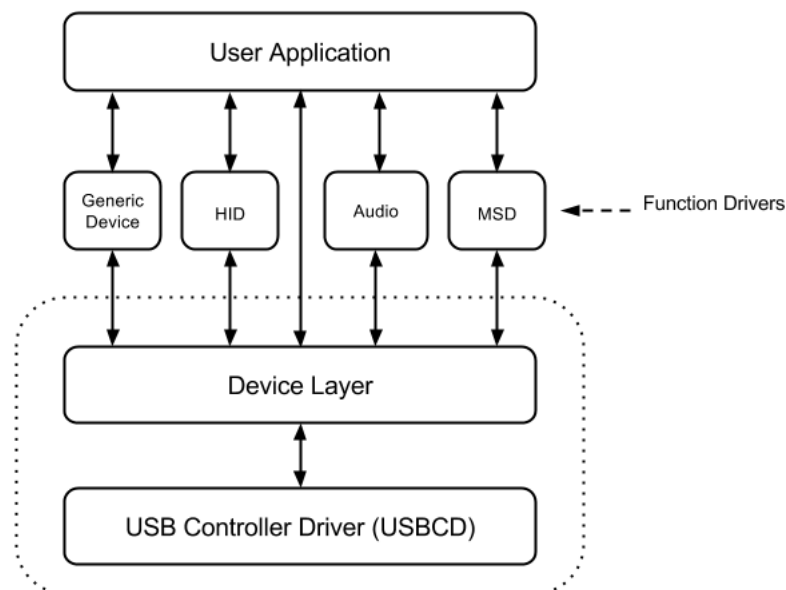


Figure: MPLAB Harmony USB Device library architecture

This USB Generic Device Library offers services to the application to interact and respond to the host requests. USB Generic Device library is also referred to as Generic function driver in this document.

5.9.5.2 Release Notes

MPLAB Harmony Version: v0.70b **USB Generic Device Library Version :** 0.01 Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.9.5.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.9.5.4 Library Architecture

5.9.5.4.1 Library Overview

Provides an overview of the Generic Device function driver.

Description

The Generic Device function driver overview diagram is shown below.

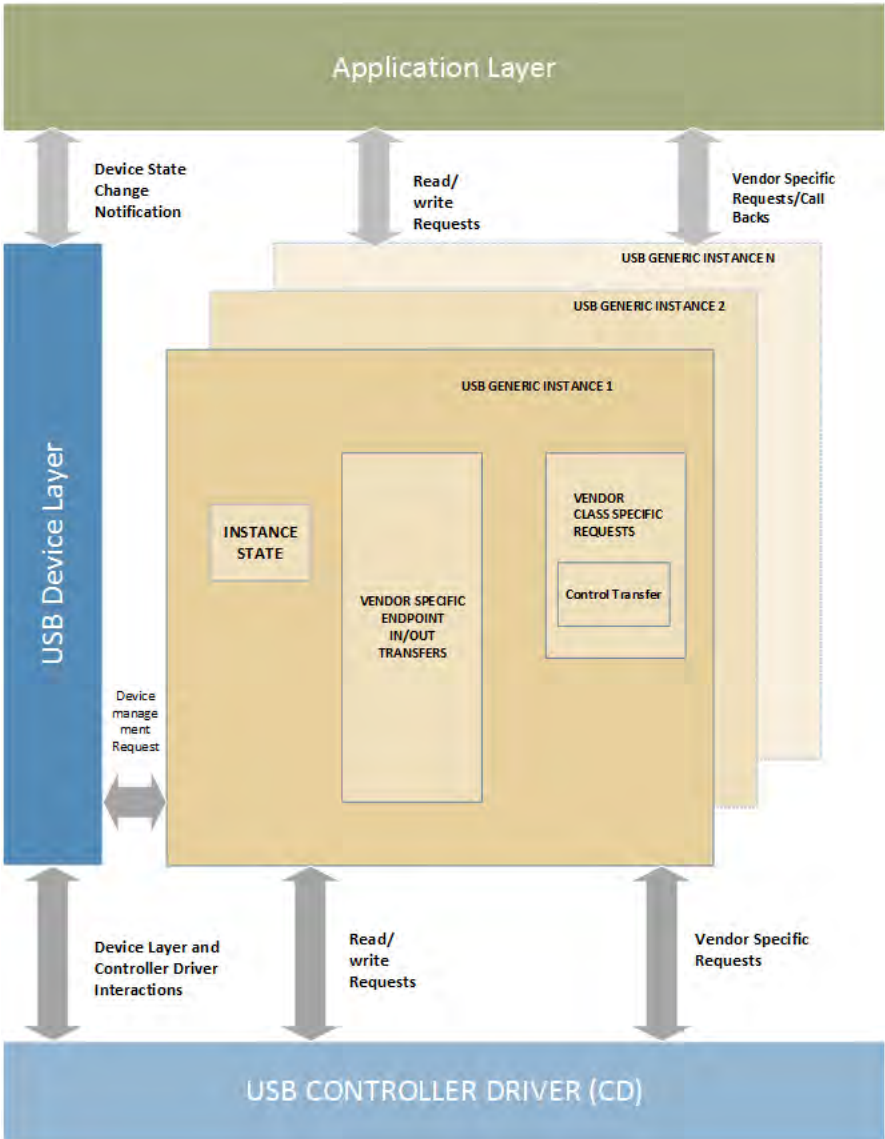


Figure: USB DEVICE GENERIC Library Overview diagram

A Generic function driver instance is a collection of Generic USB interfaces.

The device management on end point zero is handled by the device library(vendor class specific requests are routed to the Generic function driver by device library).

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the USB DEVICE Generic module.

Library Interface Section	Description
Client Setup	Details Opening an instance of the USB Generic function driver, registering callback with the USB Generic function driver and closing an instance of the USB Generic function driver.
Client Data Handling	Details read/write to/from the host
Client Subclass Specific	Details the supported sub class specific management requests and notification.

5.9.5.4.2 Abstraction Model

Provides an architectural overview of the USB Generic Device function driver.

Description

The USB Generic Device function driver offers various services to a vendor specific class USB device to communicate with the host by abstracting USB specification details. The USB Generic Device function driver with USB DEVICE layer and USB controller driver forms the basic library entity using which the device can communicate with the USB host. The basic software architecture into which Generic function driver fits into is depicted in the following diagram.

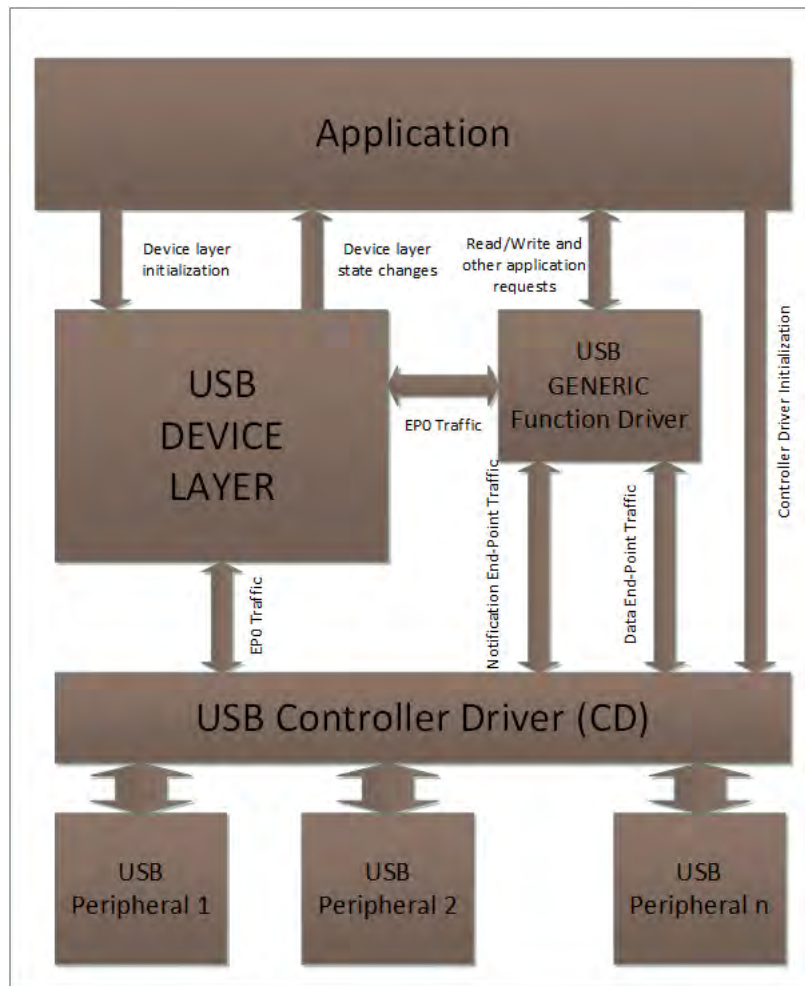


Figure: USB DEVICE Generic Software Abstraction Block Diagram

Vendor specific functions have to register a set of standard APIs with the generic library. This library calls these APIs as call back function at run time. For example, vendor specific tasks function routine is called when USB device generic function driver receives vendor specific transfer from a USB host. All generic function driver tasks are called from the task routine of USB device layer. so here maximum performance of the device can be achieved with this library.

The USB controller driver takes the responsibility of managing the USB peripheral on the device. The USB device layer handles the device enumeration etc. For more details about the USB controller driver and USB device layer library kindly refer to the corresponding documents.

5.9.5.5 Using the Library

5.9.5.5.1 Files to Include

Describes which files to include in the project to use the Generic Device Function Driver.

Description

The following table shows the files that must be included in the project in order to use the Generic Function Driver. These files are located in the framework folder of the MPLAB Harmony installation. It is assumed that the Device Layer files (and the files needed by the Device Layer) are already included in the project. A path to the framework folder in the Harmony installation should be added to the project include search path.

Filename	Description
framework\usb\src\usb_device_generic.c	Contains the implementation of the Generic device function driver.
framework\usb\usb_device_generic.h	Must be included in every file that needs to invoke the Generic device function driver API
system_config.h	User created file which contains the Generic device function driver configuration.

5.9.5.5.2 Library Configuration

Describes how to configure the Generic Function Driver.

Description

The configuration of the USB Generic Device Function Driver is contained in system_config.h These configuration settings will apply to all instances of the USB Generic Device Function Driver. This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the [Applications Overview](#) section for more details.

This section explains the data structures that need to be setup by the application to use the Generic function driver. For setting data relevant for other modules (device layer and controller driver) kindly refer to the respective module documentation.

Constants:

USB_DEVICE_GEN_DRIVER_MAX_INSTANCES: The maximum number of Generic function driver instances to be supported. The valid values are greater than or equal to 1. this constant is not defined the default value taken by the Generic function driver is 1.

Data structures:

The application should form structure USB_DEVICE_GEN_DRIVER_INIT_INSTANCE. The structure takes pointers to the following functions.

- A vendor defined initialize function that is called by the device layer when it initializes the generic function driver. This will happen when the device layer process a SET CONFIGURATION command from the host.
- A vendor defined de-initialize function that is called by the device layer when it de-initializes the generic function driver. This will happen when the device layer process a reset event or is required to change the configuration by the host.
- A vendor defined Setup packet handler. This function is called by the device layer when it needs to pass class specific control

transfer to the generic function driver.

- A vendor specified tasks functions. This is called periodically by the device layer. This function can maintain a vendor device state machine.

The application needs to define an array of USB_DEVICE_GEN_DRIVER_MAX_INSTANCES (defined by the application in system_config.h) such structures. For example, if the application needs two instances of the Generic Device, then the size of this array should be two.

Example: Creating the USB_DEVICE_GEN_DRIVER_INIT_INSTANCE structure.

```
USB_DEVICE_GEN_DRIVER_INIT_INSTANCE genUsbInitInstance[] =
{
    {
        .vendorSpecificInitialize = APP_Initialize,
        .vendorSpecificDeInitialize = APP_DeInitialize,
        .vendorSpecificSetupPacketHandler = APP_SetupPacketHandler,
        .tasks = APP_Tasks
    }
};
```

5.9.5.5.3 Client Setup

Note: To invoke any of the client setup routines listed in this section the device should be in configured state. Any attempts invoking the client setup routines before the configured state will result in graceful exit.

Before using any of the USB Generic function driver services it needs to be setup once in the application.

Setting up the USB Generic function driver involves

1. Opening an instance of the USB Generic function driver.
2. Registering callback with the USB Generic function driver.
3. Closing an instance of the USB Generic function driver.

Opening an Instance of the USB GENERIC function driver:

Before attempting to use any of the USB Generic function driver services an instance of it needs to be opened by the application. An instance of USB Generic function driver is defined as a group of minimum interfaces that constitute a Generic USB device.

To open an instance of USB Generic function driver application should call the function USB_DEVICE_GEN_DRIVER_Open.

Example: Opening a USB Generic instance

```
// Client Handle
DRV_HANDLE genHandle;

// open Generic USB device function driver instance 0
genHandle = USB_DEVICE_GEN_DRIVER_Open ( 0 );

// Check the validity of the handle returned
if(DRV_HANDLE_INVALID == genHandle)
{
    \ Handle the error
}
```

Registering callback with the USB GENERIC function driver:

USB Generic function driver will inform the status of various events (both management and data) to the application using a callback function registered with the USB Generic function driver.

This callback function should be registered with the function driver before expecting any event notifications.

There are three sources of generating events

- Data transmission request (USB_DEVICE_GEN_DRIVER_CALLBACK_TX_)
- Data reception request (USB_DEVICE_GEN_DRIVER_CALLBACK_RX)
- Management Requests (USB_DEVICE_GEN_DRIVER_CALLBACK_MANAGEMENT)

Each of these requests will result in the generation of events of type USB_DEVICE_GEN_DRIVER_CALLBACK_TYPE

For a detailed list of events go through the above enum data type.

To register a callback with the Generic function driver application should first define a function of the signature USB_DEVICE_GEN_DRIVER_CALLBACK_EVENT.

Once defined it can be registered with the Generic function driver using the function USB_DEVICE_GEN_DRIVER_RegisterCallBack.

Example: Registering an application callback with the Generic function driver

```
// Application callback to handle relevant events.
void app_gen_CallBack ( SYS_MODULE_INDEX funcDriverIndex,
                        USB_DEVICE_GEN_DRIVER_CALLBACK_TYPE callback,
                        USB_DEVICE_GEN_DRIVER_CALLBACK_EVENT event)
{
    switch (callback)
    {
        case USB_DEVICE_GEN_DRIVER_CALLBACK_TX:
            break;
        case USB_DEVICE_GEN_DRIVER_CALLBACK_RX:
            break;
        case USB_DEVICE_GEN_DRIVER_CALLBACK_MANAGEMENT:
            break;
        default:
            break;
    }
}

int main ()
{
    DRV_HANDLE genHandle;
    USB_ERROR_STATUS err;

    // open the Generic USB device driver instance 0 ( or what ever is applicable)
    genHandle = USB_DEVICE_GEN_DRIVER_Open ( 0 );

    err = USB_DEVICE_GEN_DRIVER_RegisterCallBack (genHandle,app_gen_CallBack );

    if(err)
    {
        //handle error
    }

    // app code

    // Occurrence of any event of type USB_DEVICE_GEN_DRIVER_CALLBACK_EVENT results in
    // calling of the registered application callback

    return 0;
}
```

Closing an Instance of the USB Generic function driver:

If at all the application wants to close an opened Generic function driver instance it can do so by calling the function `USB_DEVICE_GEN_DRIVER_Close`. The application should pass the client handle it received with opened the Generic function driver instance as the parameter. If the close is successful 0 is returned by the Generic to indicate success.

Example: Closing a Generic USB instance

```
// Return status
USB_ERROR_STATUS status;

// Close Generic USB device driver instance 0
status = USB_DEVICE_GEN_DRIVER_Close(0);

if(USB_ERROR_INVALID_HANDLE ==status)
{
    // Generic USB device function driver instance 0 is either not opened/initialized yet
}

else if(status == USB_ERROR_OK)
{
    // Generic USB device function driver instance 0 is successfully closed.
    // app code
}
```

Before closing the Generic function driver instance the application should be aware of the following

1. Once an instance of the Generic function driver is closed the application cannot write to or read from the data end points(if driven by Generic function driver).
2. The application will not get any event notifications .

INITIAL SETUP:

Before invoking any of the services offered by the Generic function driver following steps should have been completed

1. Certain key data structures need to be setup(initialized) by the application.
2. Certain system functions belonging to the MPLAB Harmony USB Device library should have been called.
3. The device should be in configured state.

The setup of the key data structures is explained in the Configuring the Library section. The following section focuses on the key system functions that need to be called by the application before using Generic function driver.

The following diagram explains the sequence of system functions (belonging to the MPLAB Harmony USB Device library only) invocations.

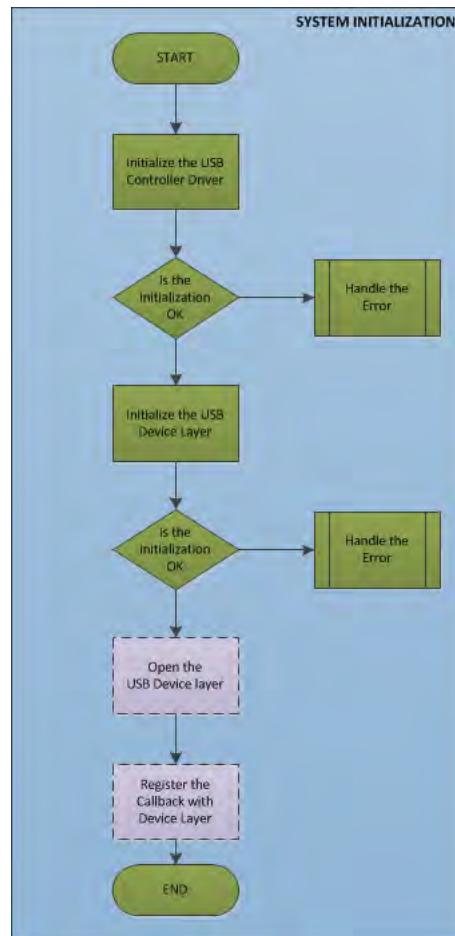


Figure: System Initialization

Example: System initialization

```

// Populate USB Controller driver init data
usbInitData.usbID = USB_ID_1;
usbInitData.usbModuleSetup.OpMode           = USB_OPMODE_DEVICE;
usbInitData.usbModuleSetup.ppMode           = USB_PING_PONG_MODE_SELECTION;
usbInitData.usbModuleSetup.StopInIdle       = false;
usbInitData.usbModuleSetup.SuspendInSleep   = false;

// Initialize the USB Control driver
driverObjectHandle = DRV_USB_Initialize(DRV_USB_INDEX_0, (SYS_MODULE_INIT*)&usbInitData);
if (SYS_MODULE_OBJ_INVALID == driverObjectHandle)
{
    // Handle error
}

// NOTE: USB controller driver is opened by both device layer and function driver and not by
// the application.

// Populate USB Device layer init data
deviceLayerInit.moduleInit.value           = SYS_MODULE_POWER_RUN_FULL;
deviceLayerInit.usbDriverIndex             = DRV_USB_INDEX_0;
deviceLayerInit.usbMasterDescriptor        = (USB_MASTER_DESCRIPTOR*)&usbMasterDescriptor;
deviceLayerInit.registeredFuncCount        = 1;
deviceLayerInit.registeredFunctions        =
(USB_DEVICE_FUNC_REGISTRATION_TABLE*)funcRegistrationTable;
  
```

```

// Initialize the USB device layer
usbDeviceObj = USB_DEVICE_Initialize( USB_DEVICE_INDEX_0, (SYS_MODULE_INIT*)&deviceLayerInit );

// Open the device layer.
usbDevHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0 );

// Register a callback with device layer to get event notification (for end point 0)
USB_DEVICE_EventCallBackSet(usbDevHandle, usbDeviceEventCallBack);

```

The following example provides a complete Generic function driver setup. This example do not use separate functions to do activities like system initialization, Generic function driver setup etc for the sake of clarity. It is recommended to use separate functions to have a structured view of the project.

```

// Application callback to handle relevant events.
void app_gen_CallBack ( SYS_MODULE_INDEX funcDriverIndex,
                        USB_DEVICE_GEN_DRIVER_CALLBACK_TYPE callback,
                        USB_DEVICE_GEN_DRIVER_CALLBACK_EVENT event)
{
    switch (callback)
    {
        case USB_DEVICE_GEN_DRIVER_CALLBACK_TX:
            break;
        case USB_DEVICE_GEN_DRIVER_CALLBACK_RX:
            break;
        case USB_DEVICE_GEN_DRIVER_CALLBACK_MANAGEMENT:
            break;
        default:
            break;
    }
}

// Device event callback of the application.
// This will be registered with the device layer
// during the system initialization.
void usbDeviceEventCallBack(USB_DEVICE_EVENTS events)
{
    if(events.setConfiguration)
    {
        // Device is configured.
        // Initialize function drivers.
        isConfigured = true;

        ////////////////////////////////////////////////// Generic function driver Setup- Begin////////////////////////////////////

        DRV_HANDLE genHandle;
        USB_ERROR_STATUS err;

        // open the Generic USB device driver instance 0 ( or what ever is applicable)
        genHandle = USB_DEVICE_GEN_DRIVER_Open ( 0 );

        err = USB_DEVICE_GEN_DRIVER_RegisterCallBack (genHandle,app_gen_CallBack );

        if(err)
        {
            //handle error
        }

        ////////////////////////////////////////////////// Generic function driver Setup- End////////////////////////////////////
    }
}

int main ()
{

```

```

// Device layer Init structure
USB_DEVICE_INIT deviceLayerInit;

// objects for Controller driver and device layer (returned by init routines)
SYS_MODULE_OBJ driverObjectHandle, usbDeviceObj;

// Client handle
DRV_HANDLE genHandle;

// Client handle for device layer
DRV_HANDLE usbDevHandle;

// USB controller driver init data
DRV_USB_INIT usbInitData;

////////// system initialization- Begin//////////

// Populate USB Controller driver init data
usbInitData.usbID = USB_ID_1;
usbInitData.usbModuleSetup.OpMode           = USB_OPMODE_DEVICE;
usbInitData.usbModuleSetup.ppMode           = USB_PING_PONG_MODE_SELECTION;
usbInitData.usbModuleSetup.StopInIdle       = false;
usbInitData.usbModuleSetup.SuspendInSleep   = false;

// Initialize the USB Control driver
driverObjectHandle = DRV_USB_Initialize(DRV_USB_INDEX_0, (SYS_MODULE_INIT*)&usbInitData);
if (SYS_MODULE_OBJ_INVALID == driverObjectHandle)
{
    // Handle error
}

// NOTE: USB controller driver is opened by both device layer and
// function driver and not by the application.

// Populate USB Device layer init data
deviceLayerInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
deviceLayerInit.usbDriverIndex   = DRV_USB_INDEX_0;
deviceLayerInit.usbMasterDescriptor = (USB_MASTER_DESCRIPTOR*)&usbMasterDescriptor;
deviceLayerInit.registeredFuncCount = 1;
deviceLayerInit.registeredFunctions =
(USB_DEVICE_FUNC_REGISTRATION_TABLE*)funcRegistrationTable;

// Initialize the USB device layer
usbDeviceObj = USB_DEVICE_Initialize( USB_DEVICE_INDEX_0, (SYS_MODULE_INIT*)&deviceLayerInit
);

// Open the device layer.
usbDevHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0 );

// Register a callback with device layer to get event notification (for end point 0)
USB_DEVICE_EventCallBackSet(usbDevHandle, usbDeviceEventCallBack);

////////// system initialization- End//////////

// the application body
while(1)
{
    // Device layer task
    USB_DEVICE_Tasks(usbDeviceObj);

    // Check if the enumeration is complete
    if(isConfigured)
    {
        // Start calling application tasks
    }
}

```

```
return 0;
}
```

5.9.5.5.4 Client Data Handling

The USB Generic function driver provides APIs to handle data to/from the host. The following section provides the usage of these APIs.

The data transfer in the Generic function driver happens over the end points with all the type of transfer modes (BULK, ISOCRONOUS, INTERRUPT) at both direction of IN and OUT.

The following sections focus on the data transmission over the end points .

The read and write naming conventions used for the data handling APIs are from the device point of view. So USB_DEVICE_GEN_DRIVER_ReadEp function is a request to the MPLAB Harmony USB Device library to get data available from the host. Similarly USB_DEVICE_GEN_DRIVER_WriteEp function is a request to make data available to the host.

Generic USB Buffer Objects:

In order place a read/write request with the Generic function driver the application should first create a buffer object of the type USB_DEVICE_GEN_DRIVER_DATA_BUFFER_OBJECT. It is recommended to create separate buffer objects for read and write operations. A USB_DEVICE_GEN_DRIVER_DATA_BUFFER_OBJECT contains a pointer to the data buffer, length of the buffer and transfer count. The transfer count is updated by the Generic function driver (after the read/write request is submitted) to indicate the actual amount of data transferred to/from the host.

Example: Generic USB DATA Buffer Object Creation

```
// buffer for read operation of size 64 bytes
uint8_t inBuffer[64];
// define and initialize the buffer object
USB_DEVICE_GEN_DRIVER_DATA_BUFFER_OBJECT inBufferObject = {(uint8_t*)&inBuffer, 64, 0};
```

Read From the Host:

As explained above the application should make use of the API USB_DEVICE_GEN_DRIVER_ReadEp to get data from the host. Before placing a read request with the Generic function driver the application should get the pipeHandle using USB_DEVICE_GEN_DRIVER_GetPipeHandle . If the received is valid , applicatio has to check if it can place the next read request. This status can be obtained using the API USB_DEVICE_GEN_DRIVER_TxRx_StatusGet. If this API returns USB_DEVICE_GEN_DRIVER_INSTANCE_RX_READY then the application can place a request for a read operation. If the received status is anything else then the application should poll until the receive status is ready. The following example shows the usage of read request.

Example: Data Read Request

```
#define APP_USB_DEVICE_GEN_DRIVER_FUNC_INDEX 0
// create buffer
uint8_t outBuffer[64];

uint8_t datalen = 64;

uint8_t epNum = 1;

dirn = USB_DEVICE_GEN_DRIVER_EP_IN_INDEX; //OUT:(0x00) or IN: 0x01

DRV_HANDLE genHandle,pipeHandle;

USB_ERROR_STATUS genErr;

//create the buffer object with datalength 64
USB_DEVICE_GEN_DRIVER_DATA_BUFFER_OBJECT readBufferObject = {&outBuffer, 64, 0};
```

```

// open the Generic USB device driver instance 0 ( or what ever is applicable)
genHandle = USB_DEVICE_GEN_DRIVER_Open ( APP_USB_DEVICE_GEN_DRIVER_FUNC_INDEX );

if(USB_DEVICE_GEN_DRIVER_RegisterCallBack (genHandle,app_gen_CallBack ))
{
    //handle error
}

//Get the pipe handle for the endpoint
pipeHandle = USB_DEVICE_GEN_DRIVER_GetPipeHandle(genHandle, epNum, dirn))
if ( DRV_HANDLE_INVALID == pipeHandle)
{
    //The request was not successful
}

//Got the pipe handle for the endpoint
//Pass the pipe handle to USB_DEVICE_GEN_DRIVER_ReadEp

genErr = USB_DEVICE_GEN_DRIVER_ReadEp( pipeHandle,&readBufferObject);

if(USB_ERROR_OK != genErr)
{
    //Read request is not accepted.
}

```

If the application has registered a callback function with the Generic function driver then any events related to the read request will be informed to the application via this callback. In the callback the application can check whether all of the data requested is received. The following example shows this.

Example: Data Read Request with application callback

```

#define APP_USB_DEVICE_GEN_DRIVER_FUNC_INDEX 0

// Application callback to handle relevant events.
void app_gen_CallBack ( SYS_MODULE_INDEX funcDriverIndex,
                        USB_DEVICE_GEN_DRIVER_CALLBACK_TYPE callback,
                        USB_DEVICE_GEN_DRIVER_CALLBACK_EVENT event)
{
    switch (callback)
    {
        case USB_DEVICE_GEN_DRIVER_CALLBACK_TX:
            break;
        case USB_DEVICE_GEN_DRIVER_CALLBACK_RX:
            if (readBufferObject.transferCount == readBufferObject.dataLen)
            {
                //entire data is read from the host.
            }
            else
            {
                //some of the data is not read.
                //get some logic in place to handle this
            }
            break;
        case USB_DEVICE_GEN_DRIVER_CALLBACK_MANAGEMENT:
            break;
        default:
            break;
    }
}

```



```

int main ()
{
    // create buffer
    uint8_t outBuffer[64];

    uint8_t datalen = 64;

    uint8_t epNum = 1;

    dirn = USB_DEVICE_GEN_DRIVER_EP_OUT_INDEX; //OUT:(0x00) or IN: 0x01

    DRV_HANDLE genHandle;

    USB_ERROR_STATUS genErr;

    DRVHANDLE pipeHandle;

    //create the buffer object with datalength 64
    USB_DEVICE_GEN_DRIVER_DATA_BUFFER_OBJECT readBufferObject = {&outBuffer, 64, 0};

    // open the Generic USB device driver instance 0 ( or what ever is applicable)
    genHandle = USB_DEVICE_GEN_DRIVER_Open ( 0 );

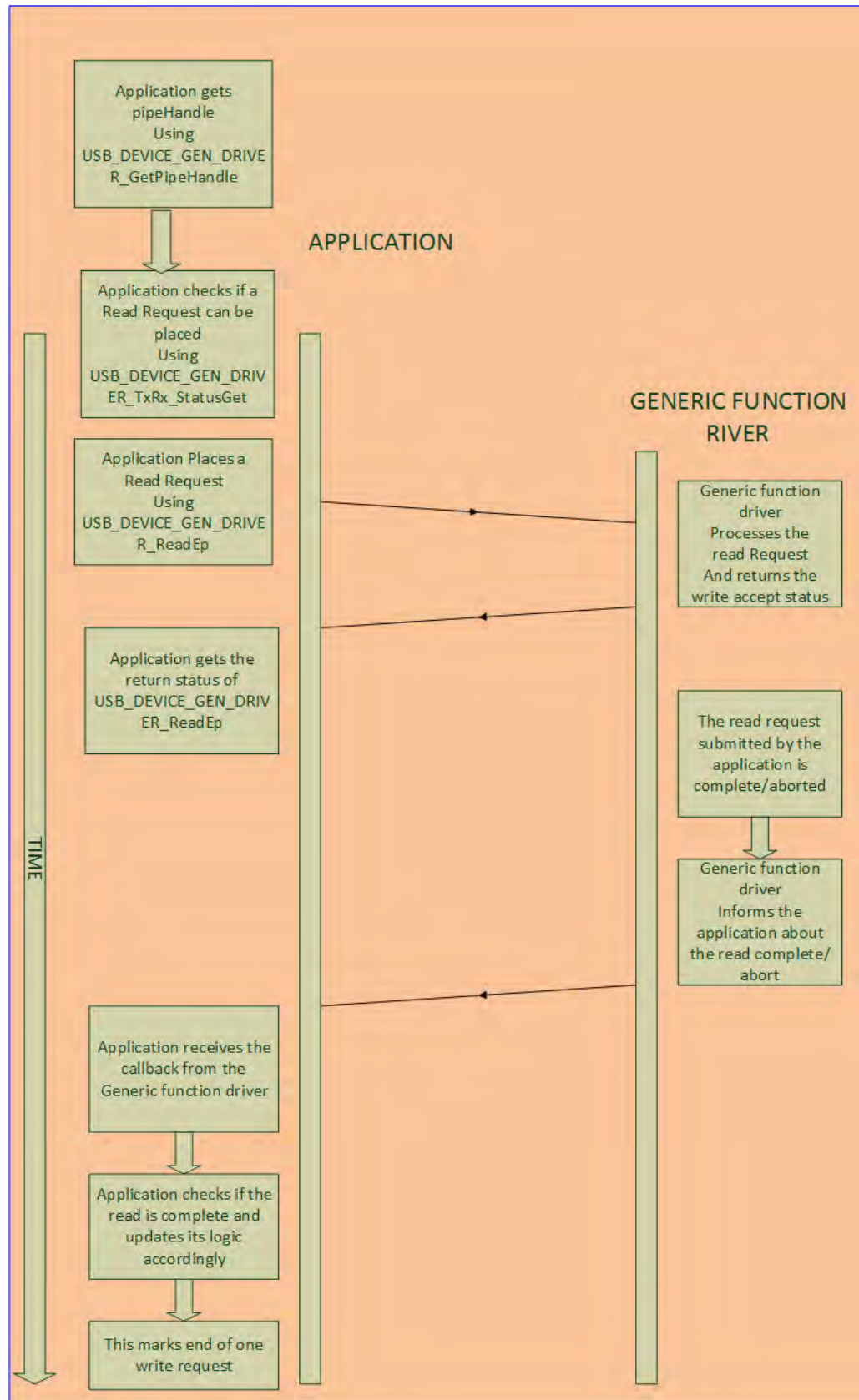
    if(USB_DEVICE_GEN_DRIVER_RegisterCallBack (genHandle,app_gen_CallBack ))
    {
        //handle error
    }
    //Get the pipe handle for the endpoint
    pipeHandle = USB_DEVICE_GEN_DRIVER_GetPipeHandle(genHandle, epNum, dirn)
    if ( DRV_HANDLE_INVALID == pipeHandle)
    {
        //The request was not successful
    }
    //we got the pipe handle for the endpoint

    //Pass the pipe handle to USB_DEVICE_GEN_DRIVER_ReadEp
    genErr = USB_DEVICE_GEN_DRIVER_ReadEp( pipeHandle,&readBufferObject);
    if(USB_ERROR_OK == genErr)
    {
        //read request is accepted
        //if app callback is registered, rest needs to be handled
        //in the appcallback, which is called by the Generic USB device function
        //driver when it is intimated by the device layer about
        //the transfer.
    }
    else
    {
        // something wrong in the read request
        // handle the error case
        switch ( genErr )
        {
            case USB_ERROR_INVALID_HANDLE:
                break;
            case USB_ERROR_INVALID_BUFFER:
                break;
            case USB_ERROR_BUSY:
                break;
        }
    }

    return 0;
}

```

The read request code flow can be better understood with the following diagram.



Write to the Host:

The application should make use of the API `USB_DEVICE_GEN_DRIVER_WriteEp` to get transfer data to the host. Before placing a write request with the Generic function driver the application should get the pipeHandle using `USB_DEVICE_GEN_DRIVER_GetPipeHandle`. If the received is valid, application has to check if it can place the next write request. This status can be obtained using the API `USB_DEVICE_GEN_DRIVER_TxRx_StatusGet`. If this API returns `USB_DEVICE_GEN_DRIVER_INSTANCE_TX_READY` then the application can place a request for a write operation. If the received status is anything else then the application should poll until the transmit status is ready. The following example shows the usage of read request.

Example: Data Read Request

```
#define APP_USB_DEVICE_GEN_DRIVER_FUNC_INDEX 0
// create buffer. Assuming data end-point max packet size is 64 bytes.
uint8_t inBuffer[64];

uint8_t datalen = 64;

uint8_t epNum = 1;

uint8_t dirn = USB_DEVICE_GEN_DRIVER_EP_IN_INDEX; // OUT : 0x00 or IN : 0x01

DRV_HANDLE pipeHandle, genHandle;

//create the buffer object with data length 64
USB_DEVICE_GEN_DRIVER_DATA_BUFFER_OBJECT writeBufferObject = {&inBuffer, 64, 0};

//open the Generic USB device driver instance 0 ( or what ever is applicable)
genHandle = USB_DEVICE_GEN_DRIVER_DRIVER_Open (APP_USB_DEVICE_GEN_DRIVER_FUNC_INDEX);

if(USB_DEVICE_GEN_DRIVER_DRIVER_RegisterCallbacks (genHandle, app_gen_Callback ))
{
    //handle error
}

//Get the pipe handle for the endpoint
pipeHandle = USB_DEVICE_GEN_DRIVER_GetPipeHandle(genHandle, epNum, dirn)

if ( DRV_HANDLE_INVALID == pipeHandle)
{
    //The request was not successful.
}
//we got the pipe handle for the endpoint
//Pass the pipe handle to USB_DEVICE_GEN_DRIVER_WriteEp

genErr = USB_DEVICE_GEN_DRIVER_WriteEp( pipeHandle, &writeBufferObject);
if(USB_ERROR_OK != genErr)
{
    //write request is not accepted
}

}
```

If the application has registered a callback function with the Generic function driver then any events related to the write request will be informed to the application via this callback. In the callback the application can check whether all of the data requested is transmitted. The following example shows this.

Example: Data Read Request with application callback

```
#define APP_USB_DEVICE_GEN_DRIVER_FUNC_INDEX 0

// Application callback to handle relevant events.
void app_gen_Callback ( SYS_MODULE_INDEX funcDriverIndex,
                        USB_DEVICE_GEN_DRIVER_CALLBACK_TYPE callback,
                        USB_DEVICE_GEN_DRIVER_CALLBACK_EVENT event)
```

```

{
    switch (callback)
    {
        case USB_DEVICE_GEN_DRIVER_CALLBACK_TX:
            if (writeBufferObject.transferCount == writeBufferObject.dataLen)
            {
                //entire data is written to host.
            }
            else
            {
                //some of the data is not sent to the host.
                //think and get some logic in place to handle this
            }
            break;
        case USB_DEVICE_GEN_DRIVER_CALLBACK_RX:
            break;
        case USB_DEVICE_GEN_DRIVER_CALLBACK_MANAGEMENT:
            break;
        default:
            break;
    }
}

int main ()
{
    // create buffer. Assuming data end-point max packet size is 64 bytes.
    uint8_t inBuffer[64] ;

    uint8_t datalen = 64;

    uint8_t epNum = 1;

    uint8_t dirn = USB_DEVICE_GEN_DRIVER_EP_OUT_INDEX;// OUT :0x00 or IN : 0x01

    DRV_HANDLE pipeHandle,genHandle;

    //create the buffer object with datalength 64
    USB_DEVICE_GEN_DRIVER_DATA_BUFFER_OBJECT writeBufferObject = {&inBuffer, 64, 0};

    //open the Generic USB device driver instance 0 ( or what ever is applicable)
    genHandle = USB_DEVICE_GEN_DRIVER_DRIVER_Open ( 0 );

    if(USB_DEVICE_GEN_DRIVER_DRIVER_RegisterCallbacks (genHandle,app_gen_CallBack ))
    {
        //handle error
    }

    //Get the pipe handle for the endpoint
    pipeHandle = USB_DEVICE_GEN_DRIVER_GetPipeHandle(genHandle, epNum, dirn))

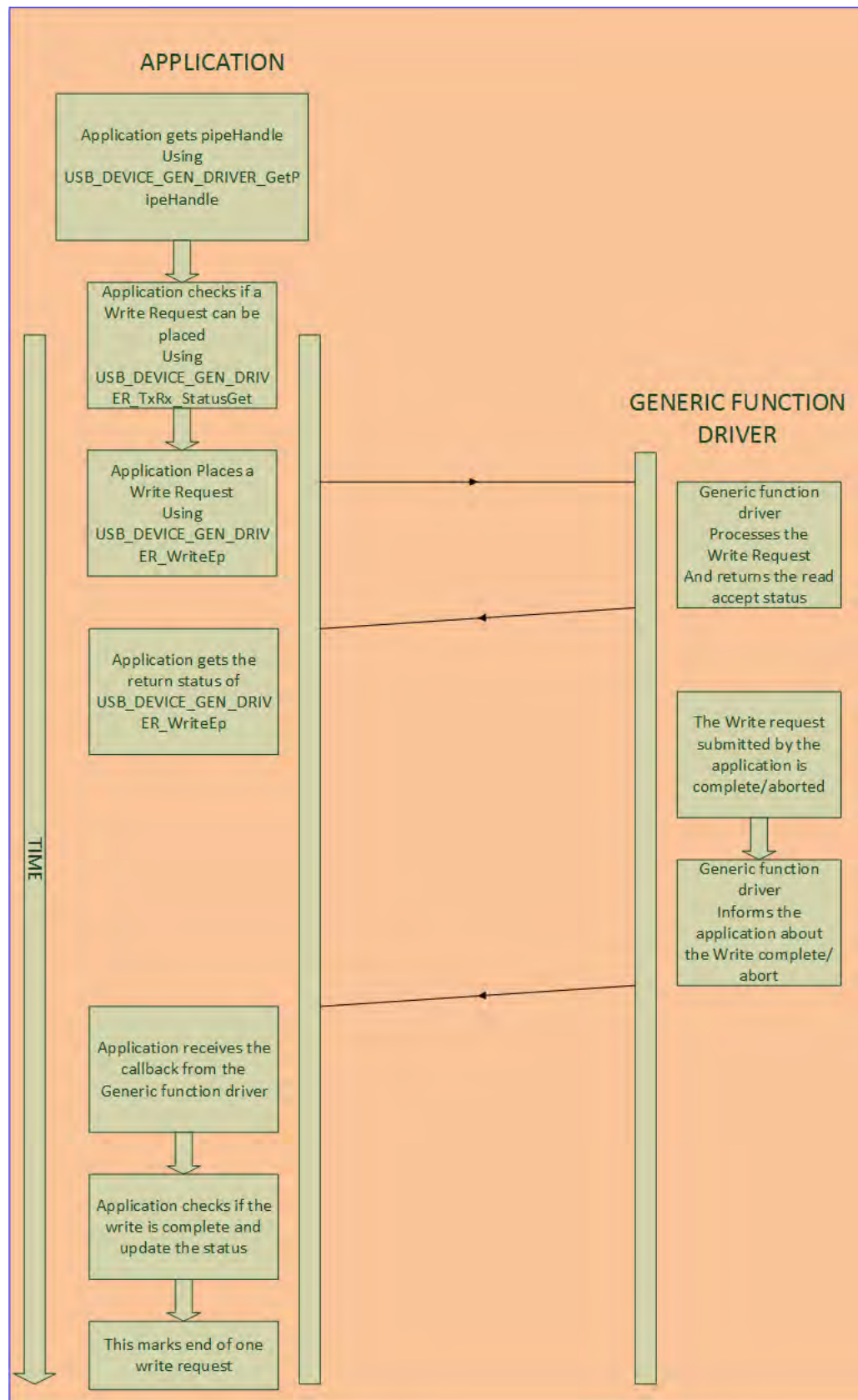
    if ( DRV_HANDLE_INVALID == pipeHandle)
    {
        //The request was not successful.
    }
    //we got the pipe handle for the endpoint
    //Pass the pipe handle to USB_DEVICE_GEN_DRIVER_WriteEp

    genErr = USB_DEVICE_GEN_DRIVER_WriteEp( pipeHandle,&writeBufferObject);
    if(USB_ERROR_OK != genErr)
    {
        //write request is not accepted
    }
}

```

```
    }  
    return 0;  
}
```

The write request code flow can be better understood with the following diagram.



5.9.5.5.5 Client Subclass Specific

This section details the vendor specific sub-class specific requests and notifications handled by the Generic function driver.

5.9.5.6 Library Interface

Data Types and Constants

	Name	Description
	<u>USB_DEVICE_GENERIC_EVENT</u>	This is type <code>USB_DEVICE_GENERIC_EVENT</code> .
	<u>USB_DEVICE_GENERIC_EVENT_DATA</u>	Union that holds event data for different generic driver events.
	<u>USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST</u>	The data-type that holds the event data associated with <code>USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST</code> event.
	<u>USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_READ_COMPLETE</u>	The data-type that holds the event data associated with <code>USB_DEVICE_GENERIC_EVENT_ENDPOINT_READ_COMPLETE</code> event.
	<u>USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_WRITE_COMPLETE</u>	The data-type that holds the event data associated with <code>USB_DEVICE_GENERIC_EVENT_ENDPOINT_WRITE_COMPLETE</code> event.
	<u>USB_DEVICE_GENERIC_EVENT_HANDLER</u>	The data-type that defines the signature of the event callback function.
	<u>USB_DEVICE_GENERIC_EVENT_RESPONSE</u>	The return type of the application callback function.
	<u>USB_DEVICE_GENERIC_INDEX</u>	Data-type for USB device generic driver index.
	<u>USB_DEVICE_GENERIC_RESULT</u>	The enumerated data-type that identifies the possible result of an operation.

	USB_DEVICE_GENERIC_TRANSFER_FLAG	Enumerated data-type that identifies the USB device generic function driver's transfer flag.
	USB_DEVICE_GENERIC_TRANSFER_HANDLE	Data-type of the generic driver transfer handle.

Functions

	Name	Description
	USB_DEVICE_GENERIC_EndpointIsEnabled	This function returns true if the requested endpoint is enabled.
	USB_DEVICE_GENERIC_EndpointRead	Reads data received from host on the requested endpoint.
	USB_DEVICE_GENERIC_EndpointStall	This function stalls the requested endpoint.
	USB_DEVICE_GENERIC_EndpointStallClear	Clears STALL on the endpoint.
	USB_DEVICE_GENERIC_EndpointStallStatusGet	USB Generic function driver library function that returns true if STALL is enabled on the endpoint.
	USB_DEVICE_GENERIC_EndpointWrite	This function allows the application client to transfer data from device to host on the requested endpoint.
	USB_DEVICE_GENERIC_EventHandlerSet	This function allows the application client to register event handler callback with the generic function driver library.

Description

This section describes the Application Programming Interface (API) functions of the USB Generic Device Library.

Refer to each section for a detailed description.

5.9.5.6.1 Functions

5.9.5.6.1.1 USB_DEVICE_GENERIC_EndpointIsEnabled Function

C

```
bool USB_DEVICE_GENERIC_EndpointIsEnabled(
    USB_DEVICE_GENERIC_INDEX iGEN,
    USB_ENDPOINT endpoint
);
```

Preconditions

USB device layer be initialized.

Parameters

Parameters	Description
iGEN	USB generic driver function driver instance ID.
endpoint	endpoint address.

Returns

true if endpoint is enabled.

Remarks

None.

Example

5.9.5.6.1.2 USB_DEVICE_GENERIC_EndpointRead Function

C

```
USB_DEVICE_GENERIC_RESULT USB_DEVICE_GENERIC_EndpointRead(  
    USB_DEVICE_GENERIC_INDEX iGEN,  
    USB_DEVICE_GENERIC_TRANSFER_HANDLE * transferHandle,  
    USB_ENDPOINT endpoint,  
    uint8_t * buffer,  
    size_t bufferSize  
);
```

Preconditions

USB device layer be initialized.

Parameters

Parameters	Description
iGEN	USB generic function driver instance ID.
transferHandle	Pointer to transfer handle.
endpoint	endpoint address
buffer	Buffer pointer
bufferSize	Buffer size

Returns

See [USB_DEVICE_GENERIC_RESULT](#).

Remarks

None.

Example

5.9.5.6.1.3 USB_DEVICE_GENERIC_EndpointStall Function

C

```
USB_DEVICE_GENERIC_RESULT USB_DEVICE_GENERIC_EndpointStall(  
    USB_DEVICE_GENERIC_INDEX iGEN,  
    USB_ENDPOINT endpoint  
);
```

Preconditions

USB device layer must be initialized.

Parameters

Parameters	Description
iGEN	USB device generic function driver instance ID.
endpoint	Endpoint address.

Returns

See [USB_DEVICE_GENERIC_RESULT](#)

Remarks

None.

Example**5.9.5.6.1.4 USB_DEVICE_GENERIC_EndpointStallClear Function****C**

```
USB_DEVICE_GENERIC_RESULT USB_DEVICE_GENERIC_EndpointStallClear(  
    USB_DEVICE_GENERIC_INDEX iGEN,  
    USB_ENDPOINT endpoint  
);
```

Preconditions

USB device layer be initialized.

Parameters

Parameters	Description
iGEN	Generic driver instance index.
endpoint	Endpoint address for which the stall has to be cleared.

Returns

See [USB_DEVICE_GENERIC_RESULT](#).

Remarks

None.

Example**5.9.5.6.1.5 USB_DEVICE_GENERIC_EndpointStallStatusGet Function****C**

```
bool USB_DEVICE_GENERIC_EndpointStallStatusGet(  
    USB_DEVICE_GENERIC_INDEX iGEN,  
    USB_ENDPOINT endpoint  
);
```

Preconditions

USB device layer be initialized.

Parameters

Parameters	Description
iGEN	USB generic function driver instance ID.
endpoint	endpoint address.

Returns

true if endpoint is stalled.

Remarks

None.

Example

5.9.5.6.1.6 USB_DEVICE_GENERIC_EndpointWrite Function

C

```
USB_DEVICE_GENERIC_RESULT USB_DEVICE_GENERIC_EndpointWrite(  
    USB_DEVICE_GENERIC_INDEX iGEN,  
    USB_DEVICE_GENERIC_TRANSFER_HANDLE * transferHandle,  
    USB_ENDPOINT endpoint,  
    uint8_t * buffer,  
    size_t bufferSize,  
    USB_DEVICE_GENERIC_TRANSFER_FLAG flags  
);
```

Preconditions

USB device layer be initialized.

Parameters

Parameters	Description
iGEN	USB generic function driver instance ID.
transferHandle	Pointer to transfer handle.
endpoint	endpoint address
buffer	Buffer pointer
bufferSize	Buffer size
flags	Flag that specifies this function whether to end the transfer with a short packet or not.

Returns

See [USB_DEVICE_GENERIC_RESULT](#).

Remarks

None.

Example

5.9.5.6.1.7 USB_DEVICE_GENERIC_EventHandlerSet Function

C

```
USB_DEVICE_GENERIC_RESULT USB_DEVICE_GENERIC_EventHandlerSet(  
    USB_DEVICE_GENERIC_INDEX iGEN,  
    USB_DEVICE_GENERIC_EVENT_HANDLER eventHandler,  
    uintptr_t userData  
);
```

Preconditions

USB device layer be initialized.

Parameters

Parameters	Description
instanceIndex	USB Generic function driver instance ID.
eventHandler	Pointer to application event handler callback function.

Returns

See [USB_DEVICE_GENERIC_RESULT](#).

Remarks

None.

Example

5.9.5.6.2 Data Types and Constants

5.9.5.6.2.1 USB_DEVICE_GENERIC_EVENT Enumeration

C

```
typedef enum {
    USB_DEVICE_GENERIC_EVENT_CONTROL_TRANSFER_SETUP_REQUEST,
    USB_DEVICE_GENERIC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED,
    USB_DEVICE_GENERIC_EVENT_CONTROL_TRANSFER_DATA_SENT,
    USB_DEVICE_GENERIC_EVENT_CONTROL_TRANSFER_ABORTED,
    USB_DEVICE_GENERIC_EVENT_ENDPOINT_WRITE_COMPLETE,
    USB_DEVICE_GENERIC_EVENT_ENDPOINT_READ_COMPLETE
} USB_DEVICE_GENERIC_EVENT;
```

Description

This is type USB_DEVICE_GENERIC_EVENT.

Members

Members	Description
USB_DEVICE_GENERIC_EVENT_CONTROL_TRANSFER_SETUP_REQUEST	Control transfer data stage is received from host to device
USB_DEVICE_GENERIC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	Control transfer data stage is received from host to device
USB_DEVICE_GENERIC_EVENT_CONTROL_TRANSFER_DATA_SENT	Control transfer data stage is transmitted from device to host

5.9.5.6.2.2 USB_DEVICE_GENERIC_EVENT_DATA Union

C

```
typedef union {
    USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_WRITE_COMPLETE endpointWriteComplete;
    USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_READ_COMPLETE endpointReadComplete;
    USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST * ptrSetupRequest;
} USB_DEVICE_GENERIC_EVENT_DATA;
```

Description

Event Data.

Members

Members	Description
USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_WRITE_COMPLETE endpointWriteComplete;	Data associated with USB_DEVICE_GENERIC_EVENT_ENDPOINT_WRITE_COMPLETE event
USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_READ_COMPLETE endpointReadComplete;	Data associated with USB_DEVICE_GENERIC_EVENT_ENDPOINT_READ_COMPLETE event

USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST * ptrSetupRequest;	Data associated with USB_DEVICE_GENERIC_EVENT_D ATACONTROL

Remarks

None.

5.9.5.6.2.3**USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST
Structure****C**

```
typedef struct {
    uint8_t bmRequestType;
    uint8_t bRequest;
    uint16_t wValue;
    uint16_t wIndex;
    uint16_t wLength;
} USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST;
```

Description

Data associated with USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST event.

Members

Members	Description
uint8_t bmRequestType;	Format of setup packet from table 9-2 of USB 2.0 spec

Remarks

None.

**5.9.5.6.2.4 USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_READ_COMPLETE
Structure****C**

```
typedef struct {
    unsigned int dataSize;
    uint8_t * data;
    USB_DEVICE_IRP_STATUS status;
    USB_DEVICE_GENERIC_TRANSFER_HANDLE transferHandle;
} USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_READ_COMPLETE;
```

Description

Data associated with USB_DEVICE_GENERIC_EVENT_ENDPOINT_READ_COMPLETE event.

Members

Members	Description
unsigned int dataSize;	data size received
uint8_t * data;	Pointer to data buffer
USB_DEVICE_IRP_STATUS status;	Status
USB_DEVICE_GENERIC_TRANSFER_HANDLE transferHandle;	Transfer handle

Remarks

None.

5.9.5.6.2.5 USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_WRITE_COMPLETE Structure

C

```
typedef struct {
    unsigned int dataSize;
    uint8_t * data;
    USB_DEVICE_IRP_STATUS status;
    USB_DEVICE_GENERIC_TRANSFER_HANDLE transferHandle;
} USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_WRITE_COMPLETE;
```

Description

Data associated with USB_DEVICE_GENERIC_EVENT_ENDPOINT_WRITE_COMPLETE event.

Members

Members	Description
unsigned int dataSize;	data size transmitted
uint8_t * data;	Pointer to data buffer
USB_DEVICE_IRP_STATUS status;	Status
USB_DEVICE_GENERIC_TRANSFER_HANDLE transferHandle;	Transfer handle

Remarks

None.

5.9.5.6.2.6 USB_DEVICE_GENERIC_EVENT_HANDLER Type

C

```
typedef USB_DEVICE_GENERIC_EVENT_RESPONSE (*
USB_DEVICE_GENERIC_EVENT_HANDLER)(USB_DEVICE_GENERIC_INDEX instanceIndex,
USB_DEVICE_CONTROL_TRANSFER_HANDLE controlTransferHandle, USB_DEVICE_GENERIC_EVENT event,
USB_DEVICE_GENERIC_EVENT_DATA * eventData, uintptr_t userData);
```

Description

USB device generic function driver event handle.

The application provided event callback function must be of this data-type.

Remarks

None.

5.9.5.6.2.7 USB_DEVICE_GENERIC_EVENT_RESPONSE Type

C

```
typedef void USB_DEVICE_GENERIC_EVENT_RESPONSE;
```

Description

Return type of the application callback function.

Remarks

Also see, [USB_DEVICE_GENERIC_EVENT_HANDLER](#)

5.9.5.6.2.8 USB_DEVICE_GENERIC_INDEX Type

C

```
typedef SYS_MODULE_INDEX USB_DEVICE_GENERIC_INDEX;
```

Description

Data-type for USB device generic driver index.

Remarks

None.

5.9.5.6.2.9 USB_DEVICE_GENERIC_RESULT Enumeration

C

```
typedef enum {  
    USB_DEVICE_GENERIC_RESULT_QUEUE_FULL,  
    USB_DEVICE_GENERIC_RESULT_OK,  
    USB_DEVICE_GENERIC_RESULT_ENDPOINT_NOT_CONFIGURED,  
    USB_DEVICE_GENERIC_RESULT_PARAMETER_INVALID  
} USB_DEVICE_GENERIC_RESULT;
```

Description

Possible results of Generic Driver operations.

Members

Members	Description
USB_DEVICE_GENERIC_RESULT_QUEUE_FULL	Queue is full
USB_DEVICE_GENERIC_RESULT_OK	No Error
USB_DEVICE_GENERIC_RESULT_ENDPOINT_NOT_CONFIGURED	Endpoint not configured
USB_DEVICE_GENERIC_RESULT_PARAMETER_INVALID	One or more parameter/s of the function is invalid

Remarks

None.

5.9.5.6.2.10 USB_DEVICE_GENERIC_TRANSFER_FLAG Enumeration

C

```
typedef enum {  
    USB_DEVICE_GENERIC_TRANSFER_FLAG_NONE,  
    USB_DEVICE_GENERIC_TRANSFER_FLAG_DATA_COMPLETE  
} USB_DEVICE_GENERIC_TRANSFER_FLAG;
```

Description

USB device generic function driver transfer flags.

Members

Members	Description
USB_DEVICE_GENERIC_TRANSFER_FLAG_NONE	Transfer flag that specifies no special request
USB_DEVICE_GENERIC_TRANSFER_FLAG_DATA_COMPLETE	Transfer flag that specifies the generic driver to complete the transfer

Remarks

Also see, [USB_DEVICE_GENERIC_EndpointWrite](#)

5.9.5.6.2.11 USB_DEVICE_GENERIC_TRANSFER_HANDLE Type

C

```
typedef uintptr_t USB_DEVICE_GENERIC_TRANSFER_HANDLE;
```

Description

Generic driver transfer handle.

Remarks

Also see, [USB_DEVICE_GENERIC_EndpointRead](#) [USB_DEVICE_GENERIC_EndpointWrite](#)

5.9.5.7 Files

Files

Name	Description
usb_device_generic.h	Generic USB device function driver header

Description

5.9.5.7.1 usb_device_generic.h

Generic USB device function driver header file

Generic USB device function driver header file

Enumerations

	Name	Description
	USB_DEVICE_GENERIC_EVENT	This is type USB_DEVICE_GENERIC_EVENT.
	USB_DEVICE_GENERIC_RESULT	The enumerated data-type that identifies the possible result of an operation.
	USB_DEVICE_GENERIC_TRANSFER_FLAG	Enumerated data-type that identifies the USB device generic function driver's transfer flag.

Functions

	Name	Description
	USB_DEVICE_GENERIC_EndpointIsEnabled	This function returns true if the requested endpoint is enabled.
	USB_DEVICE_GENERIC_EndpointRead	Reads data received from host on the requested endpoint.
	USB_DEVICE_GENERIC_EndpointStall	This function stalls the requested endpoint.
	USB_DEVICE_GENERIC_EndpointStallClear	Clears STALL on the endpoint.
	USB_DEVICE_GENERIC_EndpointStallStatusGet	USB Generic function driver library function that returns true if STALL is enabled on the endpoint.
	USB_DEVICE_GENERIC_EndpointWrite	This function allows the application client to transfer data from device to host on the requested endpoint.
	USB_DEVICE_GENERIC_EventHandlerSet	This function allows the application client to register event handler callback with the generic function driver library.

Structures

	Name	Description
	<u>USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST</u>	The data-type that holds the event data associated with USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST event.
	<u>USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_READ_COMPLETE</u>	The data-type that holds the event data associated with USB_DEVICE_GENERIC_EVENT_ENDPOINT_READ_COMPLETE event.
	<u>USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_WRITE_COMPLETE</u>	The data-type that holds the event data associated with USB_DEVICE_GENERIC_EVENT_ENDPOINT_WRITE_COMPLETE event.

Types

	Name	Description
	<u>USB_DEVICE_GENERIC_EVENT_HANDLER</u>	The data-type that defines the signature of the event callback function.
	<u>USB_DEVICE_GENERIC_EVENT_RESPONSE</u>	The return type of the application callback function.
	<u>USB_DEVICE_GENERIC_INDEX</u>	Data-type for USB device generic driver index.
	<u>USB_DEVICE_GENERIC_TRANSFER_HANDLE</u>	Data-type of the generic driver transfer handle.

Unions

	Name	Description
	<u>USB_DEVICE_GENERIC_EVENT_DATA</u>	Union that holds event data for different generic driver events.

File Name

usb_device_generic.h

Company

Microchip Technology Inc.

5.9.6 USB HID Device Library

5.9.6.1 Introduction

Introduces the MPLAB Harmony USB Device HID Library

Description

This library provides a high-level abstraction of the Human Interface Devices (HID) class under the Universal Serial Bus (USB) communication with a convenient C language interface. This library supports revision 1.11 of the USB HID specification release by the USB Implementers forum. This library is part of the MPLAB Harmony USB Device stack.

USB Human Interface Devices class (or USB HID) supports class of devices that are used by humans to control the operation of computer systems. The HID class of devices include a wide variety of human interface, data indicator, data feedback devices with various types of outputs directed to the end user. Some common examples of HID class devices include:

- Keyboards
- Pointing devices such as standard mouse, joysticks, trackballs
- Front-panel controls like knobs, switches, buttons, sliders
- Controls found on telephony, gaming or simulation devices like steering wheels, rudder pedals, dial pads
- Data devices such as bar-code scanners, thermometers, analyzers

The USB Device HID library (also referred to as HID function driver in this document) offers services to the application to interact and respond to the host requests. Additional information about the HID class can be obtained from the HID specification available www.usbif.org.

5.9.6.2 Release Notes

MPLAB Harmony Version: v0.70b **USB HID Device Library Version :** 0.01 Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.9.6.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have

paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.9.6.4 Library Architecture

5.9.6.4.1 Architecture Overview

Provides an Architectural Overview of the HID Function Driver

Description

The HID Function Driver offers services to a USB HID device to communicate with the host by abstracting the HID specification details. It must be used along with the USB Device layer and USB controller to communicate with the USB host. Figure 1 shows a block diagram representation of the MPLAB Harmony USB Architecture and where the USB HID Function Driver is placed.

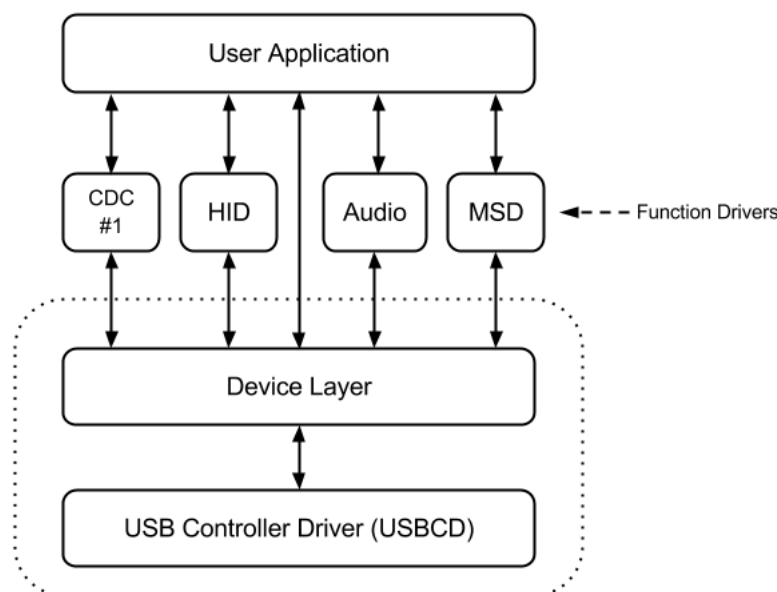


Figure 1: USB HID Function Driver

The HID function driver together with USB Device layer and USB controller driver forms the basic library entity through which a HID device can communicate with the USB host. The USB controller driver takes the responsibility of managing the USB peripheral on the device. The USB device layer handles the device enumeration etc. The USB Device layer forwards all HID specific control transfers to the HID Function Driver. The HID Function Driver interprets the control transfers and requests application's intervention through event handlers and well defined set of API. The application must register a event handler with the HID function driver in the Device Layer Set Configuration Event. While the application must respond to the HID function driver events, it can do this either in the event handler or out of it. The application interacts with HID Function Driver routines to [send](#) and receive HID reports over the USB.

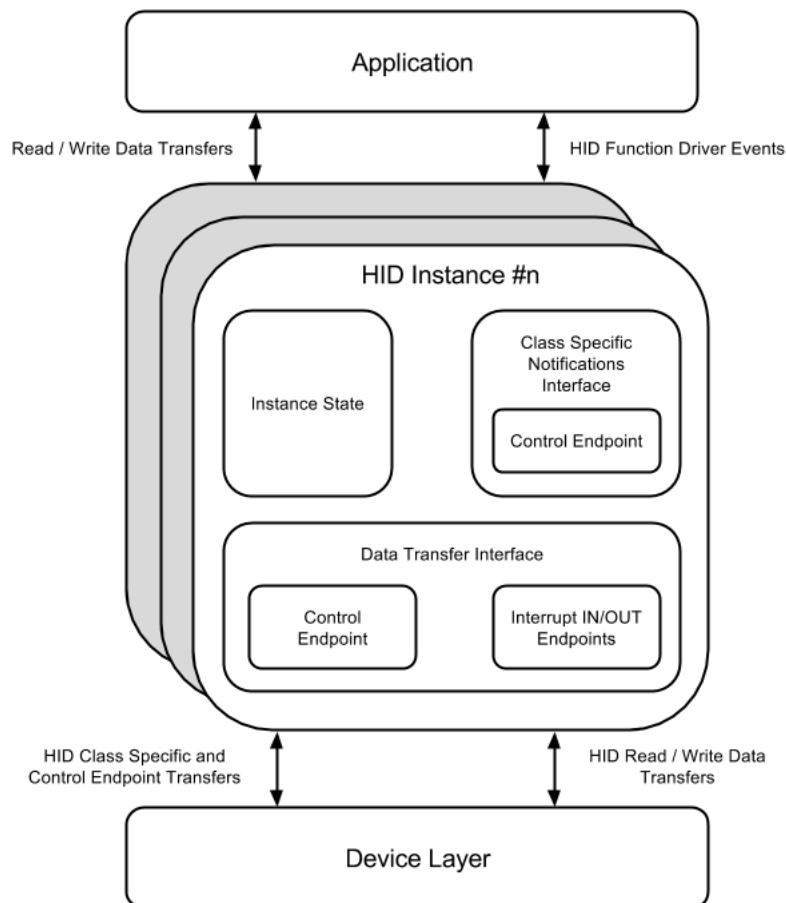


Figure 2 : Architecture of the HID function driver.

Figure 2 shows the architecture of the HID function Driver. The function driver maintains the state of each instance. It receives HID class specific control transfers from the Device Layer. Class specific control transfers that require application response are forwarded to the application as function driver events. Depending on the type of device, the HID function driver can use the control endpoint and/or interrupt endpoints for data transfers. The HID device exchanges data with host through data objects called reports. The report data format is described by the HID report descriptor which is provided to the host when requested. Refer to the HID specification available www.usb.org for more details on the USB HID device class and how reports descriptors can be created. The HID function driver allows report descriptors to be specified for every instance. This allow the application to implement a composite HID device.

5.9.6.5 Using the Library

5.9.6.5.1 Files to Include

Describes which files to include in project while using the HID Function Driver

Description

Table 1 shows the files that must be included in the project in order to use the HID Function Driver. These files are located in the

framework folder of the MPLAB Harmony installation. It is assumed that the Device Layer files (and the files needed by the Device Layer) are already included in the project.

Filename	Description
\framework\usb\src\dynamic\usb_device_hid.c	Contains the HID Control Interface and Data Interface management routines
\framework\usb\usb_device_hid.h	Must included in every source file that needs to invoke HID function driver API.
system_config.h	User created file which contains the HID function driver configuration.

Table 1: Files to be included in USB CDC Device Project.

5.9.6.5.2 Library Configuration

Describes how to configure the HID Function Driver

Description

The following configuration parameters must be defined while using the HID Function Driver. The configuration macros that implement these parameters must be located in the system_config.h file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

Configuration Macro Name	Description	Comments
USB_DEVICE_HID_MAX_INSTANCES	Number of HID Function Driver Instances in the application	This macro defines the number of instances of the HID Function Driver. For example, if the application needs to implement two instances of the HID Function Driver (for example to create a mouse + keyboard device) on one USB Device, the macro USB_DEVICE_HID_INSTANCES_NUMBER should be set to 2. Note that implementing a USB Device that features multiple HID interfaces requires appropriate USB configuration descriptors.
USB_DEVICE_HID_QUEUE_SIZE	HID Function Driver Buffer Queue Combined size	This macro defines the number of entries in all queues in all instances of the HID function driver. This value can be obtained by adding up the read and write queue sizes of each HID Function driver instance. In a simple single instance USB HID device application, that does not require buffer queuing, the USB_DEVICE_HID_QUEUE_DEPTH_COMBINED macro can be set to 2. Consider a case with two HIDfunction driver instances, CDC 1 has a read queue size of 2 and write queue size of 3, HID 2 has a read queue size of 4 and write queue size of 1, this macro should be set to 10 (2 +3 + 4 + 1).

5.9.6.5.3 Library Initialization

Describes how the HID Function Driver is initialized.

Description

The HID function driver instance for a USB device configuration is initialized by the Device Layer when the configuration is set by the host. This process does not require application intervention. Each instance of the HID function driver should be registered with the Device layer through the Device Layer Function Driver Registration Table. The HID function driver requires a initialization data structure that contains details about the report descriptor associated with the specific instance of the HID function driver. The hidFuncDriver object is a global object provided by the HID function driver and points to the HID function driver - Device Layer interface functions which are required by the Device Layer. The code snippet below shows an example of how a HID function driver instance (implementing a USB HID Mouse) can be registered with the Device Layer.

```

/* This code snippet shows an example of registering a HID function driver
 * with the Device Layer. While registering the function driver, an initialization
 * data structure must be specified. In this example, hidInit is the HID function
 * driver initialization data structure. */

/* This hid_rpt01 report descriptor describes a 3 button 2
 * axis mouse pointing device */
const uint8_t hid_rpt01[] =
{
    0x06, 0x00, 0xFF,    // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,          // Usage (Vendor Usage 1)
    0xA1, 0x01,          // Collection (Application)
    0x19, 0x01,          // Usage Minimum
    0x29, 0x40,          // Usage Maximum //64 input usages total (0x01 to 0x40)
    0x15, 0x01,          // Logical Minimum (data bytes in the report may have minimum value =
0x00)
    0x25, 0x40,          // Logical Maximum (data bytes in the report may have maximum value =
0x00FF = unsigned 255)
    0x75, 0x08,          // Report Size: 8-bit field size
    0x95, 0x40,          // Report Count: Make sixty-four 8-bit fields (the next time the
parser hits an "Input", "Output", or "Feature" item)
    0x81, 0x00,          // Input (Data, Array, Abs): Instantiates input packet fields based
on the above report size, count, logical min/max, and usage.
    0x19, 0x01,          // Usage Minimum
    0x29, 0x40,          // Usage Maximum //64 output usages total (0x01 to 0x40)
    0x91, 0x00,          // Output (Data, Array, Abs): Instantiates output packet fields.
Uses same report size and count as "Input" fields, since
nothing // new or different was specified to the
parser since the "Input" item.
    0xC0                // End Collection
};

/* HID Function Driver Initialization data structure. This
 * contains the size of the report descriptor and a pointer
 * to the report descriptor. If there are multiple HID instances
 * each with different report descriptors, then multiple such data
 * structures may be needed */

USB_DEVICE_HID_INITIALIZATION hidInit =
{
    sizeof(hid_rpt01),
    (uint8_t *)&hid_rpt01
};

/* The HID function driver instance is now registered with
 * device layer through the function driver registration
 * table. */

const USB_DEVICE_FUNC_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
        .speed = USB_SPEED_FULL|USB_SPEED_HIGH, // Supported speed
        .configurationValue = 1,                // To be initialized for Configuration 1
        .interfaceNumber = 0,                   // Starting interface number
        .numberOfInterfaces = 1,                 // Number of Interfaces
        .funcDriverIndex = 0,                   // Function Driver instance index is 0
        .funcDriverInit = &hidInit,             // Function Driver Initialization
        .driver = &hidFuncDriver                // Pointer to the function driver - Device
Layer Interface functions
    }
};

```

5.9.6.5.4 Event Handling

Describes HID function driver event handler registration and event handling.

Description

Registering a HID Function Driver Event Handler:

While creating USB HID Device based application, an event handler must be registered with the Device Layer (the Device Layer Event Handler) and every HID function driver instance (HID Function Driver Event Handler). The HID function driver event handler receives HID events. This event handler should be registered before the USB device layer acknowledges the SET CONFIGURATION request from the USB Host. In order to ensure this, the event handler should be set in the USB_DEVICE_EVENT_CONFIGURED event that is generated by the device layer. While registering the HID function driver event handler, the HID function driver allows the application to also pass a data object in the event handler register function. This data object gets associated with the instance of the HID function driver and is returned by the HID function driver when a HID function driver event occurs. The code example below shows an example of how this can be done.

```
/* This a sample Application Device Layer Event Handler
 * Note how the HID Function Driver event handler APP_USBDeviceHIDEventHandler()
 * is registered in the USB_DEVICE_EVENT_CONFIGURED event. The appData
 * object that is passed in the USB_DEVICE_HID_EventHandlerSet()
 * function will be returned as the userData parameter in the
 * when the APP_USBDeviceHIDEventHandler() function is invoked */
```

```
void APP_USBDeviceEventCallBack ( USB_DEVICE_EVENT event,
                                USB_DEVICE_EVENT_DATA * eventData )
{
    switch ( event )
    {
        case USB_DEVICE_EVENT_RESET:
        case USB_DEVICE_EVENT_DECONFIGURED:

            // USB device is reset or device is deconfigured.
            // This means that USB device layer is about to deinitalize
            // all function drivers.

            break;

        case USB_DEVICE_EVENT_CONFIGURED:

            /* check the configuration */
            if ( eventData->eventConfigured.configurationValue == 1)
            {

                /* Register the HID Device application event handler here.
                 * Note how the appData object pointer is passed as the
                 * user data */

                USB_DEVICE_HID_EventHandlerSet(USB_DEVICE_HID_INDEX_0,
                                                APP_USBDeviceHIDEventHandler, (uintptr_t)&appData);

                /* mark that set configuration is complete */
                appData.isConfigured = true;

            }
            break;

        case USB_DEVICE_EVENT_SUSPENDED:

            break;

        case USB_DEVICE_EVENT_RESUMED:
        case USB_DEVICE_EVENT_ATTACHED:
        case USB_DEVICE_EVENT_DETACHED:
        case USB_DEVICE_EVENT_ERROR:
        default:
            break;
    }
}
```



```
}

```

HID Function Driver Events:

The HID Function Driver generates events that the application must respond to. Some of these events are control requests communicated through control transfers. The application must therefore complete the control transfer. Based on the generated event, the application may be required to:

- Respond with a [USB_DEVICE_ControlSend\(\)](#) function which completes the data stage of a Control Read Transfer.
- Respond with a [USB_DEVICE_ControlReceive\(\)](#) function which provisions the data stage of a Control Write Transfer.
- Respond with a [USB_DEVICE_ControlStatus\(\)](#) function which completes the handshake stage of the Control Transfer. The application can either STALL or Acknowledge the handshake stage via the [USB_DEVICE_ControlStatus\(\)](#) function.
- Analyze the pData member of the event handler and check for event specific data.

The possible HID Function Driver events are described here along with the required application response, event specific data and likely follow up function driver event:

1. USB_DEVICE_HID_EVENT_CONTROL_GET_REPORT

Application Response: This event is generated when the HID host is requesting for a report over the control interface. The application must provide the report by calling the [USB_DEVICE_ControlSend\(\)](#) function either in the event handler or in the application (outside of the event handler function). The application must use the controlTransferHandle parameter provided in the event while calling [USB_DEVICE_ControlSend\(\)](#) function. The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (eventData): The getReport member of the eventData data structure will contain the details about the requested report.

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. This indicates that the data was sent to the host successfully. The application must acknowledge the handshake stage of the control transfer by calling [USB_DEVICE_ControlStatus\(\)](#) function with the USB_DEVICE_CONTROL_STATUS_SEND_ZLP flag.

2. USB_DEVICE_HID_EVENT_CONTROL_SET_REPORT

Application Response: This event is generated when the HID host wants to [send](#) a report over the control interface. The application must provide a buffer to receive the report by calling the [USB_DEVICE_ControlReceive\(\)](#) function either in the event handler or in the application (outside of the event handler function). The application must use the controlTransferHandle parameter provided in the event while calling [USB_DEVICE_ControlReceive\(\)](#) function. The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event to track completion of the command.

Event Specific Data (eventData): The setReport member of the eventData data structure will contain the details about the report that the host intends to [send](#).

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. This indicates that the data was received successfully. The application must either acknowledge or stall the handshake stage of the control transfer by calling [USB_DEVICE_ControlStatus\(\)](#) function with USB_DEVICE_CONTROL_STATUS_SEND_ZLP or USB_DEVICE_CONTROL_STATUS_STALL flag respectively.

3. USB_DEVICE_HID_EVENT_CONTROL_GET_IDLE

Application Response: This event is generated when the HID host wants to read the current idle rate for the specified report. The application must provide the idle rate through the [USB_DEVICE_ControlSend\(\)](#) function either in the event handler or in the application (outside of the event handler function). The application must use the controlTransferHandle parameter provided in the event while calling [USB_DEVICE_ControlSend\(\)](#) function. The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (eventData): The getIdle member of the eventData data structure will contain the details about the report for which the idle rate is requested.

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. This indicates that the data was sent to the host successfully. The application must acknowledge the handshake stage of the control transfer by calling [USB_DEVICE_ControlStatus\(\)](#) function with the USB_DEVICE_CONTROL_STATUS_SEND_ZLP flag.

4. USB_DEVICE_HID_EVENT_CONTROL_SET_IDLE

Application Response: This event is generated when the HID host sends a Set Idle request to the device . The application must inspect the eventData and determine if the idle rate is to be supported. The application must either acknowledge (if the idle rate is supported) or stall the handshake stage of the control transfer (if the idle rate is not supported) by calling [USB_DEVICE_ControlStatus\(\)](#) function with USB_DEVICE_CONTROL_STATUS_SEND_ZLP or USB_DEVICE_CONTROL_STATUS_STALL flag respectively.

Event Specific Data (eventData): The setIdle member of the eventData data structure will contain the details about the report for which the idle rate should be set and the idle duration.

Likely Follow-up event: None.

5. USB_DEVICE_HID_EVENT_CONTROL_SET_PROTOCOL

Application Response: This event is generated when the HID host sends a Set Protocol request to the device . The application must inspect the eventData and determine if the protocol is to be supported. The application must either acknowledge (if the protocol is supported) or stall the handshake stage of the control transfer (if the protocol is not supported) by calling [USB_DEVICE_ControlStatus\(\)](#) function with USB_DEVICE_CONTROL_STATUS_SEND_ZLP or USB_DEVICE_CONTROL_STATUS_STALL flag respectively.

Event Specific Data (eventData): The setProtocol member of the eventData data structure will contain the details about the protocol to switch to.

Likely Follow-up event: None.

6. USB_DEVICE_HID_EVENT_CONTROL_GET_PROTOCOL

Application Response: This event is generated when the HID host issues a Get Protocol Request. The application must provide the current protocol through the [USB_DEVICE_ControlSend\(\)](#) function either in the event handler or in the application (outside of the event handler function). The application must use the controlTransferHandle parameter provided in the event while calling [USB_DEVICE_ControlSend\(\)](#) function. The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (eventData): None.

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. This indicates that the data was sent to the host successfully. The application must acknowledge the handshake stage of the control transfer by calling [USB_DEVICE_ControlStatus\(\)](#) function with the USB_DEVICE_CONTROL_STATUS_SEND_ZLP flag.

7. USB_DEVICE_HID_EVENT_CONTROL_SET_DESCRIPTOR

Application Response: This event is generated when the HID host issues a Set Descriptor request. The application must provide a buffer to receive the descriptor through the [USB_DEVICE_ControlReceive\(\)](#) function either in the event handler or in the application (outside of the event handler function). The application must use the controlTransferHandle parameter provided in the event while calling [USB_DEVICE_ControlReceive\(\)](#) function. The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event to track completion of the command.

Event Specific Data: None

Likely Follow-up event: This event will likely be followed by the `USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED` event. This indicates that the data was received successfully. The application must either acknowledge or stall the handshake stage of the control transfer by calling `USB_DEVICE_ControlStatus()` function with `USB_DEVICE_CONTROL_STATUS_SEND_ZLP` or `USB_DEVICE_CONTROL_STATUS_STALL` flag respectively.

8. `USB_DEVICE_HID_EVENT_REPORT_SENT`

Application Response: This event occurs when a report `send` operation scheduled by calling the `USB_DEVICE_HID_ReportSend()` function has completed. This event does not require the application to respond with any function calls..

Event Specific Data (pData): The `reportSent` member of the `eventData` data structure will contains details about the report `send` operation.

Likely Follow-up event: None.

9. `USB_DEVICE_HID_EVENT_REPORT_RECEIVED`

Application Response: This event occurs when a report receive operation scheduled by calling the `USB_DEVICE_HID_ReportReceive()` function has completed. This event does not require the application to respond with any function calls..

Event Specific Data (pData): The `reportReceived` member of the `eventData` data structure will contains details about the report receive operation.

Likely Follow-up event: None

10. `USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT`

Application Response: This event occurs when the data stage of a control read transfer has completed in response to the `USB_DEVICE_ControlSend()` function. The application must acknowledge the handshake stage of the control transfer by calling `USB_DEVICE_ControlStatus()` function with `USB_DEVICE_CONTROL_STATUS_SEND_ZLP` flag.

Event Specific Data (pData): None.

Likely Follow-up event: None.

11. `USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED`

Application Response: This event occurs when the data stage of a control write transfer has completed in response to the `USB_DEVICE_ControlReceive()` function. The application must either acknowledge or stall the handshake stage of the control transfer by calling `USB_DEVICE_ControlStatus()` function with `USB_DEVICE_CONTROL_STATUS_SEND_ZLP` or `USB_DEVICE_CONTROL_STATUS_STALL` flag respectively.

Event Specific Data (pData): None

Likely Follow-up event: None.

12. `USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED`

Application Response: This event occurs when the a control transfer request is aborted by the host. The application can use this event to update its HID class specific control transfer state machine.

Event Specific Data (pData): None

Likely Follow-up event: None.

5.9.6.5.5 Sending a Report

Describes how to `send` a report.

Description

The USB HID device sends data to the HID host as reports. The HID device application should use the `USB_DEVICE_HID_ReportSend()` function to **send** the report. This function returns a transfer handler that allows the application to track the read request. The request is completed when the host as requested for the data. A report **send** request could fail if the function driver instance transfer queue is full. The completion of the write transfer is indicated by `USB_DEVICE_HID_EVENT_REPORT_SENT` event. The transfer handle and the amount of data sent is returned in the `reportSent` member of the `eventData` data structure along with the event.

The following code snippet shows an example of how USB HID Mouse application sends a report to the host.

```
/* In this code example, the application uses the
 * USB_HID_MOUSE_ReportCreate to create the mouse report
 * and then uses the USB_DEVICE_HID_ReportSend() function
 * to send the report */

USB_HID_MOUSE_ReportCreate(appData.xCoordinate, appData.yCoordinate,
    appData.mouseButton, &appData.mouseReport);

/* Send the mouse report. */
USB_DEVICE_HID_ReportSend(appData.hidInstance,
    &appData.reportTransferHandle, (uint8_t*)&appData.mouseReport,
    sizeof(USB_HID_MOUSE_REPORT));
```

5.9.6.5.6 Receiving a Report

Describes how to receive a report

Description

The application can receive a report from the host by using the `USB_DEVICE_HID_ReportReceive()` function. This function returns a transfer handler that allows the application to track the read request. The request is completed when the host sends the report. The application must make sure that it allocates a buffer size that is at least the size of the report. The return value of the function indicates the success of the request. A read request could fail if the function driver transfer queue is full. The completion of the read transfer is `USB_DEVICE_HID_EVENT_REPORT_RECEIVED` event. The `reportReceived` member of the `eventData` data structure contains details about the received report. The following code snippet shows an example of how a USB HID Keyboard can schedule a receive report operation to get the keyboard LED status.

```
/* The following code snippet shows how the
 * USB HID Keyboard application schedules a
 * receive report operation to receive the
 * keyboard output report from the host. This
 * report contains the keyboard LED status. The
 * size of the report is 1 byte */

result = USB_DEVICE_HID_ReportReceive(appData.hidInstance,
    &appData.receiveTransferHandle,
    (uint8_t *)&appData.keyboardOutputReport,1);

if(USB_DEVICE_HID_RESULT_OK != result)
{
    /* Do error handling here */
}
```

5.9.6.6 Library Interface

Data Types and Constants

	Name	Description
	<u>USB_DEVICE_HID_CONTROL_STATUS</u>	USB Device HID Function Driver Control Transfer Status
	<u>USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE</u>	USB Device HID Function Driver Control Transfer Handle
	<u>USB_DEVICE_HID_EVENT</u>	USB Device HID Function Driver Events
	<u>USB_DEVICE_HID_EVENT_DATA_GET_IDLE</u>	USB Device HID Get Idle Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_GET_PROTOCOL</u>	USB Device HID Get Protocol Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_GET_REPORT</u>	USB Device HID Get Report Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED</u>	USB Device HID Report Received Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_REPORT_SENT</u>	USB Device HID Report Sent Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR</u>	USB Device HID Set Descriptor Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_SET_IDLE</u>	USB Device HID Set Idle Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL</u>	USB Device HID Set Protocol Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_SET_REPORT</u>	USB Device HID Set Report Event Data Type.
	<u>USB_DEVICE_HID_EVENT_HANDLER</u>	USB Device HID Event Handler Function Pointer Type.
	<u>USB_DEVICE_HID_EVENT_RESPONSE</u>	USB Device HID Function Driver Event Callback Response Type
	<u>USB_DEVICE_HID_INDEX</u>	USB device HID Function Driver Index.
	<u>USB_DEVICE_HID_INIT</u>	USB Device HID Function Driver Initialization Data Structure
	<u>USB_DEVICE_HID_RESULT</u>	USB Device HID Function Driver USB Device HID Result enumeration.
	<u>USB_DEVICE_HID_TRANSFER_HANDLE</u>	USB Device HID Function Driver Transfer Handle Definition.
	<u>USB_DEVICE_HID_EVENT_RESPONSE_NONE</u>	USB Device HID Function Driver Event Handler Response Type None.
	<u>USB_DEVICE_HID_INSTANCES_NUMBER</u>	Specifies the number of HID instances.
	<u>USB_DEVICE_HID_QUEUE_DEPTH_COMINED</u>	DOM-IGONORE-BEGIN
	<u>USB_DEVICE_HID_TRANSFER_HANDLE_INVALID</u>	USB Device HID Function Driver Invalid Transfer Handle Definition.

Functions

	Name	Description
	<u>USB_DEVICE_HID_ControlReceive</u>	This function allows the application to respond to the HID function driver specific control transfer requests. It allows the application to receive data from the host in the data stage of the control transfer.
	<u>USB_DEVICE_HID_ControlSend</u>	This function allows the application to respond to the HID function driver specific control transfer requests. It allows the application to send data to the host in the data stage of the control transfer.
	<u>USB_DEVICE_HID_ControlStatus</u>	This function allows the application to complete the status stage of the the HID Function Driver specific control transfer request.
	<u>USB_DEVICE_HID_EventHandlerSet</u>	This function registers a event handler for the specified HID function driver instance.

	USB_DEVICE_HID_ReportReceive	This function submits the buffer to HID function driver library to receive a report from host to device.
	USB_DEVICE_HID_ReportSend	This function submits the buffer to HID function driver library to send a report from device to host.

Description

This section describes the Application Programming Interface (API) functions of the USB Device HID library.

Refer to each section for a detailed description.

5.9.6.6.1 Functions**5.9.6.6.1.1 USB_DEVICE_HID_ControlReceive Function****C**

```

USB_DEVICE_HID_RESULT USB_DEVICE_HID_ControlReceive(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * data,
    size_t size
);

```

Description

This function allows the application to respond to the HID function driver specific control transfer requests. It allows the application to receive data from the host in the data stage of the control transfer. For example, the [USB_DEVICE_HID_EVENT_DATA_SET_REPORT](#) event requires a control transfer response. The function can be called in the HID Function Driver event handler or can be called outside the event handler. Calling this function outside the event handler defers the response to the event. This allows the application to prepare the data buffer out of the event handler context, especially if the data buffer to receive the data is not readily available. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device. The application must use the control transfer handle that was generated along with the event as the controlTransferHandle.

Preconditions

This function should only be called in response to a HID function driver event that requires a control transfer response.

Parameters

Parameters	Description
instance	USB Device HID Function Driver instance.
controlTransferHandle	Control Transfer handle for the control transfer.
data	Data buffer to receive the data stage of the control transfer.
size	size (in bytes) of the data to sent.

Returns

USB_DEVICE_HID_RESULT_ERROR_CONTROL_TRANSFER_FAILED - The request was not successful because the USB host cancelled the control transfer.

USB_DEVICE_HID_RESULT_OK - The request was successful.

Remarks

None.

Example

```
// -----
// In this example, the application responds to the SET REPORT command with
// in the event handler. The application uses the
// USB_DEVICE_HID_ControlReceive() function to do this.
// -----

USB_DEVICE_HID_EVENT_DATA_SET_REPORT * setReportEventData;
USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle;
uint8_t someReport[128];

USB_DEVICE_HID_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_HID_EVENT_SET_REPORT:

            // In this case, the application should receive the report
            // data from the host. The application must use the
            // USB_DEVICE_HID_ControlReceive() function to receive the data.

            setReportEventData = (USB_DEVICE_HID_EVENT_DATA_SET_REPORT *) pData;
            USB_DEVICE_HID_ControlReceive(instanceIndex, controlTransferHandle,
                                         someReport, setReportEventData->reportSize);

            break;
    }
}

// -----
// In this example, the application defers the response to the
// event. This is done by calling the USB_DEVICE_HID_ControlReceive()
// function out side the event handler.
// -----

USB_DEVICE_HID_EVENT_DATA_SET_REPORT setReportEventData;
USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE thisControlTransferHandle;
bool receiveReportFromHost;
uint8_t * someReport;

USB_DEVICE_HID_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_HID_EVENT_SET_LINE_CODING:

            // In this case, the application should send the line coding
            // data to the host. The application must responds to the
            // event outside the event handler because the data buffer is
            // not available.

            setReportEventData = (USB_DEVICE_HID_EVENT_DATA_GET_REPORT *)pData;
```



```

        thisControlTransferHandle = controlTransferHandle;
        receiveReportFromHost = true;

        break;
    }
}

// This is outside the event handler.
if(receiveReportFromHost)
{
    // The application allocates a buffer using an arbitrary function
    setReport = AllocateMemoryBuffer();

    USB_DEVICE_HID_ControlReceive(instanceIndex, thisControlTransferHandle
        setReport, setReportEventData->reportSize);
}

```

5.9.6.6.1.2 USB_DEVICE_HID_ControlSend Function

C

```

USB_DEVICE_HID_RESULT USB_DEVICE_HID_ControlSend(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * data,
    size_t size
);

```

Description

This function allows the application to respond to the HID function driver specific control transfer requests. It allows the application to [send](#) data to the host in the data stage of the control transfer. For example, the [USB_DEVICE_HID_EVENT_DATA_REPORT_SENT](#) event requires a control transfer response. The function can be called in the HID Function Driver event handler or can be called outside the event handler. Calling this function outside the event handler defers the response to the event. This allows the application to prepare the response data out of the event handler context, especially if the data is not readily available. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device. The application must use the control transfer handle that was generated along with the event as the controlTransferHandle.

Preconditions

This function should only be called in response to a HID function driver event that requires a control transfer response.

Parameters

Parameters	Description
instance	USB Device HID Function Driver instance.
controlTransferHandle	Control Transfer handle for the control transfer.
data	Data that represents the data stage of the control transfer.
size	size (in bytes) of the data to sent.

Returns

USB_DEVICE_HID_RESULT_ERROR_CONTROL_TRANSFER_FAILED - The request was not successful because the USB host cancelled the control transfer.

USB_DEVICE_HID_RESULT_OK - The request was successful.

Remarks

None.

Example

```

// -----
// In this example, the application responds to the GET REPORT
// with in the event handler. The application uses the
// USB_DEVICE_HID_ControlSend() function to do this.
// -----

USB_DEVICE_HID_EVENT_DATA_GET_REPORT * getReportEventData;
USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle;
uint8_t someReport[128];

USB_DEVICE_HID_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_HID_EVENT_GET_REPORT:

            // In this case, the application should send the report
            // to the host. The application must use the
            // USB_DEVICE_HID_ControlSend() function to send the data.

            getReportEventData = (USB_DEVICE_HID_EVENT_DATA_GET_REPORT *)pData;

            USB_DEVICE_HID_ControlSend(instanceIndex, controlTransferHandle
                                         someReport,
                                         getReportEventData->reportSize);

            break;
    }
}

// -----
// In this example, the application defers the response to the
// event. This is done by calling the USB_DEVICE_HID_ControlSend()
// function out side the event handler.
// -----

USB_DEVICE_HID_EVENT_DATA_GET_REPORT * getReportEventData;
USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle;
uint8_t someReport[128];
bool sendReportToHost;

USB_DEVICE_HID_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_HID_EVENT_GET_REPORT:

            // In this case, the application should send the report
            // data to the host. The application does not call this function
            // in the event handler, because it does not have the report
            // data ready.

```

```

        getReportEventData = (USB_DEVICE_HID_EVENT_DATA_GET_REPORT *)pData;
        thisControlTransferHandle = controlTransferHandle;
        sendReportToHost = true;

        break;
    }
}

// This is outside the event handler.

if(sendReportToHost)
{
    // The application fetches the report coding from the
    // an EEPROM.

    GetReportFromEEPROM(someReport);

    USB_DEVICE_HID_ControlSend(instanceIndex, controlTransferHandle
                               someReport,
                               getReportEventData->reportSize);
}

```

5.9.6.6.1.3 USB_DEVICE_HID_ControlStatus Function

C

```

USB_DEVICE_HID_RESULT USB_DEVICE_HID_ControlStatus(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    USB_DEVICE_HID_CONTROL_STATUS status
);

```

Description

This function allows the application to complete the status stage of the the HID Function Driver specific control transfer request. The application can either **accept** the data stage or stall it. Calling this function with status set to USB_DEVICE_HID_CONTROL_STATUS_OK will acknowledge the status stage of the control transfer. The control transfer can be stalled by setting the status parameter to USB_DEVICE_HID_CONTROL_STATUS_ERROR. The function can be called in the HID Function Driver event handler or can be called outside the event handler. Calling this function outside the event handler defers the response to the event. This allows the application to analyze the event response outside the event handler. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device. The application must use the control transfer handle that was generated along with the event as the controlTransferHandle. The application must be aware of events and associated control transfers that do or do not require data stages. Incorrect usage of the USB_DEVICE_HID_ControlStatus() function could cause the device function to be non-compliant.

Preconditions

This function should only be called in response to a HID function driver event that requires a control transfer response.

Parameters

Parameters	Description
instance	USB Device HID Function Driver instance.
controlTransferHandle	Control Transfer handle for the control transfer.
status	Set to USB_DEVICE_HID_CONTROL_STATUS_OK to acknowledge the control transfer. Set to USB_DEVICE_HID_CONTROL_STATUS_ERROR to stall the transfer.

Returns

USB_DEVICE_HID_RESULT_ERROR_CONTROL_TRANSFER_FAILED - The request was not successful because the USB

host cancelled the control transfer.

USB_DEVICE_HID_RESULT_OK - The request was successful.

Remarks

None.

Example

```
// -----
// In this code example, the application responds to the the
// SET_IDLE event in the event handler.
// -----

USB_DEVICE_HID_EVENT_DATA_SET_IDLE * setIdleEventData;

USB_DEVICE_HID_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_HID_EVENT_SET_IDLE:

            // The application can check if the break duration is supported.
            setIdleEventData = (USB_DEVICE_HID_EVENT_DATA_SET_IDLE *)pData;
            if(IsDurationSupported(setIdleEventData->duration))
            {
                USB_DEVICE_HID_ControlStatus(instanceIndex,
                    controlTransferHandle, USB_DEVICE_HID_CONTROL_STATUS_OK);
            }
            else
            {
                // If the duration is not supported the application
                // can stall it.

                USB_DEVICE_HID_ControlStatus(instanceIndex,
                    controlTransferHandle, USB_DEVICE_HID_CONTROL_STATUS_ERROR);
            }
            break;

            ...

    }

    return(USB_DEVICE_HID_EVENT_RESPONSE_NONE);
}

// -----
// In this example, the application defers the response to the
// SET_IDLE event. The application responds to the event in
// outside the event handler.
// -----

USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE thisControlTransferHandle;
USB_DEVICE_HID_EVENT_DATA_SET_IDLE * setIdleEventData;
uint8_t duration;
bool setIdleEvent;

USB_DEVICE_HID_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
```

```

    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_HID_EVENT_SEND_BREAK:

            // Get the idle duration.

            setIdleEventData = (USB_DEVICE_HID_EVENT_DATA_SET_IDLE *)pData;
            duration = setIdleEventData->duration;
            setIdleEvent = true;
            break;

            ...

    }

    return(USB_DEVICE_HID_EVENT_RESPONSE_NONE);
}

// This is outside the event handler.
if(setIdleEvent)
{
    // The application checks if the duration is supported
    if(IsDurationSupported(setIdleEventData->duration))
    {
        USB_DEVICE_HID_ControlStatus(instanceIndex,
            controlTransferHandle, USB_DEVICE_HID_CONTROL_STATUS_OK);
    }
    else
    {
        // If the duration is not supported the application
        // can stall it.

        USB_DEVICE_HID_ControlStatus(instanceIndex,
            controlTransferHandle, USB_DEVICE_HID_CONTROL_STATUS_ERROR);
    }
}
}

```

5.9.6.6.1.4 USB_DEVICE_HID_EventHandlerSet Function

C

```

USB_DEVICE_HID_RESULT USB_DEVICE_HID_EventHandlerSet(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT_HANDLER eventHandler,
    uintptr_t context
);

```

Description

This function registers a event handler for the specified HID function driver instance. This function should be called by the client when it receives a SET CONFIGURATION event from the device layer. A event handler must be registered for function driver to respond to function driver specific commands. If the event handler is not registered, the device layer will stall function driver specific commands and the USB device may not function.

Preconditions

This function should be called when the function driver has been initialized as a result of a set configuration.

Parameters

Parameters	Description
instance	Instance of the HID Function Driver.
eventHandler	A pointer to event handler function.
context	Application specific context that is returned in the event handler.

Returns

USB_DEVICE_HID_RESULT_OK - The operation was successful
 USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
 USB_DEVICE_HID_RESULT_ERROR_PARAMETER_INVALID - The eventHandler parameter is NULL

Remarks

None.

Example

*// This code snippet shows an example registering an event handler. Here
 // the application specifies the context parameter as a pointer to an
 // application object (appObject) that should be associated with this
 // instance of the HID function driver.*

```
USB_DEVICE_HID_RESULT result;

USB_DEVICE_HID_EVENT_RESPONSE APP_USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t context
)
{
    // Event Handling comes here

    switch(event)
    {
        ...
    }

    return(USB_DEVICE_HID_EVENT_RESPONSE_NONE);
}

result = USB_DEVICE_HID_EventHandlerSet ( 0 ,&APP_EventHandler, (uintptr_t) &appObject);
if(USB_DEVICE_HID_RESULT_OK != result)
{
    SYS_ASSERT ( false , "Error while registering event handler" );
}
```

5.9.6.6.1.5 USB_DEVICE_HID_ReportReceive Function**C**

```
USB_DEVICE_HID_RESULT USB_DEVICE_HID_ReportReceive(
    USB_DEVICE_HID_INDEX iHID,
    USB_DEVICE_HID_TRANSFER_HANDLE * handle,
    void * buffer,
    size_t size
);
```

Description

This function submits the buffer to HID function driver library to receive a report from host to device. On completion of the

transfer the library generates `USB_DEVICE_HID_EVENT_REPORT_RECEIVED` event to the application. A handle to the request is passed in the `transferHandle` parameter. The transfer handle expires when event handler for the `USB_DEVICE_HID_EVENT_REPORT_RECEIVED` exits. If the receive request could not be accepted, the function returns an error code and `transferHandle` will contain the value `USB_DEVICE_HID_TRANSFER_HANDLE_INVALID`.

Preconditions

USB device layer must be initialized.

Parameters

Parameters	Description
<code>iHID</code>	HID instance index.
<code>transferHandle</code>	HID transfer handle.
<code>buffer</code>	Pointer to buffer where the received report has to be received stored.
<code>size</code>	Buffer size.

Returns

`USB_DEVICE_HID_RESULT_OK` - The receive request was successful. `transferHandle` contains a valid transfer handle.

`USB_DEVICE_HID_RESULT_ERROR_TRANSFER_QUEUE_FULL` - internal request queue is full. The receive request could not be added.

`USB_DEVICE_HID_RESULT_ERROR_INSTANCE_NOT_CONFIGURED` - The specified instance is not configured yet.

`USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID` - The specified instance was not provisioned in the application and is invalid.

Remarks

None.

Example

```

USB_DEVICE_HID_TRANSFER_HANDLE hidTransferHandle;
USB_DEVICE_HID_RESULT result;

// Register APP_HIDEEventHandler function
USB_DEVICE_HID_EventHandlerSet( USB_DEVICE_HID_INDEX_0 ,
                                APP_HIDEEventHandler );

// Prepare report and request HID to send the report.
result = USB_DEVICE_HID_ReportReceive( USB_DEVICE_HID_INDEX_0,
                                        &hidTransferHandle ,
                                        &appReport[0], sizeof(appReport));

if( result != USB_DEVICE_HID_RESULT_OK)
{
    //Handle error.
}

//Implementation of APP_HIDEEventHandler
USB_DEVICE_HID_EVENT_RESPONSE APP_HIDEEventHandler
{
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t context
}
{
    USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED reportReceivedEventData;
```

```

    // Handle HID events here.
    switch (event)
    {
        case USB_DEVICE_HID_EVENT_REPORT_RECEIVED:
            if( (reportReceivedEventData->reportSize == sizeof(appReport)
                && reportReceivedEventData->report == &appReport[0])
                {
                // Previous transfer was complete.
            }
            break;
        ....
    }
}

```

5.9.6.6.1.6 USB_DEVICE_HID_ReportSend Function

C

```

USB_DEVICE_HID_RESULT USB_DEVICE_HID_ReportSend(
    USB_DEVICE_HID_INDEX iHID,
    USB_DEVICE_HID_TRANSFER_HANDLE * handle,
    void * buffer,
    size_t size
);

```

Description

This function places a request to [send](#) a HID report with the USB Device HID Function Driver Layer. The function places a requests with driver, the request will get serviced when report is requested by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_HID_EVENT_REPORT_SENT event. The amount of data sent, a pointer to the report and the transfer handle associated with the request is returned along with the event in the pData parameter of the event handler. The transfer handle expires when event handler for the USB_DEVICE_HID_EVENT_REPORT_SENT exits. If the [send](#) request could not be accepted, the function returns an error code and transferHandle will contain the value [USB_DEVICE_HID_TRANSFER_HANDLE_INVALID](#).

Preconditions

USB device layer must be initialized.

Parameters

Parameters	Description
instance	USB Device HID Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_HID_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the send request was successful.
data	pointer to the data buffer containing the report to be sent.
size	Size (in bytes) of the report to be sent.

Returns

USB_DEVICE_HID_RESULT_OK - The [send](#) request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_HID_RESULT_ERROR_TRANSFER_QUEUE_FULL - Internal request queue is full. The [send](#) request could not be added.

USB_DEVICE_HID_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Remarks

None.

Example

```

USB_DEVICE_HID_TRANSFER_HANDLE hidTransferHandle;
USB_DEVICE_HID_RESULT result;

// Register APP_HIDEventHandler function
USB_DEVICE_HID_EventHandlerSet( USB_DEVICE_HID_INDEX_0 ,
                                APP_HIDEventHandler );

// Prepare report and request HID to send the report.
result = USB_DEVICE_HID_ReportSend( USB_DEVICE_HID_INDEX_0,
                                    &hidTransferHandle ,
                                    &appReport[0], sizeof(appReport));

if( result != USB_DEVICE_HID_RESULT_OK)
{
    //Handle error.
}

//Implementation of APP_HIDEventHandler
USB_DEVICE_HID_EVENT_RESPONSE APP_HIDEventHandler
{
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t context
}
{
    USB_DEVICE_HID_EVENT_DATA_REPORT_SENT * reportSentEventData;

    // Handle HID events here.
    switch (event)
    {
        case USB_DEVICE_HID_EVENT_REPORT_SENT:

            reportSentEventData = (USB_DEVICE_HID_EVENT_REPORT_SENT *)pData;
            if(reportSentEventData->reportSize == sizeof(appReport))
            {
                // The report was sent completely.
            }
            break;

            ....

    }
    return(USB_DEVICE_HID_EVENT_RESPONSE_NONE);
}

```

5.9.6.6.2 Data Types and Constants**5.9.6.6.2.1 USB_DEVICE_HID_CONTROL_STATUS Enumeration****C**

```

typedef enum {
    USB_DEVICE_HID_CONTROL_STATUS_OK,
    USB_DEVICE_HID_CONTROL_STATUS_ERROR
} USB_DEVICE_HID_CONTROL_STATUS;

```

Description

USB Device HID Function Driver Control Transfer Status

This flag is used along with the [USB_DEVICE_HID_ControlStatus\(\)](#) function to indicate success or failure of a HID class specific control transfer request.

Members

Members	Description
USB_DEVICE_HID_CONTROL_STATUS_OK	The application must use this flag when the data received through the USB_DEVICE_HID_ControlReceive() function was accepted. Using this flag causes the status stage of the associated control transfer to be completed.
USB_DEVICE_HID_CONTROL_STATUS_ERROR	The application must use this flag when the control request was not supported or the data received using the USB_DEVICE_HID_ControlReceive() function was not accepted. Using this flag causes the status stage of the associated control transfer to be stalled.

Remarks

None.

5.9.6.6.2.2 USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE Type

C

```
typedef USB_DEVICE_CONTROL_TRANSFER_HANDLE USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE;
```

Description

USB Device HID Function Driver Control Transfer Handle

This is returned by the HID function driver event handler and should be used by the application while responding to HID function driver control transfer requests.

Remarks

None.

5.9.6.6.2.3 USB_DEVICE_HID_EVENT Enumeration

C

```
typedef enum {
    USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED,
    USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT,
    USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED,
    USB_DEVICE_HID_EVENT_GET_REPORT,
    USB_DEVICE_HID_EVENT_GET_IDLE,
    USB_DEVICE_HID_EVENT_GET_PROTOCOL,
    USB_DEVICE_HID_EVENT_SET_REPORT,
    USB_DEVICE_HID_EVENT_SET_IDLE,
    USB_DEVICE_HID_EVENT_SET_PROTOCOL,
    USB_DEVICE_HID_EVENT_SET_DESCRIPTOR,
    USB_DEVICE_HID_EVENT_REPORT_SENT,
    USB_DEVICE_HID_EVENT_REPORT_RECEIVED
} USB_DEVICE_HID_EVENT;
```

Description

USB Device HID Function Driver Events

These events are specific to the USB Device HID Function Driver instance. Each event description contains details about the parameters passed with event. The contents of pData depends on the generated event.

Events that are associated with the HID Function Driver Specific Control Transfers require application response and will be

generated along with a HID Control Transfer Handle. This allows the application to respond to the HID function driver control transfer requests. The application should respond to these events by using the [USB_DEVICE_HID_ControlReceive\(\)](#), [USB_DEVICE_HID_ControlSend\(\)](#) and [USB_DEVICE_HID_ControlStatus\(\)](#) functions.

Calling the [USB_DEVICE_HID_ControlStatus\(\)](#) function with a USB_DEVICE_HID_CONTROL_STATUS_ERROR will stall the control transfer request. The application would do this if the control transfer request is not supported. Calling the [USB_DEVICE_HID_ControlStatus\(\)](#) function with a USB_DEVICE_HID_CONTROL_STATUS_OK will complete the status stage of the control transfer request. The application would do this if the control transfer request is supported

The following code snippet shows an example of a possible event handling scheme.

```
// This code example shows all HID Function Driver events and a possible
// scheme for handling these events. In this example event responses are not
// deferred.
```

```
USB_DEVICE_HID_EVENT_RESPONSE USB_AppHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE controlTransferHandle,
    void * pData,
    uintptr_t userData
)
{
    USB_DEVICE_HID_EVENT_DATA_GET_REPORT          * getReportEventData;
    USB_DEVICE_HID_EVENT_DATA_GET_PROTOCOL        currentProtocol;
    USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL        * setProtocolEventData;
    USB_DEVICE_HID_EVENT_DATA_SET_IDLE            * setIdleEventData;
    USB_DEVICE_HID_EVENT_DATA_GET_IDLE            * getIdleEventData;
    USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR      * setDescriptorEventData;
    USB_DEVICE_HID_EVENT_DATA
    USB_DEVICE_HID_EVENT_DATA_SET_REPORT * setReportEventData;
    uint8_t idleRate;
    uint8_t someHIDReport[128];
    uint8_t someHIDDescriptor[128];

    switch(event)
    {
        case USB_DEVICE_HID_EVENT_GET_REPORT:

            // In this case, pData should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_GET_REPORT pointer. The application
            // must send the requested report by using the
            // USB_DEVICE_HID_ControlSend() function.

            getReportEventData = (USB_DEVICE_HID_EVENT_DATA_GET_REPORT *)pData;
            USB_DEVICE_HID_ControlSend(instanceIndex, controlTransferHandle,
                                       someHIDReport, getReportEventData->reportLength);

            break;

        case USB_DEVICE_HID_EVENT_GET_PROTOCOL:

            // In this case, pData will be NULL. The application
            // must send the current protocol to the host by using
            // the USB_DEVICE_HID_ControlSend() function.

            USB_DEVICE_HID_ControlSend(instanceIndex, controlTransferHandle,
                                       &currentProtocol,
                                       sizeof(USB_DEVICE_HID_EVENT_DATA_GET_PROTOCOL));

            break;

        case USB_DEVICE_HID_EVENT_GET_IDLE:
```

```
// In this case, pData will be a
// USB_DEVICE_HID_EVENT_DATA_GET_IDLE pointer type. The application
// must send the current idle rate to the host by using
// the USB_DEVICE_HID_ControlSend() function.

getLineCodingData = (USB_DEVICE_HID_EVENT_DATA_GET_IDLE *) pData;
USB_DEVICE_HID_ControlSend(instanceIndex, controlTransferHandle,
                           &currentIdleRate, 1);

break;

case USB_DEVICE_HID_EVENT_SET_REPORT:

    // In this case, pData should be interpreted as a
    // USB_DEVICE_HID_EVENT_DATA_SET_REPORT type pointer. The
    // application can analyze the request and then obtain the
    // report by using the USB_DEVICE_HID_ControlReceive() function.

    setReportEventData = (USB_DEVICE_HID_EVENT_DATA_SET_REPORT *)pData;
    USB_DEVICE_HID_ControlReceive(instanceIndex, controlTransferHandle,
                                   someHIDReport, setReportEventData->reportLength);

    break;

case USB_DEVICE_HID_EVENT_SET_PROTOCOL:

    // In this case, pData should be interpreted as a
    // USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL type pointer. The
    // application can analyze the data and decide to stall
    // or accept the setting. This shows an example of accepting
    // the protocol setting.

    setProtocolEventData = (USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL *)pData;
    USB_DEVICE_HID_ControlStatus(instanceIndex, controlTransferHandle,
                                   USB_DEVICE_HID_CONTROL_STATUS_OK);

    break;

case USB_DEVICE_HID_EVENT_SET_IDLE:

    // In this case, pData should be interpreted as a
    // USB_DEVICE_HID_EVENT_DATA_SET_IDLE type pointer. The
    // application can analyze the data and decide to stall
    // or accept the setting. This shows an example of accepting
    // the protocol setting.

    setIdleEventData = (USB_DEVICE_HID_EVENT_DATA_SET_IDLE *)pData;
    USB_DEVICE_HID_ControlStatus(instanceIndex, controlTransferHandle,
                                   USB_DEVICE_HID_CONTROL_STATUS_OK);

    break;

case USB_DEVICE_HID_EVENT_SET_DESCRIPTOR:

    // In this case, the pData should be interpreted as a
    // USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR type pointer. The
    // application can analyze the request and then obtain the
    // descriptor by using the USB_DEVICE_HID_ControlReceive() function.

    setDescriptorEventData = (USB_DEVICE_HID_EVENT_DATA_SET_REPORT *)pData;
    USB_DEVICE_HID_ControlReceive(instanceIndex, controlTransferHandle,
                                   someHIDReport, setReportEventData->reportLength);

    break;

case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
```

```

        // In this case, control transfer data was received. The
        // application can inspect that data and then stall the
        // handshake stage of the control transfer or accept it
        // (as shown here).

        USB_DEVICE_HID_ControlStatus(instanceIndex, controlTransferHandle,
            USB_DEVICE_HID_CONTROL_STATUS_OK);
        break;

    case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT:

        // This means that control transfer data was sent. The
        // application would typically acknowledge the handshake
        // stage of the control transfer.

        USB_DEVICE_HID_ControlStatus(instanceIndex, controlTransferHandle,
            USB_DEVICE_HID_CONTROL_STATUS_OK);
        break;

    case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED:

        // This is an indication only event. The application must
        // reset any HID control transfer related tasks when it receives
        // this event.

        break;

    case USB_DEVICE_HID_EVENT_REPORT_RECEIVED:

        // This means a HID report receive request has completed.
        // The pData member should be interpreted as a
        // USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED pointer type.

        break;

    case USB_DEVICE_HID_EVENT_REPORT_SENT:

        // This means a HID report send request has completed.
        // The pData member should be interpreted as a
        // USB_DEVICE_HID_EVENT_DATA_REPORT_SENT pointer type.

        break;
}

return(USB_DEVICE_HID_EVENT_RESPONSE_NONE);
}

```

Members

Members	Description
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	This event occurs when the data stage of a control write transfer has completed. This event would occur after the application uses the USB_DEVICE_HID_ControlReceive() function to respond to a HID Function Driver Control Transfer Event that requires data to be received from the host. The pData parameter will be NULL.

USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT	This event occurs when the data stage of a control read transfer has completed. This event would occur after the application uses the USB_DEVICE_HID_ControlSend() function to respond to a HID Function Driver Control Transfer Event that requires data to be sent to the host. The pData parameter will be NULL.
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED	This event occurs when an ongoing control transfer was aborted. The application must stop any remaining activities on control transfers.
USB_DEVICE_HID_EVENT_GET_REPORT	This event occurs when the host issues a GET REPORT command. The application must interpret the pData parameter as USB_DEVICE_HID_EVENT_DATA_GET_REPORT pointer type. If the report request is supported, the application must send the report to the host by using the USB_DEVICE_HID_ControlSend() function either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. If the report request is not supported, the application must stall the request by calling the USB_DEVICE_HID_ControlStatus() function with a USB_DEVICE_HID_CONTROL_STATUS_ERROR flag.
USB_DEVICE_HID_EVENT_GET_IDLE	This event occurs when the host issues a GET IDLE command. The pData parameter will be a USB_DEVICE_HID_EVENT_DATA_GET_IDLE pointer type. If the request is supported, the application must send the idle rate to the host by calling the USB_DEVICE_HID_ControlSend() function. This function can be called either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. If the request is not supported, the application must stall the request by calling the USB_DEVICE_HID_ControlStatus() function with a USB_DEVICE_HID_CONTROL_STATUS_ERROR flag.

USB_DEVICE_HID_EVENT_GET_PROTOCOL	This event occurs when the host issues a GET PROTOCOL command. The pData parameter will be NULL. If the request is supported, the application must send a USB_DEVICE_HID_EVENT_DATA_GET_PROTOCOL data type object, containing the current protocol, to the host by calling the USB_DEVICE_HID_ControlSend() function. This function can be called either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. If the request is not supported, the application must stall the request by calling the USB_DEVICE_HID_ControlStatus() function with a USB_DEVICE_HID_CONTROL_STATUS_ERROR flag.
USB_DEVICE_HID_EVENT_SET_REPORT	This event occurs when the host issues a SET REPORT command. The application must interpret the pData parameter as a USB_DEVICE_HID_EVENT_DATA_SET_REPORT pointer type. If the report request is supported, the application must provide a buffer, to receive the report, to the host by calling the USB_DEVICE_HID_ControlReceive() function either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. If the report request is not supported, the application must stall the request by calling the USB_DEVICE_HID_ControlStatus() function with a USB_DEVICE_HID_CONTROL_STATUS_ERROR flag.
USB_DEVICE_HID_EVENT_SET_IDLE	This event occurs when the host issues a SET IDLE command. The pData parameter will be USB_DEVICE_HID_EVENT_DATA_SET_IDLE pointer type. The application can analyze the idle duration and acknowledge or reject the setting by calling the USB_DEVICE_HID_ControlStatus() function. This function can be called in the event handler or after the event handler exits. If application can reject the request by calling the USB_DEVICE_HID_ControlStatus() function with a USB_DEVICE_HID_CONTROL_STATUS_ERROR flag. It can accept the request by calling this function with USB_DEVICE_HID_CONTROL_STATUS_OK flag.

USB_DEVICE_HID_EVENT_SET_PROTOCOL	This event occurs when the host issues a SET PROTOCOL command. The pData parameter will be NULL. If the request is supported, the application must provide a USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL data type object to receive the current protocol from the host by calling the USB_DEVICE_HID_ControlReceive() function. This function can be called either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. If the request is not supported, the application must stall the request by calling the USB_DEVICE_HID_ControlStatus() function with a USB_DEVICE_HID_CONTROL_STATUS_ERROR flag.
USB_DEVICE_HID_EVENT_SET_DESCRIPTOR	This event occurs when the host issues a SET DESCRIPTOR command. The pData parameter will be a USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR type pointer. If the request is supported, the application must provide a buffer to receive the descriptor from the host by calling the USB_DEVICE_HID_ControlReceive() function. This function can be called either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. If the request is not supported, the application must stall the request by calling the USB_DEVICE_HID_ControlStatus() function with a USB_DEVICE_HID_CONTROL_STATUS_ERROR flag.
USB_DEVICE_HID_EVENT_REPORT_SENT	This event indicates that USB_DEVICE_HID_ReportSend() function completed a report transfer on interrupt endpoint from host to device. The pData parameter will be a USB_DEVICE_HID_EVENT_DATA_REPORT_SENT type.
USB_DEVICE_HID_EVENT_REPORT_RECEIVED	This event indicates that USB_DEVICE_HID_ReportReceive() function completed a report transfer on interrupt endpoint from device to host. The pData parameter will be a USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED type

Remarks

Some of the events allow the application to defer responses. This allows the application some time to obtain the response data rather than having to respond to the event immediately. Note that a USB host will typically wait for event response for a finite

time duration before timing out and cancelling the event and associated transactions. Even when deferring response, the application must respond promptly if such timeouts have to be avoided.

5.9.6.6.2.4 USB_DEVICE_HID_EVENT_DATA_GET_IDLE Structure

C

```
typedef struct {  
    uint8_t reportID;  
} USB_DEVICE_HID_EVENT_DATA_GET_IDLE;
```

Description

USB Device HID Get Idle Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_SET_REPORT event.

Members

Members	Description
uint8_t reportID;	Report ID of the report of which idle rate is requested.

Remarks

None.

5.9.6.6.2.5 USB_DEVICE_HID_EVENT_DATA_GET_PROTOCOL Structure

C

```
typedef struct {  
    USB_HID_PROTOCOL_CODE protocol;  
} USB_DEVICE_HID_EVENT_DATA_GET_PROTOCOL;
```

Description

USB Device HID Get Protocol Event Data Type.

This defines the data type of the data to be sent to the host in response to USB_DEVICE_HID_EVENT_GET_PROTOCOL event. The data should be sent to the host by calling the [USB_DEVICE_HID_ControlSend\(\)](#) function.

Remarks

None.

5.9.6.6.2.6 USB_DEVICE_HID_EVENT_DATA_GET_REPORT Structure

C

```
typedef struct {  
    uint8_t reportType;  
    uint8_t reportID;  
    uint16_t reportLength;  
} USB_DEVICE_HID_EVENT_DATA_GET_REPORT;
```

Description

USB Device HID Get Report Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_GET_REPORT event.

Members

Members	Description
uint8_t reportType;	Report type
uint8_t reportID;	Report ID
uint16_t reportLength;	Report Length

Remarks

None.

5.9.6.6.2.7 USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED Structure**C**

```
typedef struct {  
    USB_DEVICE_HID_TRANSFER_HANDLE handle;  
    size_t length;  
} USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED;
```

Description

USB Device HID Report Received Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_REPORT_RECEIVED event.

Members

Members	Description
USB_DEVICE_HID_TRANSFER_HANDLE handle;	Transfer handle
size_t length;	Report size received

Remarks

None.

5.9.6.6.2.8 USB_DEVICE_HID_EVENT_DATA_REPORT_SENT Structure**C**

```
typedef struct {  
    USB_DEVICE_HID_TRANSFER_HANDLE handle;  
    size_t length;  
} USB_DEVICE_HID_EVENT_DATA_REPORT_SENT;
```

Description

USB Device HID Report Sent Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_REPORT_SENT event.

Members

Members	Description
USB_DEVICE_HID_TRANSFER_HANDLE handle;	Transfer handle
size_t length;	Report size transmitted

Remarks

None.

5.9.6.6.2.9 USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR Structure**C**

```
typedef struct {  
    uint8_t descriptorType;  
    uint8_t descriptorIndex;  
    uint16_t descriptorLength;  
} USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR;
```

Description

USB Device HID Set Descriptor Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_SET_DESCRIPTOR event.

Members

Members	Description
uint8_t descriptorType;	Descriptor Type
uint8_t descriptorIndex;	Descriptor Index
uint16_t descriptorLength;	Descriptor Length

Remarks

None.

5.9.6.6.2.10 USB_DEVICE_HID_EVENT_DATA_SET_IDLE Structure**C**

```
typedef struct {  
    uint8_t duration;  
    uint8_t reportID;  
} USB_DEVICE_HID_EVENT_DATA_SET_IDLE;
```

Description

USB Device HID Set Idle Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_SET_IDLE event.

Members

Members	Description
uint8_t duration;	Idle duration
uint8_t reportID;	Report ID

Remarks

None.

5.9.6.6.2.11 USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL Structure**C**

```
typedef struct {  
    USB_HID_PROTOCOL_CODE protocol;  
} USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL;
```

Description

USB Device HID Set Protocol Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_SET_PROTOCOL event.

Remarks

None.

5.9.6.6.2.12 USB_DEVICE_HID_EVENT_DATA_SET_REPORT Structure**C**

```
typedef struct {  
    uint8_t reportType;  
    uint8_t reportID;  
    uint16_t reportLength;  
} USB_DEVICE_HID_EVENT_DATA_SET_REPORT;
```

Description

USB Device HID Set Report Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_SET_REPORT event.

Members

Members	Description
uint8_t reportType;	Report type
uint8_t reportID;	Report ID
uint16_t reportLength;	Report Length

Remarks

None.

5.9.6.6.2.13 USB_DEVICE_HID_EVENT_HANDLER Type**C**

```
typedef USB_DEVICE_HID_EVENT_RESPONSE (* USB_DEVICE_HID_EVENT_HANDLER)(USB_DEVICE_HID_INDEX  
instanceIndex, USB_DEVICE_HID_EVENT event, USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE  
controlTransferHandle, void * pData, uintptr_t context);
```

Description

USB Device HID Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Device HID Function Driver event handling callback function. The application must register a pointer to a HID Function Driver events handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the HID Function Driver. The function driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

instanceIndex - Instance index of the HID Function Driver that generated the event.

event - Type of event generated.

controlTransferHandle - Control Transfer Handle for HID function driver events that require application response. The application should use this handle when calling the USB HID Device Control Transfer functions to respond to the events.

pData - This parameter should be type casted to a event specific pointer type based on the event that has occurred. Refer to the [USB_DEVICE_HID_EVENT](#) enumeration description for more details.

context - Value identifying the context of the application that registered the event handling function.

Remarks

None.

5.9.6.6.2.14 USB_DEVICE_HID_EVENT_RESPONSE Type**C**

```
typedef void USB_DEVICE_HID_EVENT_RESPONSE;
```

Description

USB Device HID Function Driver Event Handler Response Type

This is the return type of the HID Function Driver event handler.

Remarks

None.

5.9.6.6.2.15 USB_DEVICE_HID_INDEX Type**C**

```
typedef uintptr_t USB_DEVICE_HID_INDEX;
```

Description

USB Device HID Driver Index Numbers

This uniquely identifies a HID Function Driver instance.

Remarks

None.

5.9.6.6.2.16 USB_DEVICE_HID_INIT Structure**C**

```
typedef struct {  
    size_t hidReportDescriptorSize;  
    void * hidReportDescriptor;  
    size_t queueSizeReportSend;  
    size_t queueSizeReportReceive;  
} USB_DEVICE_HID_INIT;
```

Description

USB Device HID Function Driver Initialization Data Structure

This data structure must be defined for every instance of the HID function driver. It is passed to the HID function driver, by the Device Layer, at the time of initialization. The funcDriverInit member of the Device Layer Function Driver registration table entry must point to this data structure for an instance of the HID function driver.

Members

Members	Description
size_t hidReportDescriptorSize;	Size of the HID report descriptor
void * hidReportDescriptor;	Pointer to HID report descriptor

size_t queueSizeReportSend;	Report send queue size
size_t queueSizeReportReceive;	Report receive queue size

Remarks

None.

5.9.6.6.2.17 USB_DEVICE_HID_RESULT Enumeration**C**

```
typedef enum {  
    USB_DEVICE_HID_RESULT_OK,  
    USB_DEVICE_HID_RESULT_ERROR_TRANSFER_QUEUE_FULL,  
    USB_DEVICE_HID_RESULT_ERROR_INSTANCE_NOT_CONFIGURED,  
    USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID  
} USB_DEVICE_HID_RESULT;
```

Description

USB Device HID Function Driver USB Device HID Result enumeration.

This enumeration lists the possible USB Device HID Function Driver operation results. These values USB Device HID Library functions.

Members

Members	Description
USB_DEVICE_HID_RESULT_OK	The operation was successful

5.9.6.6.2.18 USB_DEVICE_HID_TRANSFER_HANDLE Type**C**

```
typedef uintptr_t USB_DEVICE_HID_TRANSFER_HANDLE;
```

Description

USB Device HID Function Driver Transfer Handle Definition

This definition defines a USB Device HID Function Driver Transfer Handle. A Transfer Handle is owned by the application but its value is modified by the [USB_DEVICE_HID_ReportSend\(\)](#) and [USB_DEVICE_HID_ReportReceive\(\)](#) functions. The transfer handle is valid for the life time of the transfer and expires when the transfer related event has occurred.

Remarks

None.

5.9.6.6.2.19 USB_DEVICE_HID_EVENT_RESPONSE_NONE Macro**C**

```
#define USB_DEVICE_HID_EVENT_RESPONSE_NONE
```

Description

USB Device HID Function Driver Event Handler Response None

This is the definition of the HID Function Driver Event Handler Response Type none.

Remarks

Intentionally defined to be empty.

5.9.6.6.2.20 USB_DEVICE_HID_INSTANCES_NUMBER Macro

C

```
#define USB_DEVICE_HID_INSTANCES_NUMBER
```

Description

USB Device HID Maximum Number of Instances

This macro defines the number of instances of the HID Function Driver. For example, if the application needs to implement two instances of the HID Function Driver (to create composite device) on one USB Device, the macro should be set to 2. Note that implementing a USB Device that features multiple HID interfaces requires appropriate USB configuration descriptors.

Remarks

None.

5.9.6.6.2.21 USB_DEVICE_HID_QUEUE_DEPTH_COMINED Macro

C

```
#define USB_DEVICE_HID_QUEUE_DEPTH_COMINED
```

Description

DOM-IGONORE-BEGIN

5.9.6.6.2.22 USB_DEVICE_HID_TRANSFER_HANDLE_INVALID Macro

C

```
#define USB_DEVICE_HID_TRANSFER_HANDLE_INVALID ((USB_DEVICE_HID_TRANSFER_HANDLE)(-1))
```

Description

USB Device HID Function Driver Invalid Transfer Handle Definition

This definition defines a USB Device HID Function Driver Invalid Transfer Handle. A Invalid Transfer Handle is returned by the [USB_DEVICE_HID_ReportReceive\(\)](#) and [USB_DEVICE_HID_ReportSend\(\)](#) functions when the request was not successful.

Remarks

None.

5.9.6.7 Files

Files

Name	Description
usb_device_hid.h	USB HID Function Driver
usb_device_hid_config_template..h	USB device HID class configuration definitions template,

Description

5.9.6.7.1 usb_device_hid.h

USB HID Function Driver

USB HID Function Driver

Enumerations

	Name	Description
	<u>USB_DEVICE_HID_CONTROL_STATUS</u>	USB Device HID Function Driver Control Transfer Status
	<u>USB_DEVICE_HID_EVENT</u>	USB Device HID Function Driver Events
	<u>USB_DEVICE_HID_RESULT</u>	USB Device HID Function Driver USB Device HID Result enumeration.

Functions

	Name	Description
	<u>USB_DEVICE_HID_ControlReceive</u>	This function allows the application to respond to the HID function driver specific control transfer requests. It allows the application to receive data from the host in the data stage of the control transfer.
	<u>USB_DEVICE_HID_ControlSend</u>	This function allows the application to respond to the HID function driver specific control transfer requests. It allows the application to send data to the host in the data stage of the control transfer.
	<u>USB_DEVICE_HID_ControlStatus</u>	This function allows the application to complete the status stage of the the HID Function Driver specific control transfer request.
	<u>USB_DEVICE_HID_EventHandlerSet</u>	This function registers a event handler for the specified HID function driver instance.
	<u>USB_DEVICE_HID_ReportReceive</u>	This function submits the buffer to HID function driver library to receive a report from host to device.
	<u>USB_DEVICE_HID_ReportSend</u>	This function submits the buffer to HID function driver library to send a report from device to host.

Macros

	Name	Description
	<u>USB_DEVICE_HID_EVENT_RESPONSE_NONE</u>	USB Device HID Function Driver Event Handler Response Type None.
	<u>USB_DEVICE_HID_TRANSFER_HANDLE_INVALID</u>	USB Device HID Function Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	<u>USB_DEVICE_HID_EVENT_DATA_GET_IDLE</u>	USB Device HID Get Idle Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_GET_PROTOCOL</u>	USB Device HID Get Protocol Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_GET_REPORT</u>	USB Device HID Get Report Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED</u>	USB Device HID Report Received Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_REPORT_SENT</u>	USB Device HID Report Sent Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR</u>	USB Device HID Set Descriptor Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_SET_IDLE</u>	USB Device HID Set Idle Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL</u>	USB Device HID Set Protocol Event Data Type.
	<u>USB_DEVICE_HID_EVENT_DATA_SET_REPORT</u>	USB Device HID Set Report Event Data Type.
	<u>USB_DEVICE_HID_INIT</u>	USB Device HID Function Driver Initialization Data Structure

Types

	Name	Description
	<u>USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE</u>	USB Device HID Function Driver Control Transfer Handle
	<u>USB_DEVICE_HID_EVENT_HANDLER</u>	USB Device HID Event Handler Function Pointer Type.

	USB_DEVICE_HID_EVENT_RESPONSE	USB Device HID Function Driver Event Callback Response Type
	USB_DEVICE_HID_INDEX	USB device HID Function Driver Index.
	USB_DEVICE_HID_TRANSFER_HANDLE	USB Device HID Function Driver Transfer Handle Definition.

Company

Microchip Technology Inc.

FileName: usb_hid_function_driver.h

5.9.6.7.2 usb_device_hid_config_template..h

USB Device HID Class Configuration Definitions for the Template

These definitions statically define the device HID Class mode of operation.

Macros

	Name	Description
	USB_DEVICE_HID_INSTANCES_NUMBER	Specifies the number of HID instances.
	USB_DEVICE_HID_QUEUE_DEPTH_COMINED	DOM-IGONORE-BEGIN

File Name

usb_device_hid_config_template.h

Company

Microchip Technology Inc.

5.9.7 USB MSD Device Library**5.9.7.1 Introduction****USB Mass Storage Device (MSD) Class Library for Microchip Microcontrollers**

This library provides a high-level abstraction of USB Mass Storage Device (MSD) function driver with a convenient C language interface. This library supports USB Mass Storage Class Bulk-Only Transport, Revision 1.0 from USB Implementers Forum.

Description

The mass storage device class function driver library enables the developers to add mass storage functionality to a device. This library along with USB device stack and a required media driver can be used to build devices like USB thumb drive and card readers.

5.9.7.2 Release Notes

MPLAB Harmony Version: v0.70b **USB MSD Device Library Version :** 0.01 Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.9.7.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

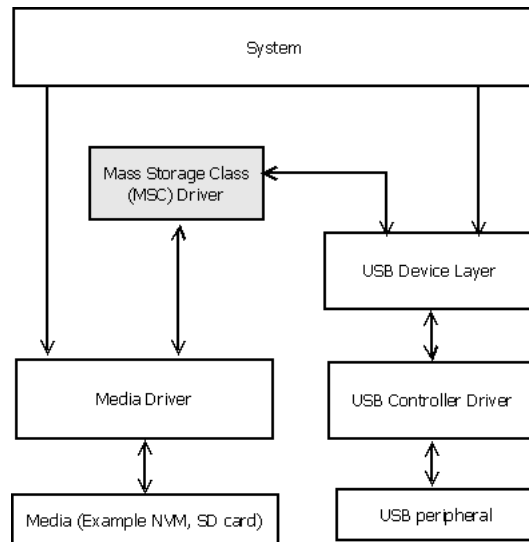
To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.9.7.4 Using the Library

5.9.7.4.1 Abstraction Model

The basic software architecture into which USB mass storage device class function driver fits into is depicted in the following diagram.



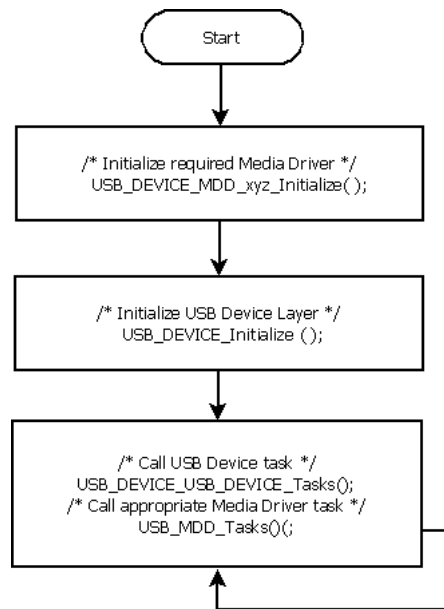
The system makes no direct calls to MSD function driver. It is the USB device layer that initializes and deinitializes the MSD function driver. The MSD function driver is initialized when host configures the device. It is deinitialized when device exits the configured state. The device exits the configured state when device is detached from the host or if host issues a bus reset to the device or if host changes the device configuration to a new value. The MSD function driver task is called from within the USB device layer task. This means that MSD function driver task runs at the same priority level as USB device layer task. So, the user has to register the MSD function driver with USB device layer to allow the USB device layer to make calls to MSD function driver callback functions at runtime. This registration can be done using compile time configuration.

The system does the media driver initialization and is responsible for calling the media driver tasks. The MSD function driver interacts with media driver at runtime to perform sector read, write and erase by calling the media driver callback functions. This requires for the user to register the media driver with the MSD function driver using compile time configuration.

The subsequent sections of this help document describes how to register the MSD function driver and media driver with USB device layer and MSD function driver respectively.

5.9.7.4.2 How the Library Works

5.9.7.4.2.1 System Interaction



The above figure shows the system interaction required for MSD function driver. The system has to initialize the media driver and the USB device layer. There are no direct calls the system has to make to initialize the MSD function driver. The USB device layer initializes and deinitializes the MSD function driver appropriately at runtime. The system is also responsible for calling the media driver tasks if applicable.

5.9.7.4.2.2 Object Initialization

Each instance of MSD function driver must be initialized by an object of data type `USB_DEVICE_MSD_INIT` at compile time. If there are more than one instance of MSD function driver, then user must create multiple objects of this data type. The Init Object is then linked to an instance of MSD function driver by adding its pointer to the function driver registration table (Refer to section "Registering MSD function driver").

The MSD Init Object contains following data about an instance of MSD.

1. Interface number.
2. Bulk-IN endpoint address
3. Bulk-OUT endpoint address
4. Size of the endpoint
5. Number of logical units (LUN) supported by the particular instance of MSD.
6. One or more media driver instance number depending on the number of logical units supported. This is the media driver with which the particular logical unit will interact.
7. One or more inquiry response structure depending on the number of logical units supported.
8. Pointers to one or more sets of media driver callback functions depending on the number of logical units supported.

The device layer passes the Init Object to MSD function driver when it initializes the function driver.

The following code example shows how to initialize the MSD using Init Object.

```

const USB_DEVICE_MSD_INIT msdInit =
{
    0, /* Interface number */

```

```

1,  /* Bulk IN endpoint */
1,  /* Bulk OUT endpoint */
64, /* Size of the bulk endpoint */

1, /* Number of logical unit (LUN) */

{
    /* Begin first LUN info */
    {
        /* Media driver instance number */
        USB_DEVICE_MDD_INTFLASH_INDEX,

        /* Inquiry response for the first logical unit */
        {
            0x00, /* peripheral device is connected, direct access block device */
            0x80, /* removable media */
            0x04, /* version = 00=> does not conform to any standard, 4=> SPC-2 */
            0x02, /* response is in format specified by SPC-2 */
            0x20, /* n-4 = 36-4=32= 0x20 */
            0x00, /* sccs etc. */
            0x00, /* bque=1 and cmdque=0, indicates simple queuing 00 is obsolete
                    but as in case of other device, we are just using 00. */
            0x00, /* 00 obsolete, 0x80 for basic task queuing */
            {
                'M','i','c','r','o','c','h','p'
            },
            /* this is the T10 assigned Vendor ID */
            {
                'M','a','s','s',' ','S','t','o','r','a','g','e',' ',' ',' ',' ',' '
            },
            {
                '0','0','0','1'
            }
        },

        /* MDD function callback pointers for first loical uint */
        {
            &USB_DEVICE_MDD_INTFLASH_Status,
            &USB_DEVICE_MDD_INTFLASH_Open,
            &USB_DEVICE_MDD_INTFLASH_ReadCapacity,
            &USB_DEVICE_MDD_INTFLASH_ReadSectorSize ,
            &USB_DEVICE_MDD_INTFLASH_MediaDetect,
            &USB_DEVICE_MDD_INTFLASH_SectorRead,
            &USB_DEVICE_MDD_INTFLASH_WriteProtectState,
            &USB_DEVICE_MDD_INTFLASH_SectorWrite
        }
    }
} /* End of first LUN info */

{ /* Start of second LUN info */

} /* End of second LUN info */

};

```

5.9.7.4.2.3 MSD Function Driver Registration

The MSD function driver is registered with the USB device layer using the function registration table structure. The following code example shows how the MSD function driver can be registered with the USB device layer using function registration table. The structure variable *msdFunctionDriver* contains pointers to MSD function driver callbacks.

Example:

```

/* The device layer invokes the MSD function driver if the speed is full speed
and the host selects configuration value as 1 */
funcRegistrationTable[1] =
{
    {
        /* Speed
        USB_SPEED_FULL,

        /* Configuration value */
        1,

        /* Interface number */
        0,

        /* Number of interfaces */
        1,

        /* MSD instance index */
        USBMSD_FUNC_INDEX,

        /* MSD driver init object */
        (void*)&msdInit,

        /* MSD driver */
        &msdFunctionDriver
    }
};

```

5.9.7.4.2.4 Media Driver Registration

The MSD function driver makes call to media driver functions at runtime to perform sector erase, read and write operations. Therefore, the user must register the pointers to media driver callback functions with the MSD function driver . This has to be done at compile time while initializing the Init Object. Refer to the code example shown in the "Init Object" section for more information.

5.9.7.5 Library Interface

This section describes the configuration options provided by the MSD function driver to the system and the available data types. Refer to each section for a detailed description.

5.9.7.5.1 System Configuration Functions

Macros

	Name	Description
	<u>USB_DEVICE_MSD_MAX_INSTANCES</u>	This constant sets maximum possible number of instances of USB device MSD function driver that can be instantiated by using USB_DEVICE_MSD_Initialize() routine.
	<u>USB_DEVICE_MSD_MAX_LUN</u>	This constant sets maximum possible number of Logical Unit an instance of MSD can support.
	<u>USB_DEVICE_MSD_MAX_SECTOR_SIZE</u>	This constant defines the max possible media sector size supported by the MSD.

Description

5.9.7.5.1.1 USB_DEVICE_MSD_MAX_INSTANCES Macro

C

```
#define USB_DEVICE_MSD_MAX_INSTANCES 1
```

Remarks

The static implementation supports only one instance. Setting the value of this constant to > 1 has no effect on static implementations. Only in dynamic implementation of USB device MSD function driver this value can be set > 1.

USB device MSD function driver has to support atleast one instance. Hence, make sure the value of this constant is set to > 0.

Increasing the instance count consumes RAM and can lead to performance degradation.

5.9.7.5.1.2 USB_DEVICE_MSD_MAX_LUN Macro

C

```
#define USB_DEVICE_MSD_MAX_LUN 1
```

Description

This constant sets maximum possible number of Logical Unit (LUN) an instance of MSD can support.

Remarks

The value of this macro must not be set to zero. Each instance of USB MSDfunction driver can support atleast one LUN. Hence set the value to atleast 1.

5.9.7.5.1.3 USB_DEVICE_MSD_MAX_SECTOR_SIZE Macro

C

```
#define USB_DEVICE_MSD_MAX_SECTOR_SIZE 512
```

Description

If there are two logical units (LUNs), one that supports sector size 512 bytes and then the other one that supports 1024 bytes, then set the value of this macro to 1024.

Remarks

none.

5.9.7.5.2 Data Types and Constants

Macros

	Name	Description
	USB_DEVICE_MSD_INTF	MSD Interface Class Code
	USB_DEVICE_MSD_INTF_SUBCLASS	Only scsi transparent is supported.
	USB_DEVICE_MSD_PROTOCOL	MSD Interface Class Protocol Codes

Structures

	Name	Description
	USB_DEVICE_MSD_MEDIA_FUNCTIONS	This structure contains all the media driver callback function pointers. The MSD driver makes call to these function pointers at run time to know the status of the media and to read and write the media.
	USB_DEVICE_MSD_INIT	This structure contains required parameters for MSD initialization.

	USB_DEVICE_MSD_INQUIRY_RESPONSE	Inquiry response structure.
	USB_DEVICE_MSD_MEDIA_INIT_DATA	This structure holds media related data of a particular logical unit.

Description**5.9.7.5.2.1 USB_DEVICE_MSD_INTF Macro****C**

```
#define USB_DEVICE_MSD_INTF 0x08
```

Description

MSD Interface Class Code

5.9.7.5.2.2 USB_DEVICE_MSD_INTF_SUBCLASS Macro**C**

```
#define USB_DEVICE_MSD_INTF_SUBCLASS 0x06 // Only scsi transparent is supported.
```

Description

Only scsi transparent is supported.

5.9.7.5.2.3 USB_DEVICE_MSD_PROTOCOL Macro**C**

```
#define USB_DEVICE_MSD_PROTOCOL 0x50
```

Description

MSD Interface Class Protocol Codes

5.9.7.5.2.4 USB_DEVICE_MSD_MEDIA_FUNCTIONS Structure**C**

```
struct USB_DEVICE_MSD_MEDIA_FUNCTIONS {
    SYS_STATUS (* mediaInitState)(SYS_MODULE_OBJ objIndex);
    DRV_HANDLE (* open)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
    void (* close)(DRV_HANDLE hClient);
    uint32_t (* readCapacity)(DRV_HANDLE drvHandle);
    uint32_t (* readSectorSize)(DRV_HANDLE drvHandle);
    bool (* mediaDetect)(DRV_HANDLE drvHandle);
    bool (* sectorRead)(DRV_HANDLE drvHandle, uint32_t sector_addr, uint8_t * buffer, void *
refHandle, MEDIA_OP_CMPLT_CB callBack);
    uint8_t (* writeProtectState)(DRV_HANDLE drvHandle);
    bool (* sectorWrite)(DRV_HANDLE drvHandle, uint32_t sector_addr, uint8_t* buffer, void *
refHandle, MEDIA_OP_CMPLT_CB callBack);
};
```

Description

The user must provide the callback function address of the media at compile time. MSD driver calls these function during run time to perform media operation or to know the media status.

Members

Members	Description
<code>SYS_STATUS (* mediaInitState)(SYS_MODULE_OBJ objIndex);</code>	Media init state.
<code>DRV_HANDLE (* open)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);</code>	Function pointer to the MediaInitialize() function of the physical media being used.
<code>void (* close)(DRV_HANDLE hClient);</code>	Function pointer for closing the physical media.
<code>uint32_t (* readCapacity)(DRV_HANDLE drvHandle);</code>	Function pointer to the ReadCapacity() function of the physical media being used.
<code>uint32_t (* readSectorSize)(DRV_HANDLE drvHandle);</code>	Function pointer to the ReadSectorSize() function of the physical media being used.
<code>bool (* mediaDetect)(DRV_HANDLE drvHandle);</code>	Function pointer to the MediaDetect() function of the physical media being used.
<code>bool (* sectorRead)(DRV_HANDLE drvHandle, uint32_t sector_addr, uint8_t * buffer, void * refHandle, MEDIA_OP_CMPLT_CB callBack);</code>	Function pointer to the SectorRead() function of the physical media being used.
<code>uint8_t (* writeProtectState)(DRV_HANDLE drvHandle);</code>	Function pointer to the WriteProtectState() function of the physical media being used.
<code>bool (* sectorWrite)(DRV_HANDLE drvHandle, uint32_t sector_addr, uint8_t * buffer, void * refHandle, MEDIA_OP_CMPLT_CB callBack);</code>	Function pointer to the SectorWrite() function of the physical media being used.

Remarks

None.

5.9.7.5.2.5 USB_DEVICE_MSD_INIT Structure**C**

```
struct USB_DEVICE_MSD_INIT {
    uint8_t numberOfLogicalUnits;
    USB_DEVICE_MSD_MEDIA_INIT_DATA mediaInit[USB_DEVICE_MSD_MAX_LUN];
};
```

Description

USB MSD init structure.

This structure contains interface number, bulk-IN and bulk-OUT endpoint addresses, endpointSize, number of logical units supported and pointer to array of structure that contains media initialization.

Members

Members	Description
<code>uint8_t numberOfLogicalUnits;</code>	Number of logical units supported.
<code>USB_DEVICE_MSD_MEDIA_INIT_DATA mediaInit[USB_DEVICE_MSD_MAX_LUN];</code>	Media related data

Remarks

This structure must be configured by the user at compile time.

5.9.7.5.2.6 USB_DEVICE_MSD_INQUIRY_RESPONSE Structure

C

```

struct USB_DEVICE_MSD_INQUIRY_RESPONSE {
    uint8_t peripheral;
    uint8_t removable;
    uint8_t version;
    uint8_t responseDataFormat;
    uint8_t additionalLength;
    uint8_t sccstp;
    uint8_t bqueetc;
    uint8_t cmdQue;
    uint8_t vendorID[8];
    uint8_t productID[16];
    uint8_t productRev[4];
};

```

Description

USB Device MSD inquiry response structure as defined in SCSI Primary Commands - 4.

Members

Members	Description
uint8_t peripheral;	Peripheral_Qualifier:3; Peripheral_DevType:5;
uint8_t removable;	Removable medium bit7 = 0 means non removable, rest reserved
uint8_t version;	Version
uint8_t responseDataFormat;	b7,b6 Obsolete, b5 Access control co-ordinator, b4 hierarchical addressing support b3:0 response data format 2 indicates response is in format defined by spec
uint8_t additionalLength;	length in bytes of remaining in standard inquiry data
uint8_t sccstp;	b7 SCCS, b6 ACC, b5-b4 TGPS, b3 3PC, b2-b1 Reserved, b0 Protected
uint8_t bqueetc;	b7 bque, b6- EncServ, b5-VS, b4-MultiP, b3-MChngr, b2-b1 Obsolete, b0-Addr16
uint8_t cmdQue;	b7-b6 Obsolete, b5-WBUS, b4-Sync, b3-Linked, b2 Obsolete, b1 Cmdque, b0-VS
uint8_t vendorID[8];	Vendor ID
uint8_t productID[16];	Product ID
uint8_t productRev[4];	Product Revision

Remarks

None.

5.9.7.5.2.7 USB_DEVICE_MSD_MEDIA_INIT_DATA Structure

C

```

struct USB_DEVICE_MSD_MEDIA_INIT_DATA {
    SYS_MODULE_INDEX instanceIndex;
    USB_DEVICE_MSD_INQUIRY_RESPONSE inquiryResponse;
    USB_DEVICE_MSD_MEDIA_FUNCTIONS mediaCallBackPtrs;
};

```

Description

It holds pointer to inquiry response, instance index and pointer to a structure that contains all media callback functions.

Members

Members	Description
SYS_MODULE_INDEX instanceIndex;	Instance index of the media

USB_DEVICE_MSD_INQUIRY_RESPONSE inquiryResponse;	Pointer to inquiry response
USB_DEVICE_MSD_MEDIA_FUNCTIONS mediaCallBackPtrs;	Callback function pointers of the media

Remarks

An object of this structure must be configured by the user at compile time.

5.9.7.6 Files

Files

Name	Description
usb_device_msd.h	USB device MSD function driver interface header
usb_device_msd_config_template.h	usb device MSD configuration template header file

Description

5.9.7.6.1 usb_device_msd.h

USB MSD function driver interface header

USB device MSD function driver interface header

Macros

	Name	Description
	USB_DEVICE_MSD_INTF	MSD Interface Class Code
	USB_DEVICE_MSD_INTF_SUBCLASS	Only scsi transparent is supported.
	USB_DEVICE_MSD_PROTOCOL	MSD Interface Class Protocol Codes

Structures

	Name	Description
	USB_DEVICE_MSD_INIT	This structure contains required parameters for MSD initialization.
	USB_DEVICE_MSD_INQUIRY_RESPONSE	Inquiry response structure.
	USB_DEVICE_MSD_MEDIA_FUNCTIONS	This structure contains all the media driver callback function pointers. The MSD driver makes call to these function pointers at run time to know the status of the media and to read and write the media.
	USB_DEVICE_MSD_MEDIA_INIT_DATA	This structure holds media related data of a particular logical unit.

File Name

usb_device_msd.h

Company

Microchip Technology Inc.

5.9.7.6.2 usb_device_msd_config_template.h

USB device MSD function driver compile time options

This file contains USB device MSD function driver compile time options(macros) that has to be configured by the user. This file is

a template file and must be used as an example only. This file must not be directly included in the project.

Macros

	Name	Description
	<u>USB_DEVICE_MSD_MAX_INSTANCES</u>	This constant sets maximum possible number of instances of USB device MSD function driver that can be instantiated by using USB_DEVICE_MSD_Initialize() routine.
	<u>USB_DEVICE_MSD_MAX_LUN</u>	This constant sets maximum possible number of Logical Unit an instance of MSD can support.
	<u>USB_DEVICE_MSD_MAX_SECTOR_SIZE</u>	This constant defines the max possible media sector size supported by the MSD.

File Name

usb_device_msd_config_template.h

Company

Microchip Technology Incorporated

5.9.8 USB Host Layer Library

5.9.8.1 Introduction

Introduces the MPLAB Harmony USB Host Library

Description

The USB Host layer library is part of USB Host stack that is available for the Microchip family of microcontrollers. This library uses the USB Host Controller Driver to interact with the USB peripheral and therefore cannot operate independently of it. Within the USB Host stack, the USB Host layer is responsible for enumeration the attached device, matching available class drivers and maintaining the class drivers.

The USB Host layer library implementation adheres to USB Device Framework of chapter-9 of USB specification 2.0. In the USB Host stack, the host layer features the following:

- Supports USB Low-Speed, Full-Speed, and Hi-Speed USB Devices
- Based on a modular and event driven architecture.
- Supports the PIC32MX and PIC32MZ families of microcontrollers
- Designed to support multiple device and multiple configurations
- Contains class driver to supports the following type of devices
 - CDC
 - MSD
- Supports non-blocking operation and is RTOS friendly
- Designed to integrate readily with other Harmony Middleware
- Supports both interrupt and polling operation

5.9.8.2 Release Notes

MPLAB Harmony Version: v0.70b **USB Host Layer Library Version :** 0.01 Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

- The host Stack has not been tested with a Real Time Operating System.
- Attach/Detach behavior has been tested in a limited capacity.
- The host stack only supports interrupt mode operation. Polled mode operation will be added in a future release.
- While running the host stack on PIC32MZ microcontroller, the stack requires 3 seconds to initialize. This is due to a hardware issue with PIC32MZ USB module.
- MSD Host has been tested with a limited number of commercially available USB Flash Drives.
- MSD Host and the USB Host Layer have not been tested for read/write throughput. This will be done in a future release.
- Host stack has not been tested with low speed devices.

5.9.8.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.9.8.4 Library Interface

Data Types and Constants

	Name	Description
	<u>USB_HOST_HANDLE</u>	Handle identifying a client when calling Host layer client functions.
	<u>USB_HOST_INIT</u>	Defines the data required to initialize a USB Host Layer instance.
	<u>USB_HOST_HANDLE_INVALID</u>	Definition of an invalid USB Host Client Handle

	USB_DEVICE_INFORMATION	Defines the USB Device Information data structure.
	USB_HOST_EVENT_CALLBACK	USB Host Event Handler Function Pointer Type.
	USB_HOST_EVENT_DATA_VBUS_REQUEST_POWER	Defines the data returned along with the USB_HOST_EVENT_VBUS_REQUEST_POWER event.
	USB_HOST_EVENT_RESPONSE	USB Host Event Handler Response Type
	USB_HOST_EVENTS	Identifies the possible events that the host layer can generate.
	USB_HOST_TPL_FLAGS	USB Host Layer Target Peripheral List Entry flags
	TPL_MATCH_VID_PID	This macro when used in the TPL table will direct the Host Layer to match the corresponding class driver by device vendor ID (VID) and product ID (PID).
	USB_HOST_0	USB Host module index definitions
	USB_HOST_1	This is macro USB_HOST_1.
	USB_HOST_ENDPOINT_TABLE_SIZE	USB Host Endpoint Table size needed while running the host on PIC32MX devices.
	USB_HOST_EVENT_RESPONSE_NONE	USB Host Layer Event Handler Response Type None.

Functions

	Name	Description
	USB_HOST_Close	Closes an opened-instance of the USB Host Layer.
	USB_HOST_Open	Opens the specified Host Layer instance and returns a handle to it
	USB_HOST_Deinitialize	Deinitializes the specified instance of the USB Host Layer.
	USB_HOST_DeviceInformationGet	Returns select information about the attached device.
	USB_HOST_DeviceIsResumed	Checks if the attached USB device has been resumed.
	USB_HOST_DeviceIsSuspended	Checks if the attached USB device has been suspended.
	USB_HOST_DeviceResume	Resumes the USB.
	USB_HOST_DeviceSuspend	Suspends the USB.
	USB_HOST_EventCallBackSet	Allows a client to identify an event handling function for the host layer to call back when host layer events are generated.
	USB_HOST_Initialize	Initializes the USB Host layer instance for the specified instance index.
	USB_HOST_OperationDisable	Disabled host operation.
	USB_HOST_OperationEnable	Enables host operation.
	USB_HOST_OperationIsEnabled	Allows the application to check if the host operation is enabled.
	USB_HOST_Status	Gets the current status of the USB Host Layer.
	USB_HOST_Tasks	Maintains the USB Host's state machine. It manages the USB Module driver and responds to USB Module driver events.

Description

This section describes the Application Programming Interface (API) functions of the USB Host Layer

5.9.8.4.1 Functions

5.9.8.4.1.1 USB_HOST_Close Function

C

```
void USB_HOST_Close(
    USB_HOST_HANDLE hostHandle
);
```

Description

This routine closes an opened-instance of the USB Host Layer, invalidating the handle.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the Host Layers's open routine

Returns

None

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling [USB_HOST_Open](#) before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually there is no need for the driver client to verify that the Close operation has completed.

Example

```
USB_HOST_HANDLE handle; // Returned from USB_HOST_Open

USB_HOST_Close(handle);
```

5.9.8.4.1.2 USB_HOST_Open Function

C

```
USB_HOST_HANDLE USB_HOST_Open(
    SYS_MODULE_INDEX index,
    const DRV_IO_INTENT ioIntent
);
```

Description

This routine opens the specified Host Layer instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the host layer. The host layer can be opened exclusively by a client by specifying the ioIntent as DRV_IO_INTENT_EXCLUSIVE. Any other flag settings will not have an effect on clients interaction with the host layer.

Preconditions

Function [USB_HOST_Initialize](#) must have been called before calling this function.

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	can be DRV_IO_INTENT_EXCLUSIVE which allows a client to open the host layer exclusively. Any other flags are ignored.

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is [USB_HOST_HANDLE_INVALID](#). Error can occur

- if the number of client objects allocated via USB_HOST_CLIENTS_NUMBER is insufficient.

- if the client is trying to open the host layer but host layer has been opened exclusively by another client.
- if the host layer instance being opened is not initialized or is invalid.

Remarks

The handle returned is valid until the [USB_HOST_Close](#) routine is called. This routine will NEVER block waiting for hardware. This function is thread safe in a RTOS application. It should not be called in an ISR.

Example

```
USB_HOST_HANDLE handle;

handle = USB_HOST_Open(USB_HOST_0, DRV_IO_INTENT_EXCLUSIVE);
if (USB_HOST_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

5.9.8.4.1.3 USB_HOST_Deinitialize Function

C

```
void USB_HOST_Deinitialize(
    SYS_MODULE_OBJ driverObject
);
```

Description

Deinitializes the specified instance of the USB Host Layer. This function will also deinitialize the USB Module driver and hence stop all USB Host related operation on the bus. All internal data structures will be reset.

Preconditions

Function [USB_HOST_Initialize](#) should have been called before calling this function.

Parameters

Parameters	Description
object	USB Host layer object handle, returned from the USB_HOST_Initialize routine

Returns

None.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Example

```
SYS_MODULE_OBJ    object;    // Returned from USB_HOST_Initialize
SYS_STATUS        status;

USB_HOST_Deinitialize(object);

status = USB_HOST_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

```
}
```

5.9.8.4.1.4 USB_HOST_DeviceInformationGet Function

C

```
USB_ERROR USB_HOST_DeviceInformationGet(  
    USB_HOST_HANDLE hostHandle,  
    uint8_t deviceAddress,  
    USB_DEVICE_INFORMATION * deviceInfo  
);
```

Description

This function returns select information about the attached device. The application may use this information to update its display or for other application purposes. The device information is returned in the deviceInformation data structure.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the Host Layers's open routine
address	Address of the device.
deviceInformation	pointer to a data structure where the device information will be stored

Returns

USB_ERROR_NONE - Function was successful. USB_ERROR_PARAMETER_INVALID - Device address or host handle is invalid or deviceInformation pointer is NULL.

Remarks

None.

Example

```
USB_HOST_HANDLE handle; // Returned from USB_HOST_Open  
USB_ERROR result;  
USB_DEVICE_INFORMATION deviceInfo;  
  
// Get informationa about attached device with address 2  
result = USB_HOST_DeviceInformationGet(handle, 2, &deviceInfo);  
if(USB_ERROR_NONE == result)  
{  
    // Display the string descriptor of the attached device.  
    APP_PrintToDisplay(deviceInformation.stringDescriptor);  
}
```

5.9.8.4.1.5 USB_HOST_DeviceIsResumed Function

C

```
bool USB_HOST_DeviceIsResumed(  
    USB_HOST_HANDLE hostHandle,  
    uint8_t deviceAddress  
);
```

Description

This function checks if the attached USB device has been resumed. This function can be called after calling the [USB_HOST_DeviceResume\(\)](#) function to check if resume request has been completed.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the Host Layers's open routine
address	Address of the device.

Returns

true - Device is resumed. false - Device is not resumed or device address or handle is invalid.

Remarks

In cases where the device is connected directly to the USB Module, the bus resume request will be serviced immediately. In cases where the device is connected to the USB module via a hub, the application must call the [USB_HOST_DeviceIsResumed\(\)](#) function to check if the device has been resumed.

Example

```
USB_HOST_HANDLE handle; // Returned from USB_HOST_Open
USB_ERROR result;

// Resume the device with address 2
result = USB_HOST_DeviceResume(handle, 2);
if(USB_ERROR_NONE == result)
{
    // Wait till device is resumed.
    while(!USB_HOST_DeviceIsResumed(handle, 2));
}
```

5.9.8.4.1.6 USB_HOST_DeviceIsSuspended Function

C

```
bool USB_HOST_DeviceIsSuspended(
    USB_HOST_HANDLE hostHandle,
    uint8_t deviceAddress
);
```

Description

This function checks if the attached USB device has been suspended. This function can be called after calling the [USB_HOST_DeviceSuspend\(\)](#) function to check if suspend request has been completed.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the Host Layers's open routine
address	Address of the device.

Returns

true - Device is suspended. false - Device is not suspended or device address or handle is invalid.

Remarks

In cases where the device is connected directly to the USB Module, the bus suspend request will be serviced immediately. In cases where the device is connected to the USB module via a hub, the application must call the [USB_HOST_DeviceIsSuspended\(\)](#) function to check if the device has been suspended.

Example

```
USB_HOST_HANDLE handle; // Returned from USB_HOST_Open
USB_ERROR result;

// Suspend the device with address 2
result = USB_HOST_Device(handle, 2);
if(USB_ERROR_NONE == result)
{
    // Wait till device is suspended.
    while(!USB_HOST_DeviceIsSuspended(handle, 2));
}
```

5.9.8.4.1.7 USB_HOST_DeviceResume Function**C**

```
USB_ERROR USB_HOST_DeviceResume(
    USB_HOST_HANDLE hostHandle,
    uint8_t deviceAddress
);
```

Description

The function resumes the USB. If address is an address of a device that is connected directly to the USB module, this function will directly resume the bus. If the address is an address of a device that is connected to the module via a hub, the USB Host layer will resume the hub port to which this device is connected. The function only places a request to resume the device. The application must use the `USB_HOST_DeviceIsResumed()` function to check if the device resume is complete.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the Host Layers's open routine
address	Address of the device.

Returns

USB_ERROR_NONE - The request was accepted successfully. USB_ERROR_PARAMETER_INVALID - The device address or handle is invalid.

Remarks

In cases where the device is connected directly to the USB Module, the bus resume request will be serviced immediately. In cases where the device is connected to the USB module via a hub, the application must call the [USB_HOST_DeviceIsResumed\(\)](#) function to check if the device has been resumed.

Example

```
USB_HOST_HANDLE handle; // Returned from USB_HOST_Open
USB_ERROR result;

// Resume the device with address 2
result = USB_HOST_Device(handle, 2);
if(USB_ERROR_NONE == result)
{
    // Wait till device is resumed.
    while(!USB_HOST_DeviceIsResumed(handle, 2));
}
```

5.9.8.4.1.8 USB_HOST_DeviceSuspend Function

C

```
USB_ERROR USB_HOST_DeviceSuspend(  
    USB_HOST_HANDLE hostHandle,  
    uint8_t deviceAddress  
);
```

Description

The function suspends the USB. If address is an address of a device that is connected directly to the USB module, this function will directly suspend the bus. If the address is an address of a device that is connected to the module via a hub, the USB Host layer will suspend the hub port to which this device is connected. The function only places a request to suspend the device. The application must use the [USB_HOST_DeviceIsSuspended\(\)](#) function to check if the device suspend is complete.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the Host Layers's open routine
address	Address of the device.

Returns

USB_ERROR_NONE - The request was accepted successfully. USB_ERROR_PARAMETER_INVALID - The device address or handle is invalid.

Remarks

In cases where the device is connected directly to the USB Module, the bus suspend request will be serviced immediately. In cases where the device is connected to the USB module via a hub, the application must call the [USB_HOST_DeviceIsSuspended\(\)](#) function to check if the device has been suspended.

Example

```
USB_HOST_HANDLE handle; // Returned from USB_HOST_Open  
USB_ERROR result;  
  
// Suspend the device with address 2  
result = USB_HOST_Device(handle, 2);  
if(USB_ERROR_NONE == result)  
{  
    // Wait till device is suspended.  
    while(!USB_HOST_DeviceIsSuspended(handle, 2));  
}
```

5.9.8.4.1.9 USB_HOST_EventCallbackSet Function

C

```
USB_ERROR USB_HOST_EventCallbackSet(  
    USB_HOST_HANDLE hostHandle,  
    USB_HOST_EVENT_CALLBACK callback,  
    uintptr_t context  
);
```

Description

This function allows a client to identify an event handling function for the host layer to call back when host layer events are generated.

The event handler should be set before the client the host layer operation is enabled. The event handler once set, persists until the client closes the driver or sets another event handler.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified Host Layer instance.

[USB_HOST_Open](#) must have been called to obtain a valid opened Host Layer handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the host layer's open routine
callback	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Returns

USB_ERROR_NONE - Callback function was registered successfully. USB_ERROR_PARAMETER_INVALID - Callback function was NULL or hostHandle is invalid.

Remarks

None.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

USB_HOST_HANDLE usbHostHandle;

// usbHostHandle is the handle returned
// by the USB_HOST_Open function.

// Client registers an event handler with host layer.

USB_HOST_EventCallbackSet( usbHostHandle, APP_USBHostEventHandler,
                           (uintptr_t)&myAppObj );

USB_HOST_EVENT_RESPONSE APP_USBHostEventHandler(uintptr_t context,
        USB_HOST_EVENTS event, void * eventData)
{
    switch(event)
    {
        case USB_HOST_EVENT_UNSUPPORTED_DEVICE:
            break;
        ...
    }
    return(USB_HOST_EVENT_RESPONSE_NONE);
}
```

5.9.8.4.1.10 USB_HOST_Initialize Function

C

```
SYS_MODULE_OBJ USB_HOST_Initialize(
    const SYS_MODULE_INDEX index,
    const SYS_MODULE_INIT * init
);
```

Description

This routine initializes the USB Host Layer instance for the specified instance index, making it ready for clients to open and use it. The initialization data is specified by the `init` parameter. The initialization may fail if the number of host layer objects allocated are insufficient or if the specified host layer instance is already initialized. This function also initializes the USB Module driver for host mode operation. The `init` data structure thus contains information required for driver initialization. Note that initializing the Host Layer does not enable its operations. This must be done using [USB_HOST_OperationEnable\(\)](#) function.

Preconditions

None.

Parameters

Parameters	Description
<code>index</code>	Identifier for the instance to be initialized
<code>init</code>	Pointer to a USB_HOST_INIT data structure containing data necessary to initialize the driver.

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns [SYS_MODULE_OBJ_INVALID](#).

Remarks

This routine must be called before any other USB Host routine is called.

This routine should only be called once during system initialization unless [USB_HOST_Deinitialize](#) is called to de-initialize the Host Layer instance. This routine will NEVER block for hardware access.

Example

```
// Initializes the host layer for use with PIC32MX devices.
// We assume that the TPL table is already created and has two
// entries.

USB_HOST_TARGET_PERIPHERAL_LIST * tplTable;

USB_HOST_INIT    usbHostInit;
SYS_MODULE_OBJ   objectHandle;

// Allocate an endpoint table
uint8_t __attribute__((aligned(512))) endpointTable[USB_HOST_ENDPOINT_TABLE_SIZE];

usbHostInit.moduleInit = SYS_MODULE_POWER_RUN_FULL;

// The usbID member of the USB_HOST_INIT data structure
// should be USB_ID_x for PIC32MX devices and should be
// USBHS_ID_x PIC32MZ devices. Typical values are USB_ID_1
// for PIC32MX and USBHS_ID_0 for PIC32MZ devices.

usbHostInit.usbID      = USB_ID_1; //for PIC32MX devices
usbHostInit.stopInIdle = false;
usbHostInit.suspendInSleep = false;
usbHostInit.endpointTable = endpointTable;
usbHostInit.interruptSource = INT_SOURCE_USB_1;
usbHostInit.nTPLEntries  = 2;
usbHostInit.usbSpeed     = USB_SPEED_FULL;
usbHostInit.tplTable     = tplTable;

objectHandle = USB_HOST_Initialize(USB_HOST_0, (SYS_MODULE_INIT*)usbHostInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

5.9.8.4.1.11 USB_HOST_OperationDisable Function

C

```
void USB_HOST_OperationDisable(  
    USB_HOST_HANDLE hostHandle  
)  
;
```

Description

The function disables host operation. Disabling the host operation will cause the host to ignore attached device. All bus communication will be halted and USB bus will be suspended.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the Host Layers's open routine

Returns

None

Remarks

This function may not be used in a typical USB Host application. One possible use could be in a case where a fatal system error has occurred. Once disabled, the Host operation must be enabled again using the [USB_HOST_OperationEnable\(\)](#) function.

Example

```
USB_HOST_HANDLE handle; // Returned from USB_HOST_Open  
  
// Enabled host operation  
USB_HOST_OperationDisable(handle);
```

5.9.8.4.1.12 USB_HOST_OperationEnable Function

C

```
void USB_HOST_OperationEnable(  
    USB_HOST_HANDLE hostHandle  
)  
;
```

Description

The function enables host operation. When enabled, the host layer can detect and enumerate attached devices. The application must call the [USB_HOST_OperationIsEnabled\(\)](#) function to check if the operation has completed.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the Host Layers's open routine

Returns

None

Remarks

It is recommended that only on host layer client call this function. Multiple host layer clients calling this may cause indeterministic behavior.

Example

```
USB_HOST_HANDLE handle; // Returned from USB_HOST_Open

// Enabled host operation
USB_HOST_OperationEnable(handle);

// Wait till host operation is enabled.
while(!USB_HOST_OperationIsEnabled(handle));
```

5.9.8.4.1.13 USB_HOST_OperationIsEnabled Function**C**

```
bool USB_HOST_OperationIsEnabled(
    USB_HOST_HANDLE hostHandle
);
```

Description

The function allows the application to check if the host operation is enabled. This function should be called after the [USB_HOST_OperationEnable\(\)](#) function is called to check if the operation has been enabled.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the Host Layers's open routine

Returns

None.

Remarks

None.

Example

```
USB_HOST_HANDLE handle; // Returned from USB_HOST_Open

// Enabled host operation
USB_HOST_OperationEnable(handle);

// Wait till host operation is enabled.
while(!USB_HOST_OperationIsEnabled(handle));
```

5.9.8.4.1.14 USB_HOST_Status Function**C**

```
SYS_STATUS USB_HOST_Status(
    SYS_MODULE_OBJ driverObject
);
```

Description

This routine provides the current status of the USB Host Layer.

Preconditions

Function [USB_HOST_Initialize](#) should have been called before calling this function.

Parameters

Parameters	Description
object	USB Host Layer object handle, returned from the USB_HOST_Initialize routine

Returns

SYS_STATUS_READY - Indicates that the USB Host layer is ready for operations.

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been de-initialized

Remarks

None.

Example

```
SYS_MODULE_OBJ    object;    // Returned from USB_HOST_Initialize
SYS_STATUS        status;

status = USB_HOST_Status(object);
if (SYS_STATUS_READY == status)
{
    // This means the driver can be opened using the
    // USB_HOST_Open() function.
}
```

5.9.8.4.1.15 USB_HOST_Tasks Function**C**

```
void USB_HOST_Tasks(
    SYS_MODULE_OBJ driverObject
);
```

Description

This routine maintains the USB Host layer's state machine. It manages and the USB Module driver and responds to USB module driver events. This function should be called from the [SYS_Tasks\(\)](#) function.

Preconditions

The [USB_HOST_Initialize](#) routine must have been called for the specified USB Host Layer instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from USB_HOST_Initialize)

Returns

None.

Remarks

This routine is normally not called directly by an application.

Example

```
SYS_MODULE_OBJ    object;    // Returned from USB_HOST_Initialize

while (true)
{
    USB_HOST_Tasks (object);

    // Do other tasks
}
```


5.9.8.4.2 Data Types and Constants

5.9.8.4.2.1 USB_HOST_HANDLE Type

C

```
typedef uintptr_t USB_HOST_HANDLE;
```

Description

USB Host Client Handle

A USB Host Client Handle is returned by a call to the [USB_HOST_Open\(\)](#) function. The handle maintains the relationship between USB Host Layer and its client. All USB Host Client functions required the client handle to be specified when the functions are called. A client handle is valid till the the client calls the [USB_HOST_Close\(\)](#) function at which point it becomes invalid.

Remarks

None.

5.9.8.4.2.2 USB_HOST_INIT Structure

C

```
typedef struct {
    SYS_MODULE_INIT moduleInit;
    uint8_t usbID;
    bool stopInIdle;
    bool suspendInSleep;
    void * endpointTable;
    INT_SOURCE interruptSource;
    unsigned int nTPLEntries;
    USB_SPEED usbSpeed;
    USB_HOST_TARGET_PERIPHERAL_LIST * tplList;
} USB_HOST_INIT;
```

Description

USB Host Initialization Data Structure

This data type defines the data required to initialize an USB Host Layer Instance. A pointer to a structure of this type is required by the [USB_HOST_Initialize\(\)](#) function.

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System Module Initialization data
uint8_t usbID;	Identifies the Peripheral Library ID of the USB Peripheral to be used by this instance of the host layer
bool stopInIdle;	Identifies the Idle mode behavior. If true the USB module will stop when the microcontroller enter IDLE mode
bool suspendInSleep;	If true, the USB module will automatically suspend when the the microcontroller enter sleep mode
void * endpointTable;	Pointer to an endpoint table whose size should be USB_HOST_ENDPOINT_TABLE_SIZE number of bytes.
INT_SOURCE interruptSource;	Interrupt Peripheral Library ID of the interrupt source of USB module which this Host layer will control
unsigned int nTPLEntries;	Number of entries in the TPL table

USB_SPEED usbSpeed;	Speed at which the module should operate
USB_HOST_TARGET_PERIPHERAL_LIST * tplList;	Pointer to the TPL table

Remarks

Intentionally defined to be empty.

5.9.8.4.2.3 USB_HOST_HANDLE_INVALID Macro

C

```
#define USB_HOST_HANDLE_INVALID
```

Description

USB Host Invalid Client Handle

This is the definition of an invalid USB Host Client Handle. An invalid client handle is returned by the [USB_HOST_Open\(\)](#) function when it was not able to open the USB Host Layer successfully.

Remarks

None.

5.9.8.4.2.4 USB_DEVICE_INFORMATION Structure

C

```
typedef struct {
    uint16_t vid;
    uint16_t pid;
    uint16_t stringDescriptor[USB_HOST_DEVICE_INFORMATION_STRING_LENGTH];
} USB_DEVICE_INFORMATION;
```

Description

USB Device Information Structure

This data type defines the USB Device information data structure. This data structure is passed to [USB_HOST_DeviceInformationGet\(\)](#) function to obtain specific information about the device. The application can use this information to update its display or for other purposes. When processed by the [USB_HOST_DeviceInformationGet\(\)](#) function, the stringDescriptor member of this data structure will contain a NULL terminated string descriptor string in UTF-16 format. The vid and pid members of the data structure will be populated with VID and PID of the attached device respectively.

Members

Members	Description
uint16_t vid;	VID of the attached device
uint16_t pid;	PID of the attached device
uint16_t stringDescriptor[USB_HOST_DEVICE_INFORMATION_STRING_LENGTH];	String descriptor string in UTF-16 format. The string when returned by USB_HOST_DeviceInformationGet() function will be NULL terminated.

Remarks

None.

5.9.8.4.2.5 USB_HOST_EVENT_CALLBACK Type

C

```
typedef USB_HOST_EVENT_RESPONSE (* USB_HOST_EVENT_CALLBACK)(USB_HOST_EVENTS event, void *  
eventData, uintptr_t context);
```

Description

USB Host Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Host Layer event handling callback function. The application must register a pointer to a USB Host Event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the USB Host Layer. The function driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

event - Type of event generated.

eventData - This parameter should be type casted to a event specific pointer type based on the event that has occurred. Refer to the [USB_HOST_EVENTS](#) enumeration description for more details.

context - Value identifying the context of the application that registered the event handling function.

Remarks

None.

5.9.8.4.2.6 USB_HOST_EVENT_DATA_VBUS_REQUEST_POWER Structure

C

```
typedef struct {  
    uint8_t port;  
    uint8_t current;  
} USB_HOST_EVENT_DATA_VBUS_REQUEST_POWER;
```

Description

USB Host VBUS Power Request Event Data Structure

This data type defines the data returned along with the USB_HOST_EVENT_VBUS_REQUEST_POWER event.

Members

Members	Description
uint8_t port;	Port where the device is connected
uint8_t current;	Current (in 2mA units) that the device is requesting

Remarks

None.

5.9.8.4.2.7 USB_HOST_EVENT_RESPONSE Type

C

```
typedef void USB_HOST_EVENT_RESPONSE;
```

Description

USB Host Event Handler Response Type

This is the return type of the USB Host Layer Event Handler

Remarks

None.

5.9.8.4.2.8 USB_HOST_EVENTS Enumeration

C

```
typedef enum {
    USB_HOST_EVENT_VBUS_REQUEST_POWER,
    USB_HOST_EVENT_UNSUPPORTED_DEVICE,
    USB_HOST_EVENT_CANNOT_ENUMERATE
} USB_HOST_EVENTS;
```

Description

USB Host Events

This enumeration identifies the possible events that the host layer can generate. A Host Layer client should register an event handler using the [USB_HOST_EventCallBackSet\(\)](#) function to receive host layer events. An event may have data associated with it. The event description before provides details of such cases.

Members

Members	Description

5.9.8.4.2.9 USB_HOST_TPL_FLAGS Enumeration

C

```
typedef enum {
    TPL_FLAG_VID_PID = 0x1,
    TPL_FLAG_CLASS_SUBCLASS_PROTOCOL = 0x2
} USB_HOST_TPL_FLAGS;
```

Description

USB Host Layer TPL Table Entry Flags

This type defines the possible flags that can be used when adding an entry into the TPL table. Using the TPL_FLAG_VID_PID will direct the host layer to match the class driver to the attached device based on the device Vendor ID and the Product ID. Using the TPL_FLAG_CLASS_SUBCLASS_PROTOCOL will direct the host layer to match the class driver to the attached device based on the device class, subclass and protocol fields.

Members

Members	Description
TPL_FLAG_VID_PID = 0x1	Match by VID and PID
TPL_FLAG_CLASS_SUBCLASS_PROTOCOL = 0x2	Match by Class, Subclass and Protocol

Remarks

None.

5.9.8.4.2.10 TPL_MATCH_VID_PID Macro

C

```
#define TPL_MATCH_VID_PID(vid,pid)
```

Description

USB Host Class Driver Match by VID and PID

This macro when used in the TPL table will direct the Host Layer to match the corresponding class driver by device vendor ID and product ID. The application must set the device member of the TPL table entry to this macro in order to match the attached device to the class driver by device Vendor ID and Product ID.

Remarks

None.

5.9.8.4.2.11 USB_HOST_0 Macro

C

```
#define USB_HOST_0 0
```

Description

Driver USB Host Module Index

These constants provide USB Host index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the [USB_HOST_Initialize\(\)](#) and [USB_HOST_Open\(\)](#) routines to identify the host layer instance in use.

5.9.8.4.2.12 USB_HOST_1 Macro

C

```
#define USB_HOST_1 1
```

Description

This is macro USB_HOST_1.

5.9.8.4.2.13 USB_HOST_ENDPOINT_TABLE_SIZE Macro

C

```
#define USB_HOST_ENDPOINT_TABLE_SIZE
```

Description

USB Host Endpoint Table Size

This macro defines the USB Endpoint Table size while running the host layer on PIC32MX devices. The application should allocate a `uint8_t` type array of this size. The array should be aligned at a 512 byte boundary.

Remarks

None.

5.9.8.4.2.14 USB_HOST_EVENT_RESPONSE_NONE Macro

C

```
#define USB_HOST_EVENT_RESPONSE_NONE
```

Description

USB Host Layer Event Handler Response None

This is the definition of the USB Host Layer Event Handler Response Type None.

Remarks

Intentionally defined to be empty.

5.9.8.5 Files

Files

Name	Description
usb_host.h	USB HOST Layer Interface Header

Description

5.9.8.5.1 usb_host.h

USB Host Layer Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB HOST layer.

Enumerations

	Name	Description
	USB_HOST_EVENTS	Identifies the possible events that the host layer can generate.
	USB_HOST_TPL_FLAGS	USB Host Layer Target Peripheral List Entry flags

Functions

	Name	Description
	USB_HOST_Close	Closes an opened-instance of the USB Host Layer.
	USB_HOST_Deinitialize	Deinitializes the specified instance of the USB Host Layer.
	USB_HOST_DeviceInformationGet	Returns select information about the attached device.
	USB_HOST_DeviceIsResumed	Checks if the attached USB device has been resumed.
	USB_HOST_DeviceIsSuspended	Checks if the attached USB device has been suspended.
	USB_HOST_DeviceResume	Resumes the USB.
	USB_HOST_DeviceSuspend	Suspends the USB.
	USB_HOST_EventCallBackSet	Allows a client to identify an event handling function for the host layer to call back when host layer events are generated.
	USB_HOST_Initialize	Initializes the USB Host layer instance for the specified instance index.
	USB_HOST_Open	Opens the specified Host Layer instance and returns a handle to it
	USB_HOST_OperationDisable	Disabled host operation.
	USB_HOST_OperationEnable	Enables host operation.
	USB_HOST_OperationIsEnabled	Allows the application to check if the host operation is enabled.
	USB_HOST_Status	Gets the current status of the USB Host Layer.
	USB_HOST_Tasks	Maintains the USB Host's state machine. It manages the USB Module driver and responds to USB Module driver events.

Macros

	Name	Description
	<u>TPL_MATCH_VID_PID</u>	This macro when used in the TPL table will direct the Host Layer to match the corresponding class driver by device vendor ID (VID) and product ID (PID).
	<u>USB_HOST_0</u>	USB Host module index definitions
	<u>USB_HOST_1</u>	This is macro USB_HOST_1.
	<u>USB_HOST_ENDPOINT_TABLE_SIZE</u>	USB Host Endpoint Table size needed while running the host on PIC32MX devices.
	<u>USB_HOST_EVENT_RESPONSE_NONE</u>	USB Host Layer Event Handler Response Type None.
	<u>USB_HOST_HANDLE_INVALID</u>	Definition of an invalid USB Host Client Handle

Structures

	Name	Description
	<u>USB_DEVICE_INFORMATION</u>	Defines the USB Device Information data structure.
	<u>USB_HOST_EVENT_DATA_VBUS_REQUEST_POWER</u>	Defines the data returned along with the USB_HOST_EVENT_VBUS_REQUEST_POWER event.
	<u>USB_HOST_INIT</u>	Defines the data required to initialize a USB Host Layer instance.

Types

	Name	Description
	<u>USB_HOST_EVENT_CALLBACK</u>	USB Host Event Handler Function Pointer Type.
	<u>USB_HOST_EVENT_RESPONSE</u>	USB Host Event Handler Response Type
	<u>USB_HOST_HANDLE</u>	Handle identifying a client when calling Host layer client functions.

File Name

usb_host.h

Company

Microchip Technology Incorporated

5.9.9 USB CDC Host Library

5.9.9.1 Introduction

Introduces the MPLAB Harmony USB Host CDC Class Driver Library

Description

The USB Host CDC Class Driver library is a part of the USB Host stack that is available for Microchip family of microcontrollers. This library uses the USB Host Layer to interact with the attached USB CDC device and therefore cannot operate independently of it. Within the USB Host stack, the CDC Class Driver library is responsible for providing the host application with access to the functionality of the attached CDC Device.

The USB Host CDC Class Driver library implementation adheres to revision 1.2 of the USB CDC Specification release. The USB Host CDC Class Driver provides the following features:

- Support CDC-ACM devices.
- Abstracts CDC function interface details from the application and provides a clear interface to device functions.
- Provides functions to [send](#) commands such as Set Line Coding , Get Line Coding to the attached CDC Device.
- Provides functions to read and write data and request serial line notification from the device.
- Generates bus related events to user application and class drivers.
- Provides events to indicate completion and status of class driver functions.

5.9.9.2 Release Notes

MPLAB Harmony Version: v0.70b **USB CDC Host Library Version :** 0.01 Alpha **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

5.9.9.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

5.9.9.4 Library Interface

5.9.9.4.1 Functions

Functions

	Name	Description
	USB_HOST_CDC_BreakSend	This function sends a request to the attached to update it's break duration.
	USB_HOST_CDC_ControlLineStateSet	This function sets the state of the control line connecting the host to the device.
	USB_HOST_CDC_EventHandlerSet	This class registers a event handler for the specified CDC class driver instance.
	USB_HOST_CDC_LineCodingGet	This function requests line coding from the attached CDC device.
	USB_HOST_CDC_LineCodingSet	This function sets the line coding of the attached CDC device.
	USB_HOST_CDC_Read	This function reads data from the attached CDC device.
	USB_HOST_CDC_SerialStateNotificationGet	This function requests serial state noitification data from the attached CDC device.
	USB_HOST_CDC_Write	This function writes data to the attached CDC device.

Description

5.9.9.4.1.1 USB_HOST_CDC_BreakSend Function

C

```

USB_HOST_CDC_RESULT USB_HOST_CDC_BreakSend(
    USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,
    uint16_t breakDuration
);

```

Description

This function sends a request to the attached to update it's break duration. The function schedules a SEND BREAK control transfer. If successful, the transferHandle parameter will contain a valid transfer handle, else it will contain [USB_HOST_CDC_TRANSFER_HANDLE_INVALID](#). When completed, the CDC class driver will generate a USB_HOST_CDC_EVENT_SEND_BREAK_COMPLETE event. The CDC Class driver does not support queuing of control transfers. If a control transfer is in progress, this function will return with an error.

Preconditions

None.

Parameters

Parameters	Description
index	CDC Class Driver Instance Index where the request should be scheduled.
transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
breakDuration	Break duration.

Returns

USB_HOST_CDC_RESULT_OK - The operation was successful. transferHandle will contain a valid transfer handle.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
 USB_HOST_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - Another control transfer is in progress and this transfer cannot be scheduled.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_NOT_READY - The specified instance does not have

a CDC device associated with it. This can happen if the attached device was disconnected or the set request was called with an CDC class driver instance that was not attached with a CDC device.

Remarks

None.

Example

*// The following code snippet shows an example of scheduling a CDC Class
// Driver Control Line State Set Request.*

```
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
USB_HOST_CDC_RESULT result;
uint16_t breakDuration;

result = USB_HOST_CDC_BreakSend(USB_HOST_CDC_0,
                                &transferHandle, breakDuration);

if(USB_HOST_CDC_RESULT_OK != result)
{
    // Error handling here
}

// The completion of the set request can be tracked by using the
// USB_HOST_CDC_EVENT_SEND_BREAK_COMPLETE event.
```

5.9.9.4.1.2 USB_HOST_CDC_ControlLineStateSet Function

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_ControlLineStateSet(
    USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,
    USB_CDC_CONTROL_LINE_STATE * controlLineState
);
```

Description

This function sets the state of the control line connecting the host to the device. The function schedules a SET CONTROL LINE STATE control transfer. If successful, the transferHandle parameter will contain a valid transfer handle, else it will contain [USB_HOST_CDC_TRANSFER_HANDLE_INVALID](#). When completed, the CDC class driver will generate a USB_HOST_CDC_EVENT_SET_CONTROL_LINE_STATE_COMPLETE event. The CDC Class driver does not support queuing of control transfers. If a control transfer is in progress, this function will return with an error.

Preconditions

None.

Parameters

Parameters	Description
index	CDC Class Driver Instance Index where the request should be scheduled.
transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
controlLineState	pointer to the buffer containing control line state data to be sent to the device

Returns

USB_HOST_CDC_RESULT_OK - The operation was successful. transferHandle will contain a valid transfer handle.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
 USB_HOST_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - Another control transfer is in progress and this transfer cannot be scheduled.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_NOT_READY - The specified instance does not have a CDC device associated with it. This can happen if the attached device was disconnected or the set request was called with an

CDC class driver instance that was not attached with a CDC device.
USB_HOST_CDC_RESULT_ERROR_PARAMETER_INVALID - The controlLineState buffer pointer is NULL.

Remarks

None.

Example

```
// The following code snippet shows an example of scheduling a CDC Class  
// Driver Control Line State Set Request.
```

```
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;  
USB_HOST_CDC_RESULT result;  
USB_CDC_HOST_CONTROL_LINE_STATE * buffer;  
  
result = USB_HOST_CDC_ControlLineStateSet(USB_HOST_CDC_0,  
                                           &transferHandle, buffer);  
  
if(USB_HOST_CDC_RESULT_OK != result)  
{  
    // Error handling here  
}
```

```
// The completion of the set request can be tracked by using the  
// USB_HOST_CDC_EVENT_SET_CONTROL_LINE_STATE_COMPLETE event.
```

5.9.9.4.1.3 USB_HOST_CDC_EventHandlerSet Function**C**

```
USB_HOST_CDC_RESULT USB_HOST_CDC_EventHandlerSet(  
    USB_HOST_CDC_INDEX index,  
    USB_HOST_CDC_EVENT_HANDLER eventHandler,  
    uintptr_t context  
);
```

Description

This function registers an application event handler for the specified CDC class driver instance events. The event handler should be registered before the host layer operation is enabled.

Preconditions

None.

Parameters

Parameters	Description
instance	Instance of the CDC Class Driver.
eventHandler	A pointer to event handler function.
context	Application specific context that is returned in the event handler.

Returns

USB_HOST_CDC_RESULT_OK - The operation was successful
USB_HOST_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
USB_HOST_CDC_RESULT_ERROR_PARAMETER_INVALID - The eventHandler parameter is NULL

Remarks

None.

Example

```
// This code snippet shows an example of registering an event handler. Here  
// the application specifies the context parameter as a pointer to an
```

```

// application object (appObject) that should be associated with this
// instance of the CDC class driver.

USB_HOST_CDC_RESULT result;

USB_HOST_CDC_EVENT_RESPONSE APP_USBHostCDCEventHandler
(
    USB_HOST_CDC_INDEX instanceIndex ,
    USB_HOST_CDC_EVENT event ,
    void * event, uintptr_t context
)
{
    // Event Handling comes here

    switch(event)
    {
        ...
    }

    return(USB_HOST_CDC_EVENT_RESPONSE_NONE);
}

result = USB_HOST_CDC_EventHandlerSet ( 0 ,&APP_EventHandler, (uintptr_t) &appObject);

if(USB_HOST_CDC_RESULT_OK != result)
{
    SYS_ASSERT ( false , "invalid event handler class" );
}

```

5.9.9.4.1.4 USB_HOST_CDC_LineCodingGet Function

C

```

USB_HOST_CDC_RESULT USB_HOST_CDC_LineCodingGet(
    USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,
    USB_CDC_LINE_CODING * lineCoding
);

```

Description

This function requests line coding from the attached CDC device. The function schedules a GET LINE CODING control transfer. If successful, the transferHandle parameter will contain a valid transfer handle, else it will contain [USB_HOST_CDC_TRANSFER_HANDLE_INVALID](#). When completed, the CDC class driver will generate a USB_HOST_CDC_EVENT_GET_LINE_CODING_COMPLETE event. When completed, the lineCoding parameter will contain the line coding. The CDC Class driver does not support queuing of control transfers. If a control transfer is in progress, this function will return with an error.

Preconditions

None.

Parameters

Parameters	Description
index	CDC Class Driver Instance Index where the request should be scheduled.
transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
lineCoding	pointer to the buffer where the line coding data will be stored.

Returns

USB_HOST_CDC_RESULT_OK - The operation was successful. transferHandle will contain a valid transfer handle.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.

USB_HOST_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - Another control transfer is in progress and this transfer cannot be scheduled. USB_HOST_CDC_RESULT_ERROR_INSTANCE_NOT_READY - The specified instance does not have a CDC device associated with it. This can happen if the attached device was disconnected or the get request was called with an CDC class driver instance that was not attached with a CDC device. USB_HOST_CDC_RESULT_ERROR_PARAMETER_INVALID - The lineCoding buffer pointer is NULL.

Remarks

None.

Example

*// The following code snippet shows an example of scheduling a CDC Class
// Driver Get Line Coding Request.*

```
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
USB_HOST_CDC_RESULT result;
USB_CDC_LINE_CODING * buffer;

result = USB_HOST_CDC_LineCodingGet(USB_HOST_CDC_0,
                                   &transferHandle, buffer);

if(USB_HOST_CDC_RESULT_OK != result)
{
    // Error handling here
}

// The completion of the get request can be tracked by using the
// USB_HOST_CDC_EVENT_GET_LINE_CODING_COMPLETE event.
```

5.9.9.4.1.5 USB_HOST_CDC_LineCodingSet Function

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_LineCodingSet(
    USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,
    USB_CDC_LINE_CODING * lineCoding
);
```

Description

This function sets the line coding of the attached CDC device. The function schedules a SET LINE CODING control transfer. If successful, the transferHandle parameter will contain a valid transfer handle, else it will contain [USB_HOST_CDC_TRANSFER_HANDLE_INVALID](#). When completed, the CDC class driver will generate a USB_HOST_CDC_EVENT_SET_LINE_CODING_COMPLETE event. The lineCoding parameter contain the line coding to be sent to the attached device. The CDC Class driver does not support queuing of control transfers. If a control transfer is in progress, this function will return with an error.

Preconditions

None.

Parameters

Parameters	Description
index	CDC Class Driver Instance Index where the request should be scheduled.
transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
lineCoding	pointer to the buffer containing line coding data to be sent to the device

Returns

USB_HOST_CDC_RESULT_OK - The operation was successful. transferHandle will contain a valid transfer handle.

USB_HOST_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
 USB_HOST_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - Another control transfer is in progress and this transfer cannot be scheduled.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_NOT_READY - The specified instance does not have a CDC device associated with it. This can happen if the attached device was disconnected or the set request was called with an CDC class driver instance that was not attached with a CDC device.
 USB_HOST_CDC_RESULT_ERROR_PARAMETER_INVALID - The lineCoding buffer pointer is NULL.

Remarks

None.

Example

*// The following code snippet shows an example of scheduling a CDC Class
 // Driver Set Line Coding Request.*

```
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
USB_HOST_CDC_RESULT result;
USB_CDC_LINE_CODING * buffer;

result = USB_HOST_CDC_LineCodingSet(USB_HOST_CDC_0,
                                     &transferHandle, buffer);

if(USB_HOST_CDC_RESULT_OK != result)
{
    // Error handling here
}

// The completion of the set request can be tracked by using the
// USB_HOST_CDC_EVENT_SET_LINE_CODING_COMPLETE event.
```

5.9.9.4.1.6 USB_HOST_CDC_Read Function**C**

```
USB_HOST_CDC_RESULT USB_HOST_CDC_Read(
    USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,
    void * destination,
    size_t length
);
```

Description

This function reads data from the attached CDC device. The function will schedule a read transfer. If successful, the transferHandle parameter will contain a valid transfer handle, else it will contain [USB_HOST_CDC_TRANSFER_HANDLE_INVALID](#). When completed, the CDC class driver will generate a USB_HOST_CDC_EVENT_READ_COMPLETE event. Multiple read requests can be queued. In such a case, the transfer handle for each request will be unique.

Preconditions

None.

Parameters

Parameters	Description
index	CDC Class Driver Instance Index where the request should be scheduled.
transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
dest	pointer to the buffer where the read data will be stored.
length	Amount of data to read (in bytes).

Returns

USB_HOST_CDC_RESULT_OK - The operation was successful. transferHandle will contain a valid transfer handle.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
 USB_HOST_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - The transfer queue is full and the requested transfer cannot be scheduled.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_NOT_READY - The specified instance does not have a CDC device associated with it. This can happen if the attached device was disconnected or the read function was called with an CDC class driver instance that was not attached to a CDC device.
 USB_HOST_CDC_RESULT_ERROR_PARAMETER_INVALID - The destination buffer pointer is NULL or the length parameter is zero.

Remarks

None.

Example

*// The following code snippet shows an example of scheduling a CDC Class
 // Driver read. In this example, the class driver reads 64 bytes from the
 // attached device.*

```
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
USB_HOST_CDC_RESULT result;
uint8_t buffer[64];

result = USB_HOST_CDC_Read(USB_HOST_CDC_0, &transferHandle, buffer, 64);

if(USB_HOST_CDC_RESULT_OK != result)
{
    // Error handling here
}

// The completion of the read request can be tracked by using the
// USB_HOST_CDC_EVENT_READ_COMPLETE.
```

5.9.9.4.1.7 USB_HOST_CDC_SerialStateNotificationGet Function**C**

```
USB_HOST_CDC_RESULT USB_HOST_CDC_SerialStateNotificationGet(
    USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,
    USB_HOST_CDC_SERIAL_STATE_NOTIFICATION_DATA * dest
);
```

Description

This function requests serial state notification data from the attached CDC device. The function will schedule a read transfer over the CDC notification interface. If successful, the transferHandle parameter will contain a valid transfer handle, else it will contain [USB_HOST_CDC_TRANSFER_HANDLE_INVALID](#). When completed, the CDC class driver will generate a USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED event. Multiple get requests can be queued. In such a case, the transfer handle for each request will be unique.

Preconditions

None.

Parameters

Parameters	Description
index	CDC Class Driver Instance Index where the request should be scheduled.
transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.

dest	pointer to the buffer where the serial state notification data should be stored.

Returns

USB_HOST_CDC_RESULT_OK - The operation was successful. transferHandle will contain a valid transfer handle.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
 USB_HOST_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - The transfer queue is full and the requested transfer cannot be scheduled.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_NOT_READY - The specified instance does not have a CDC device associated with it. This can happen if the attached device was disconnected or the get request was called with an CDC class driver instance that was not attached with a CDC device.
 USB_HOST_CDC_RESULT_ERROR_PARAMETER_INVALID - The source buffer pointer is NULL.

Remarks

None.

Example

*// The following code snippet shows an example of scheduling a CDC Class
 // Driver Serial State Notification Get function.*

```
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
USB_HOST_CDC_RESULT result;
USB_HOST_CDC_SERIAL_STATE_NOTIFICATION_DATA * buffer;

result = USB_HOST_CDC_SerialStateNotificationGet(USB_HOST_CDC_0,
                                                &transferHandle, buffer);

if(USB_HOST_CDC_RESULT_OK != result)
{
    // Error handling here
}

// The completion of the get request can be tracked by using the
// USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED.
```

5.9.9.4.1.8 USB_HOST_CDC_Write Function**C**

```
USB_HOST_CDC_RESULT USB_HOST_CDC_Write(
    USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,
    void * source,
    size_t length
);
```

Description

This function writes data to the attached CDC device. The function will schedule a write transfer. If successful, the transferHandle parameter will contain a valid transfer handle, else it will contain **USB_HOST_CDC_TRANSFER_HANDLE_INVALID**. When completed, the CDC class driver will generate a USB_HOST_CDC_EVENT_WRITE_COMPLETE event. Multiple write requests can be queued. In such a case, the transfer handle for each request will be unique.

Preconditions

None.

Parameters

Parameters	Description
index	CDC Class Driver Instance Index where the request should be scheduled.

transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
source	pointer to the buffer containing data to be written to the device.
length	Amount of data to written (in bytes).

Returns

USB_HOST_CDC_RESULT_OK - The operation was successful. transferHandle will contain a valid transfer handle.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
 USB_HOST_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - The transfer queue is full and the requested transfer cannot be scheduled.
 USB_HOST_CDC_RESULT_ERROR_INSTANCE_NOT_READY - The specified instance does not have a CDC device associated with it. This can happen if the attached device was disconnected or the write function was called with an CDC class driver instance that was not attached to a CDC device.
 USB_HOST_CDC_RESULT_ERROR_PARAMETER_INVALID - The source buffer pointer is NULL or the length parameter is zero.

Remarks

None.

Example

// The following code snippet shows an example of scheduling a CDC Class Driver Write. In this example, the class driver writes 64 bytes to the attached device.

```

USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
USB_HOST_CDC_RESULT result;
uint8_t buffer[64];

result = USB_HOST_CDC_Write(USB_HOST_CDC_0, &transferHandle, buffer, 64);

if(USB_HOST_CDC_RESULT_OK != result)
{
    // Error handling here
}

// The completion of the write request can be tracked by using the
// USB_HOST_CDC_EVENT_WRITE_COMPLETE.
```

5.9.9.4.2 Data Types and Constants**Enumerations**

	Name	Description
	USB_HOST_CDC_EVENT	Identifies the possible events that the CDC Class Driver can generate.
	USB_HOST_CDC_RESULT	USB Host CDC Class Driver CDC Result enumeration.
	USB_HOST_CDC_TRANSFER_STATUS	USB Host CDC Class Driver Transfer Status enumeration.

Macros

	Name	Description
	USB_HOST_CDC_EVENT_RESPONSE_NONE	USB Host CDC Class Driver Event Handler Response Type None.
	USB_HOST_CDC_TRANSFER_HANDLE_INVALID	USB Host CDC Class Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE	USB Host CDC Class Driver Event Data.
	USB_HOST_CDC_EVENT_DATA_READ_COMPLETE	USB Host CDC Class Driver Event Data.
	USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE	USB Host CDC Class Driver Event Data.
	USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED	USB Host CDC Class Driver Event Data.
	USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE	USB Host CDC Class Driver Event Data.
	USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE	USB Host CDC Class Driver Event Data.
	USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE	USB Host CDC Class Driver Event Data.

Types

	Name	Description
	USB_HOST_CDC_TRANSFER_HANDLE	USB Host CDC Class Driver Transfer Handle
	USB_HOST_CDC_EVENT_HANDLER	USB Host CDC Class Driver Event Handler Function Pointer Type.
	USB_HOST_CDC_EVENT_RESPONSE	USB Host CDC Class Driver Event Callback Response Type
	USB_HOST_CDC_INDEX	USB Host CDC Class Driver Index
	USB_HOST_CDC_SERIAL_STATE_NOTIFICATION_DATA	USB Host Serial State Notification Data Type.
	USB_HOST_CDC_CONTROL_LINE_STATE	USB Host Control Line State Data Type.

Description

5.9.9.4.2.1 USB_HOST_CDC_EVENT Enumeration

C

```
typedef enum {
    USB_HOST_CDC_EVENT_READ_COMPLETE,
    USB_HOST_CDC_EVENT_WRITE_COMPLETE,
    USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED,
    USB_HOST_CDC_EVENT_GET_LINE_CODING_COMPLETE,
    USB_HOST_CDC_EVENT_SET_LINE_CODING_COMPLETE,
    USB_HOST_CDC_EVENT_SET_CONTROL_LINE_STATE_COMPLETE,
    USB_HOST_CDC_EVENT_SEND_BREAK_COMPLETE,
    USB_HOST_CDC_EVENT_ATTACH,
    USB_HOST_CDC_EVENT_DETACH
} USB_HOST_CDC_EVENT;
```

Description

CDC Class Driver Events

This enumeration identifies the possible events that the CDC Class Driver can generate. The application should register an event handler using the [USB_HOST_CDC_EventHandlerSet\(\)](#) function to receive CDC Class Driver events.

An event may have data associated with it. Events that are generated due to a transfer of data between the host and device are accompanied by data structures that provide the status of the transfer termination. For example, the

USB_HOST_CDC_EVENT_SET_LINE_CODING_COMPLETE event is accompanied by a pointer to a [USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE](#) data structure. The transferStatus member of this data structure indicates the success or failure of the transfer. A transfer may fail due to device not responding on the bus, if the device stalls any stages of the transfer or due to NAK timeouts. The event description provides details on the nature of the event and the data that is associated with the event.

The following code snippet shows an example event handling scheme for

```
// This code snippet shows an example event handling scheme for
// CDC Class Driver events.
```

```
USB_HOST_CDC_EVENT_RESPONSE APP_USBHostCDCEventHandler
(
    USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    uint8_t deviceAddress;
    USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE * setLineCodingEventData;
    USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE * getLineCodingEventData;
    USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE * setControlLineStateEventData;
    USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE * sendBreakEventData;
    USB_HOST_CDC_EVENT_DATA_READ_COMPLETE * readCompleteEventData;
    USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE * writeCompleteEventData;
    USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED *
serialStateNotificationEventData;

    switch(event)
    {
        case USB_HOST_CDC_EVENT_ATTACH:

            // The event data in this case is the address of the
            // attached device.

            deviceAddress = (uint8_t *)eventData;
            break;

        case USB_HOST_CDC_EVENT_DETACH:

            // This means the device was detached. There is no event data
            // associated with this event.

            break;

        case USB_HOST_CDC_EVENT_SET_LINE_CODING_COMPLETE:

            // This means the Set Line Coding request completed. We can
            // find out if the request was successful.

            setLineCodingEventData =
                (USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE *)eventData;
            if(setLineCodingEventData->transferStatus
                == USB_HOST_CDC_TRANSFER_STATUS_ERROR)
            {
                // This means there transfer terminated because of an
                // error.
            }
            break;

        case USB_HOST_CDC_EVENT_GET_LINE_CODING_COMPLETE:

            // This means the Get Line Coding request completed. We can
            // find out if the request was successful.
```

```

    getLineCodingEventData =
        (USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE *)eventData;
    if(getLineCodingEventData->transferStatus
        == USB_HOST_CDC_TRANSFER_STATUS_ERROR)
    {
        // This means there transfer terminated because of an
        // error.
    }

case USB_HOST_CDC_EVENT_SET_CONTROL_LINE_STATE_COMPLETE:

    // This means the Set Control Line State request completed. We can
    // find out if the request was successful.

    setControlLineStateEventData =
        (USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE *)eventData;
    if(setControlLineStateEventData->transferStatus
        == USB_HOST_CDC_TRANSFER_STATUS_ERROR)
    {
        // This means there transfer terminated because of an
        // error.
    }
    break;

case USB_HOST_CDC_EVENT_SEND_BREAK_COMPLETE:

    // This means the Send Break request completed. We can
    // find out if the request was successful.

    sendBreakEventData =
        (USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE *)eventData;
    if(sendBreakEventData->transferStatus
        == USB_HOST_CDC_TRANSFER_STATUS_ERROR)
    {
        // This means there transfer terminated because of an
        // error.
    }
    break;

case USB_HOST_CDC_EVENT_WRITE_COMPLETE:

    // This means the Write request completed. We can
    // find out if the request was successful.

    writeCompleteEventData =
        (USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE *)eventData;
    if(writeCompleteEventData->transferStatus
        == USB_HOST_CDC_TRANSFER_STATUS_ERROR)
    {
        // This means there transfer terminated because of an
        // error.
    }
    break;

case USB_HOST_CDC_EVENT_READ_COMPLETE:

    // This means the Read request completed. We can
    // find out if the request was successful.

    readCompleteEventData =
        (USB_HOST_CDC_EVENT_DATA_READ_COMPLETE *)eventData;
    if(readCompleteEventData->transferStatus
        == USB_HOST_CDC_TRANSFER_STATUS_ERROR)
    {
        // This means there transfer terminated because of an

```

```

        // error.
    }
    break;

case USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED:

    // This means the Serial State Notification request completed.
    // We can find out if the request was successful.

    serialStateNotificationEventData =
        (USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED *)eventData;
    if(serialStateNotificationEventData->transferStatus
        == USB_HOST_CDC_TRANSFER_STATUS_ERROR)
    {
        // This means there transfer terminated because of an
        // error.
    }
    break;

default:
    break;
}
return(USB_HOST_CDC_EVENT_RESPONSE_NONE);
}

```

Members

Members	Description
USB_HOST_CDC_EVENT_READ_COMPLETE	This event occurs when a CDC Class Driver Read operation has completed i.e when the data has been received from the connected CDC device. This event is generated after the application calls the USB_HOST_CDC_Read() function. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_DATA_READ_COMPLETE structure. This contains details about the transfer handle associated with this read request, the amount of data read and the termination status of the read request.
USB_HOST_CDC_EVENT_WRITE_COMPLETE	This event occurs when a CDC Class Driver Write operation has completed i.e when the data has been written to the connected CDC device. This event is generated after the application calls the USB_HOST_CDC_Write() function. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE structure. This contains details about the transfer handle associated with this write request, the amount of data written and the termination status of the write request.

USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED	This event occurs when a CDC Class Driver Serial State Notification Get operation has completed. This event is generated after the application calls the USB_HOST_CDC_SerialStateNotificationGet() and the device sends a serial state notification to the host. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED structure. This contains details about the transfer handle associated with this request, the amount of data received and the termination status of the get request.
USB_HOST_CDC_EVENT_GET_LINE_CODING_COMPLETE	This event occurs when a CDC Class Driver Get Line Coding request has completed. This event is generated after the application calls the USB_HOST_CDC_LineCodingGet() function and the device sends the line coding to the host. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE structure. This contains details about the transfer handle associated with this request, the amount of data received and the termination status of the get request.
USB_HOST_CDC_EVENT_SET_LINE_CODING_COMPLETE	This event occurs when a CDC Class Driver Set Line Coding request has completed. This event is generated after the application calls the USB_HOST_CDC_LineCodingSet() function and the device either acknowledged or stalled the request. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE structure. This contains details about the transfer handle associated with this request, the amount of data sent and the termination status of the set request.
USB_HOST_CDC_EVENT_SET_CONTROL_LINE_STATE_COMPLETE	This event occurs when a CDC Class Driver Set Control Line State request has completed. This event is generated after the application calls the USB_HOST_CDC_ControlLineStateSet() function and the device has either acknowledged or stalled the request. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE structure. This contains details about the transfer handle associated with this request, the amount of data sent and the termination status of the set request.

USB_HOST_CDC_EVENT_SEND_BREAK_COMPLETE	This event occurs when a CDC Class Driver Send Break request has completed. This event is generated after the application calls the USB_HOST_CDC_BreakSend() function and the device has either acknowledged or stalled the request. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE structure. This contains details about the transfer handle associated with this request, the amount of data sent and the termination status of the set request.
USB_HOST_CDC_EVENT_ATTACH	This event occurs when the host layer has successfully attached the CDC Class Driver instance to the attached USB CDC device. The event also indicates that the class driver is fully initialized and is ready to accept command and data transfer requests from the application. The eventData parameter should be interpreted as a <code>uint8_t</code> pointer to the device address.
USB_HOST_CDC_EVENT_DETACH	This event occurs when host layer has detached the CDC class driver instance from a USB CDC device. This can happen if the device itself was detached, or if the device configuration was changed. There is no event data associated with this event.

Remarks

None.

5.9.9.4.2.2 USB_HOST_CDC_RESULT Enumeration

C

```
typedef enum {
    USB_HOST_CDC_RESULT_OK,
    USB_HOST_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL,
    USB_HOST_CDC_RESULT_ERROR_INSTANCE_INVALID,
    USB_HOST_CDC_RESULT_ERROR_INSTANCE_NOT_READY,
    USB_HOST_CDC_RESULT_ERROR_PARAMETER_INVALID
} USB_HOST_CDC_RESULT;
```

Description

USB Host CDC Class Driver Result enumeration.

This enumeration lists the possible USB Host CDC Class Driver operation results. These values are returned by CDC Class Driver functions.

Members

Members	Description
USB_HOST_CDC_RESULT_OK	The operation was successful

5.9.9.4.2.3 USB_HOST_CDC_TRANSFER_HANDLE Type

C

```
typedef uintptr_t USB_HOST_CDC_TRANSFER_HANDLE;
```

Description

USB Host CDC Class Driver Transfer Handle

This is returned by the CDC Class driver command and data transfer routines and should be used by the application to track the transfer especially in cases where transfers are queued.

Remarks

None.

5.9.9.4.2.4 USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE Structure

C

```
typedef struct {
    USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
    USB_HOST_CDC_TRANSFER_STATUS transferStatus;
    size_t length;
} USB_HOST_CDC_EVENT_DATA_READ_COMPLETE, USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED,
USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE;
```

Description

USB Host CDC Class Driver Event Data.

This data type defines the data structure returned by the driver along with the following events:

[USB_HOST_CDC_EVENT_DATA_READ_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED](#)

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
USB_HOST_CDC_TRANSFER_STATUS transferStatus;	Termination transfer status
size_t length;	Amount of data transferred

Remarks

None.

5.9.9.4.2.5 USB_HOST_CDC_EVENT_DATA_READ_COMPLETE Structure

C

```
typedef struct {
    USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
    USB_HOST_CDC_TRANSFER_STATUS transferStatus;
    size_t length;
} USB_HOST_CDC_EVENT_DATA_READ_COMPLETE, USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED,
USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE,
```



```
USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE;
```

Description

USB Host CDC Class Driver Event Data.

This data type defines the data structure returned by the driver along with the following events:

```
USB_HOST_CDC_EVENT_DATA_READ_COMPLETE,
USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE,
USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED
```

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
USB_HOST_CDC_TRANSFER_STATUS transferStatus;	Termination transfer status
size_t length;	Amount of data transferred

Remarks

None.

5.9.9.4.2.6 USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE Structure

C

```
typedef struct {
    USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
    USB_HOST_CDC_TRANSFER_STATUS transferStatus;
    size_t length;
} USB_HOST_CDC_EVENT_DATA_READ_COMPLETE, USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED,
USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE;
```

Description

USB Host CDC Class Driver Event Data.

This data type defines the data structure returned by the driver along with the following events:

```
USB_HOST_CDC_EVENT_DATA_READ_COMPLETE,
USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE,
USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED
```

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer

USB_HOST_CDC_TRANSFER_STATUS transferStatus;	Termination transfer status
size_t length;	Amount of data transferred

Remarks

None.

5.9.9.4.2.7 USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED Structure

C

```
typedef struct {
    USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
    USB_HOST_CDC_TRANSFER_STATUS transferStatus;
    size_t length;
} USB_HOST_CDC_EVENT_DATA_READ_COMPLETE, USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED,
USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE;
```

Description

USB Host CDC Class Driver Event Data.

This data type defines the data structure returned by the driver along with the following events:

[USB_HOST_CDC_EVENT_DATA_READ_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED](#)

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
USB_HOST_CDC_TRANSFER_STATUS transferStatus;	Termination transfer status
size_t length;	Amount of data transferred

Remarks

None.

5.9.9.4.2.8 USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE Structure

C

```
typedef struct {
    USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
    USB_HOST_CDC_TRANSFER_STATUS transferStatus;
    size_t length;
} USB_HOST_CDC_EVENT_DATA_READ_COMPLETE, USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED,
USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE,
```

```

USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE;

```

Description

USB Host CDC Class Driver Event Data.

This data type defines the data structure returned by the driver along with the following events: [USB_HOST_CDC_EVENT_DATA_READ_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED](#)

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
USB_HOST_CDC_TRANSFER_STATUS transferStatus;	Termination transfer status
size_t length;	Amount of data transferred

Remarks

None.

5.9.9.4.2.9 USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE Structure

C

```

typedef struct {
    USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
    USB_HOST_CDC_TRANSFER_STATUS transferStatus;
    size_t length;
} USB_HOST_CDC_EVENT_DATA_READ_COMPLETE, USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED,
USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE;

```

Description

USB Host CDC Class Driver Event Data.

This data type defines the data structure returned by the driver along with the following events: [USB_HOST_CDC_EVENT_DATA_READ_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED](#)

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer

USB_HOST_CDC_TRANSFER_STATUS transferStatus;	Termination transfer status
size_t length;	Amount of data transferred

Remarks

None.

5.9.9.4.2.10 USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE Structure

C

```
typedef struct {
    USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
    USB_HOST_CDC_TRANSFER_STATUS transferStatus;
    size_t length;
} USB_HOST_CDC_EVENT_DATA_READ_COMPLETE, USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED,
USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE,
USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE;
```

Description

USB Host CDC Class Driver Event Data.

This data type defines the data structure returned by the driver along with the following events:
[USB_HOST_CDC_EVENT_DATA_READ_COMPLETE](#), [USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE](#),
[USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED](#)

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
USB_HOST_CDC_TRANSFER_STATUS transferStatus;	Termination transfer status
size_t length;	Amount of data transferred

Remarks

None.

5.9.9.4.2.11 USB_HOST_CDC_EVENT_HANDLER Type

C

```
typedef USB_HOST_CDC_EVENT_RESPONSE (* USB_HOST_CDC_EVENT_HANDLER)(USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_EVENT event, void * eventData, uintptr_t context);
```

Description

USB Host CDC Class Driver Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Host CDC Class Driver event handling callback function. The application must register a pointer to a CDC Class Driver events handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the CDC Class

Driver. The class driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

instanceIndex - Instance index of the CDC Class Driver that generated the event.

event - Type of event generated.

eventData - This parameter should be type casted to a event specific pointer type based on the event that has occurred. Refer to the [USB_HOST_CDC_EVENT](#) enumeration description for more details.

context - Value identifying the context of the application that was registered along with the event handling function.

Remarks

None.

5.9.9.4.2.12 USB_HOST_CDC_EVENT_RESPONSE Type**C**

```
typedef void USB_HOST_CDC_EVENT_RESPONSE;
```

Description

USB Host CDC Class Driver Event Handler Response Type

This is the return type of the CDC Class Driver event handler.

Remarks

None.

5.9.9.4.2.13 USB_HOST_CDC_INDEX Type**C**

```
typedef uintptr_t USB_HOST_CDC_INDEX;
```

Description

USB Host CDC Class Driver Index

This uniquely identifies a CDC Class Driver instance.

Remarks

None.

5.9.9.4.2.14 USB_HOST_CDC_SERIAL_STATE_NOTIFICATION_DATA Type**C**

```
typedef USB_CDC_SERIAL_STATE USB_HOST_CDC_SERIAL_STATE_NOTIFICATION_DATA;
```

Description

USB Host CDC Serial State Notification Data Type.

This defines the data type for the CDC Serial State. This data is requested from the device over the CDC notification interface.

Remarks

None.

5.9.9.4.2.15 USB_HOST_CDC_TRANSFER_STATUS Enumeration

C

```
typedef enum {
    USB_HOST_CDC_TRANSFER_STATUS_OK,
    USB_HOST_CDC_TRANSFER_STATUS_ERROR
} USB_HOST_CDC_TRANSFER_STATUS;
```

Description

USB Host CDC Class Driver Transfer Status.

This enumeration lists the possible termination status of CDC class driver transfers. This information is a part of the event data that is returned along with transfer related events.

Members

Members	Description
USB_HOST_CDC_TRANSFER_STATUS_OK	The transfer completed successfully. In case of read transfers, either the requested amount of data was received or the device sent less data
USB_HOST_CDC_TRANSFER_STATUS_ERROR	The transfer terminated due to an error. The error could be due to device malfunction on the bus, a NAK timeout occurred or the device stalled the request

Remarks

None.

5.9.9.4.2.16 USB_HOST_CDC_EVENT_RESPONSE_NONE Macro

C

```
#define USB_HOST_CDC_EVENT_RESPONSE_NONE
```

Description

USB Host CDC Class Driver Event Handler Response None

This is the definition of the CDC Class Driver Event Handler Response Type none.

Remarks

Intentionally defined to be empty.

5.9.9.4.2.17 USB_HOST_CDC_TRANSFER_HANDLE_INVALID Macro

C

```
#define USB_HOST_CDC_TRANSFER_HANDLE_INVALID ((USB_HOST_CDC_TRANSFER_HANDLE) (-1))
```

Description

USB Host CDC Class Driver Invalid Transfer Handle Definition

This definition defines a USB Host CDC Class Driver Invalid Transfer Handle. A Invalid Transfer Handle is returned by the CDC Class Driver data and command transfer routines when the request was not successful.

Remarks

None.

5.9.9.4.2.18 USB_HOST_CDC_CONTROL_LINE_STATE Type

C

```
typedef USB_CDC_CONTROL_LINE_STATE USB_HOST_CDC_CONTROL_LINE_STATE;
```

Description

USB Host CDC Control Line State Data Type.

This defines the data type for the CDC Control Line State. This data is requested from the device over the CDC control interface.

Remarks

None.

5.9.9.5 Files

Files

Name	Description
usb_host_cdc.h	USB Host CDC Class Driver Interface Header

Description

5.9.9.5.1 usb_host_cdc.h

USB Host CDC Class Driver Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB Host CDC Class Driver.

Enumerations

	Name	Description
	USB_HOST_CDC_EVENT	Identifies the possible events that the CDC Class Driver can generate.
	USB_HOST_CDC_RESULT	USB Host CDC Class Driver CDC Result enumeration.
	USB_HOST_CDC_TRANSFER_STATUS	USB Host CDC Class Driver Transfer Status enumeration.

Functions

	Name	Description
	USB_HOST_CDC_BreakSend	This function sends a request to the attached to update it's break duration.
	USB_HOST_CDC_ControlLineStateSet	This function sets the state of the control line connecting the host to the device.
	USB_HOST_CDC_EventHandlerSet	This class registers a event handler for the specified CDC class driver instance.
	USB_HOST_CDC_LineCodingGet	This function requests line coding from the attached CDC device.
	USB_HOST_CDC_LineCodingSet	This function sets the line coding of the attached CDC device.
	USB_HOST_CDC_Read	This function reads data from the attached CDC device.
	USB_HOST_CDC_SerialStateNotificationGet	This function requests serial state noitification data from the attached CDC device.
	USB_HOST_CDC_Write	This function writes data to the attached CDC device.

Macros

	Name	Description
	<u>USB_HOST_CDC_EVENT_RESPONSE_NONE</u>	USB Host CDC Class Driver Event Handler Response Type None.
	<u>USB_HOST_CDC_TRANSFER_HANDLE_INVALID</u>	USB Host CDC Class Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	<u>USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE</u>	USB Host CDC Class Driver Event Data.
	<u>USB_HOST_CDC_EVENT_DATA_READ_COMPLETE</u>	USB Host CDC Class Driver Event Data.
	<u>USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE</u>	USB Host CDC Class Driver Event Data.
	<u>USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED</u>	USB Host CDC Class Driver Event Data.
	<u>USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE</u>	USB Host CDC Class Driver Event Data.
	<u>USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE</u>	USB Host CDC Class Driver Event Data.
	<u>USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE</u>	USB Host CDC Class Driver Event Data.

Types

	Name	Description
	<u>USB_HOST_CDC_CONTROL_LINE_STATE</u>	USB Host Control Line State Data Type.
	<u>USB_HOST_CDC_EVENT_HANDLER</u>	USB Host CDC Class Driver Event Handler Function Pointer Type.
	<u>USB_HOST_CDC_EVENT_RESPONSE</u>	USB Host CDC Class Driver Event Callback Response Type
	<u>USB_HOST_CDC_INDEX</u>	USB Host CDC Class Driver Index
	<u>USB_HOST_CDC_SERIAL_STATE_NOTIFICATION_DATA</u>	USB Host Serial State Notification Data Type.
	<u>USB_HOST_CDC_TRANSFER_HANDLE</u>	USB Host CDC Class Driver Transfer Handle

File Name

usb_host_cdc.h

Company

Microchip Technology Incorporated

6 Third-Party Library Help

6.1 Third-Party Library Overview

6.1.1 Introduction

This topic provides an overview of the Third-Party Libraries in MPLAB Harmony.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

6.1.2 Release Notes

MPLAB Harmony Version: v0.70b **Release Date:** 18Nov2013

New This Release:

Nothing to report in this release.

Known Issues:

Nothing to report in this release.

6.1.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

7 Board Support Packages Help

7.1 Board Support Packages Overview

7.1.1 Introduction

This topic provides information for the Board Support Package (BSP) in MPLAB Harmony.

Description

A Board Support Package (BSP) provides code and configuration items necessary to support board-specific hardware. A BSP may be provided for Microchip development or demo boards or may be defined by customers for their own boards (which is recommended to make it easy to support multiple boards in the future, even if only a single board exists when a project is first created).

A BSP may contain a board-specific configuration header ("[bsp_config.h](#)" and possibly others), a board-specific system initialization file ("[bsp_sys_init.c](#)"), a file containing board-specific ISR implementations ("[bsp_sys_int.c](#)"), a board-specific system "tasks" routine (implemented in a "[bsp_sys_tasks.c](#)" file), and even board-specific driver implementations (each with its own directory).

Everything that is contained within a BSP can be either used by or replaced by application-specific items if desired. For example, the application may configure the system initialization routine ("[SYS_Initialize](#)") to directly call a board-specific initialization routine (called "[BSP_Initialize](#)") or it can use the BSP-specific initialization routine as an example from which to start developing application-specific board initialization code. Normally, Microchip demo applications will use the BSP code directly to avoid duplication of board-specific code.

7.1.2 Configuration

This topic describes the BSP configuration header options.

Description

A BSP configuration header defines configuration options that are "fixed" by the board. That is, any option that cannot be changed unless the hardware on the board is changed may be defined in a BSP's "[bsp_config.h](#)" file (or in a file included by it). This will normally include board parameters such as the oscillator frequency for a fixed-frequency oscillator or convenient names by which board-specific features may be identified (such as which pins [connect](#) to switches or LEDs). It may also include options for libraries and drivers that can be used by an application running on that board that are "fixed" by the hardware on the board. However, it may not define all options necessary for a library, only the board-specific options. Application-specific headers may include BSP-specific headers to define the board-hardware-specific options or it may define them itself, thus overriding a BSP's options.

7.1.3 Board Drivers

This topic describes the board drivers of the BSP.

Description

Some peripheral hardware may not be directly built into the microcontroller. For example, devices such as external CODECs, EEPROMs, etc, may be built onto a board and will not be available as peripherals in the microcontroller. Devices such as these still need device drivers and these drivers still need to follow the MPLAB harmony driver architecture. However, they will not normally be provided as part of the standard MPLAB Harmony driver set for Microchip microcontrollers. Instead, all files related to drivers for devices mounted on Microchip demo and development boards will be contained within the "bsp/drivers" directory. They will otherwise, follow the same rules as all other MPLAB Harmony device drivers.

7.1.4 System Initialization

This topic describes BSP system initialization.

Description

Normally, an application will define or configure the "[SYS_Initialize](#)" routine in any way that is necessary to initialize the system as desired. However, board-specific initialization may be necessary. When it is, a BSP must implement a board-specific initialization routine called "[BSP_Initialize](#)" that performs any necessary low-level board initialization necessary to support normal system operation. The "[BSP_Initialize](#)" routine may have an associated "initialization" structure, a pointer to which may be passed in as a parameter, to define any application-specific parameters necessary to initialize any low-level board hardware. Low-level board hardware may be things like power subsystems, bus enable signals, or other items necessary for basic board operations. It should not include the initialization of board-specific drivers as that is done separately. However, a BSP may define initialization data used to initialize board-specific drivers if that data is "fixed" by the board.

7.1.5 ISR Implementation

This topic describes BSP ISR implementation.

Description

Like the [SYS_Initialize](#) routine, an ISR may be implemented by an application in any way necessary to support the desired system behavior. Unlike the initialization support, a BSP-specific ISR implementation should never be called by the application-specific ISR implementation. It must either be used exactly as defined by the BSP or the application's system interrupt support must define its own ISR implementation. If the application defines its own implementation, the BSP ISR can be used as an example.

7.2 Explorer 16 PIC32MX795F512L BSP Library

7.2.1 Introduction

Explorer 16 PIC32MX795F512L BSP Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the Explorer 16 PIC32MX795F512L BSP Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

7.2.2 Release Notes

MPLAB Harmony Version: v0.70b **Explorer 16 PIC32MX795F512L BSP Library Version :** 0.51.03b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

7.2.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS

BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

7.2.4 Using the Library

This topic describes the basic architecture of the Explorer 16 PIC32MX795F512L BSP Library and provides information and examples on how to use it.

Interface Header File: [bsp_config.h](#)

The interface to the Explorer 16 PIC32MX795F512L BSP library is defined in the "[bsp_config.h](#)" header file. Any C language source (.c) file that uses the Explorer 16 PIC32MX795F512L BSP library should include "[bsp_config.h](#)".

7.2.4.1 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the Explorer 16 PIC32MX795F512L BSP Library module.

Library Interface Section	Description
Initialization Functions	This section provides initialization functions.
LED Control Functions	This section provides functions for LED Control.
Other Functions	This section provides additional functions for this API.
Data Types and Constants	This section provides various definitions describing this API.

7.3 PIC32 USB Starter Kit II BSP Library

7.3.1 Introduction

PIC32 USB Starter Kit II BSP Library

for

Microchip Microcontrollers

This library provides a low-level abstraction of the PIC32 USB Starter Kit II Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

7.3.2 Release Notes

MPLAB Harmony Version: v0.70b **PIC32 USB Starter Kit II BSP Library Version :** 0.51.03b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

7.3.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR

CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

7.3.4 Using the Library

This topic describes the basic architecture of the PIC32 USB Starter Kit II BSP Library and provides information and examples on how to use it.

Interface Header File: [bsp_config.h](#)

The interface to the PIC32 USB Starter Kit II BSP Library is defined in the "[bsp_config.h](#)" header file. Any C language source (.c) file that uses the PIC32 USB Starter Kit II BSP Library should include "[bsp_config.h](#)".

7.3.4.1 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the PIC32 USB Starter Kit II Library module.

Library Interface Section	Description
Initialization Functions	This section provides initialization functions.
LED Control Functions	This section provides functions for LED Control.
Other Functions	This section provides additional functions for this API.
Data Types and Constants	This section provides various definitions describing this API.

7.4 PIC32MX USB Audio BSP Library

7.4.1 Introduction

PIC32MX USB Audio BSP Library
for
Microchip Microcontrollers

This library provides a low-level abstraction of the PIC32MX USB Audio BSP Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

Note: This information was not available at time of release and will be provided in a future release of MPLAB Harmony.

7.4.2 Release Notes

MPLAB Harmony Version: v0.70b **PIC32MX USB Audio BSP Library Version :** 0.51.03b **Release Date:** 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

7.4.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR

CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

7.4.4 Using the Library

This topic describes the basic architecture of the PIC32MX USB Audio BSP Library and provides information and examples on how to use it.

Interface Header File: [bsp_config.h](#)

The interface to the PIC32MX USB Audio BSP Library is defined in the "[bsp_config.h](#)" header file. Any C language source (.c) file that uses the PIC32MX USB Audio BSP Library should include "[bsp_config.h](#)".

7.4.4.1 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the PIC32MX USB Audio BSP Library module.

Library Interface Section	Description
Initialization Functions	This section provides initialization functions.
LED Control Functions	This section provides functions for LED Control.
Other Functions	This section provides additional functions for this API.
Data Types and Constants	This section provides various definitions describing this API.

7.5 PIC32MZ Embedded Connectivity (EC) Starter Kit BSP Library

7.5.1 Introduction

**PIC32MZ Embedded Connectivity (EC) Starter Kit BSP Library
for
Microchip Microcontrollers**

This library provides a low-level abstraction of the PIC32MZ Embedded Connectivity (EC) Starter Kit BSP Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

Note: This information was not available at time of release and will be added in a future release of MPLAB Harmony.

7.5.2 Release Notes

MPLAB Harmony Version: v0.70b **PIC32MZ Embedded Connectivity (EC) Starter Kit BSP Library Version :** 0.51.03b
Release Date: 18Nov2013

New This Release:

This is the first release of the library. The interface can change in the beta and/or 1.0 release.

Known Issues:

Nothing to report in this release.

7.5.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

7.5.4 Using the Library

This topic describes the basic architecture of the PIC32MZ Embedded Connectivity (EC) Starter Kit BSP Library and provides information and examples on how to use it.

Interface Header File: [bsp_config.h](#)

The interface to the PIC32MZ Embedded Connectivity (EC) Starter Kit BSP Library is defined in the "[bsp_config.h](#)" header file. Any C language source (.c) file that uses the PIC32MZ Embedded Connectivity (EC) Starter Kit BSP Library should include "[bsp_config.h](#)".

7.5.4.1 Library Overview

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the PIC32MZ Embedded Connectivity (EC) Starter Kit BSP Library module.

Library Interface Section	Description
Initialization Functions	This section provides initialization functions.
LED Control Functions	This section provides functions for LED Control.
Other Functions	This section provides additional functions for this API.
Data Types and Constants	This section provides various definitions describing this API.

7.6 Library Interface

7.6.1 Initialization Functions

Functions

	Name	Description
	BSP_Initialize	Performs the necessary actions to initialize a board

Description

7.6.1.1 BSP_Initialize Function

C

```
void BSP_Initialize();
```

Description

This routine performs the neccessary actions to initialize a board

Preconditions

None

Returns

None

7.6.2 LED Control Functions

Functions

	Name	Description
	BSP_SwitchOFFLED	Turns OFF specified LED on the board.
	BSP_SwitchONLED	Turns ON specified LED on the board.
	BSP_ToggleLED	Toggles specified LED on the board.
	BSP_SwitchToggleLED	Toggles the onboard LED.

Description

7.6.2.1 BSP_SwitchOFFLED Function

C

```
void BSP_SwitchOFFLED(  
    BSP_LED led  
);
```

Description

This function turns OFF specified LED on the board.

Preconditions

BSP should be initialized by calling void [BSP_Initialize](#)(void) function.

Parameters

Parameters	Description
led	LED ID as specified by BSP_LED enum.

Returns

None

7.6.2.2 BSP_SwitchONLED Function

C

```
void BSP_SwitchONLED(  
    BSP_LED led  
);
```

Description

This function turns ON specified LED on the board.

Preconditions

BSP should be initialized by calling void [BSP_Initialize](#)(void) function.

Parameters

Parameters	Description
led	LED ID as specified by BSP_LED enum.

Returns

None

7.6.2.3 BSP_ToggleLED Function

C

```
void BSP_ToggleLED(  
    BSP_LED led  
);
```

Description

This function Toggles specified LED on the board.

Preconditions

BSP should be initialized by calling void [BSP_Initialize](#)(void) function.

Parameters

Parameters	Description
led	LED ID as specified by BSP_LED enum.

Returns

None

7.6.2.4 BSP_SwitchToggleLED Function

C

```
void BSP_SwitchToggleLED(  
    BSP_LED led  
);
```

Description

This routine toggles the onboard LED.

7.6.3 Other Functions

Functions

	Name	Description
	SYSTEMConfigPerformance	Configures the system cache and flash wait states for maximum performance.
	BSP_ReadCoreTimer	Returns the current core timer value.
	BSP_ReadSwitch	Reads the switch state.

Description

7.6.3.1 SYSTEMConfigPerformance Function

C

```
unsigned int SYSTEMConfigPerformance(  
    unsigned int sys_clock  
);
```

Description

This function configures the system cache and flash wait states for maximum performance.

Preconditions

None

Parameters

Parameters	Description
sys_clock	Value of system clock

7.6.3.2 BSP_ReadCoreTimer Function

C

```
uint32_t BSP_ReadCoreTimer();
```

Description

Returns the current core timer value.

Preconditions

None

7.6.3.3 BSP_ReadSwitch Function

C

```
BSP_SWITCH_STATE BSP_ReadSwitch(  
    BSP_SWITCH bspSwitch  
);
```

Description

Reads the switch state.

Preconditions

None

Parameters

Parameters	Description
bspSwitch	Switch ID as specified by BSP_SWITCH enum.

Return Values

Return Values	Description
BSP_SWITCH_STATE_PRESSED	Switch is pressed.
BSP_SWITCH_STATE_RELEASED	Switch is not pressed.

7.6.4 Data Types and Constants

Enumerations

	Name	Description
	BSP_LED	Holds LED numbers.
	BSP_SWITCH	Holds Switch numbers.
	BSP_SWITCH_STATE	Holds Switch status.

Macros

	Name	Description
	<u>BSP_ANALOG_PIN</u>	Defines the constant which identifies analog pin
	<u>BSP_DIGITAL_PIN</u>	Defines the constant which identifies digital pin
	<u>BSP_INPUT</u>	Defines the constant which identifies input
	<u>BSP_OUTPUT</u>	Defines the constant which identifies output

Description

7.6.4.1 BSP_ANALOG_PIN Macro

C

```
#define BSP_ANALOG_PIN PORTS_PIN_MODE_ANALOG
```

Description

analog Pin Constant

This constant identifies analog pin

7.6.4.2 BSP_DIGITAL_PIN Macro

C

```
#define BSP_DIGITAL_PIN PORTS_PIN_MODE_DIGITAL
```

Description

Digital Pin Constant

This constant identifies digital pin

7.6.4.3 BSP_INPUT Macro

C

```
#define BSP_INPUT 1
```

Description

Input Constant

This constant identifies input

7.6.4.4 BSP_OUTPUT Macro

C

```
#define BSP_OUTPUT 0
```

Description

Output Constant

This constant identifies output

7.6.4.5 BSP_LED Enumeration

C

```
typedef enum {
    LED_3 = PORTS_BIT_POS_0,
    LED_4 = PORTS_BIT_POS_1,
    LED_5 = PORTS_BIT_POS_2,
    LED_6 = PORTS_BIT_POS_3,
    LED_7 = PORTS_BIT_POS_4,
    LED_8 = PORTS_BIT_POS_5,
    LED_9 = PORTS_BIT_POS_6,
    LED_10 = PORTS_BIT_POS_7
} BSP_LED;
```

Description

LED Number.

This enumeration defines the LED numbers.

Members

Members	Description
LED_3 = PORTS_BIT_POS_0	LED 3
LED_4 = PORTS_BIT_POS_1	LED 4
LED_5 = PORTS_BIT_POS_2	LED 5
LED_6 = PORTS_BIT_POS_3	LED 6
LED_7 = PORTS_BIT_POS_4	LED 7
LED_8 = PORTS_BIT_POS_5	LED 8
LED_9 = PORTS_BIT_POS_6	LED 9
LED_10 = PORTS_BIT_POS_7	LED 10

Remarks

None.

7.6.4.6 BSP_SWITCH Enumeration

C

```
typedef enum {
    SWITCH_3 = PORTS_BIT_POS_6,
    SWITCH_4 = PORTS_BIT_POS_13,
    SWITCH_5 = PORTS_BIT_POS_7,
    SWITCH_6 = PORTS_BIT_POS_7
} BSP_SWITCH;
```

Description

BSP Switch.

This enumeration defines the Switch numbers.

Members

Members	Description
SWITCH_3 = PORTS_BIT_POS_6	SWITCH 3
SWITCH_4 = PORTS_BIT_POS_13	SWITCH 4
SWITCH_5 = PORTS_BIT_POS_7	SWITCH 5
SWITCH_6 = PORTS_BIT_POS_7	SWITCH 6

Remarks

None.

7.6.4.7 BSP_SWITCH_STATE Enumeration

C

```
typedef enum {  
    BSP_SWITCH_STATE_PRESSED = 0,  
    BSP_SWITCH_STATE_RELEASED = 1  
} BSP_SWITCH_STATE;
```

Description

BSP Switch state.

This enumeration defines the switch state.

Members

Members	Description
BSP_SWITCH_STATE_PRESSED = 0	Switch pressed
BSP_SWITCH_STATE_RELEASED = 1	Switch not pressed

Remarks

None.

7.7 Files

Files

Name	Description
bsp_config.h	Board support configuration file.

Description

7.7.1 bsp_config.h

Board support configuration file.

This contains all the configuration that is required to be done for the application running on PIC32 USB starter kit

Enumerations

	Name	Description
	BSP_LED	Holds LED numbers.
	BSP_SWITCH	Holds Switch numbers.
	BSP_SWITCH_STATE	Holds Switch status.

Functions

	Name	Description
	BSP_Initialize	Performs the necessary actions to initialize a board
	BSP_ReadCoreTimer	Returns the current core timer value.
	BSP_ReadSwitch	Reads the switch state.
	BSP_SwitchOFFLED	Turns OFF specified LED on the board.
	BSP_SwitchONLED	Turns ON specified LED on the board.
	BSP_SwitchToggleLED	Toggles the onboard LED.
	BSP_ToggleLED	Toggles specified LED on the board.
	SYSTEMConfigPerformance	Configures the system cache and flash wait states for maximum performance.

Macros

	Name	Description
	BSP_ANALOG_PIN	Defines the constant which identifies analog pin
	BSP_DIGITAL_PIN	Defines the constant which identifies digital pin
	BSP_INPUT	Defines the constant which identifies input
	BSP_OUTPUT	Defines the constant which identifies output

File Name

bsp_config.h

Company

Microchip Technology Incorporated

8 Symbol Reference

8.1 Macros

Macros

	Name	Description
	DRV_ETHMAC_INDEX_0	Ethernet driver index definitions.
	DRV_ETHMAC_INDEX_COUNT	Number of valid Ethernet driver indices.

8.1.1 DRV_ETHMAC_INDEX_0 Macro

C

```
#define DRV_ETHMAC_INDEX_0 0
```

Description

Ethernet Driver Module Index Numbers

These constants provide Ethernet driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_ETHMAC_Initialize and DRV_ETHMAC_Open routines to identify the driver instance in use.

8.1.2 DRV_ETHMAC_INDEX_COUNT Macro

C

```
#define DRV_ETHMAC_INDEX_COUNT ETH_NUMBER_OF_MODULES
```

Description

Ethernet Driver Module Index Count

This constant identifies number of valid Ethernet driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

Index

-
- `_LIBQ_Q15_cos_Q2_13` function 5-1287
 - `_LIBQ_Q15_sin_Q2_13` function 5-1286
 - `_LIBQ_Q15FromFloat` function 5-1298
 - `_LIBQ_Q15FromString` function 5-1301
 - `_LIBQ_Q15Rand` function 5-1297
 - `_LIBQ_Q16_tan_Q2_29` function 5-1288
 - `_LIBQ_Q16Div` function 5-1281
 - `_LIBQ_Q16Exp` function 5-1285
 - `_LIBQ_Q16Power` function 5-1284
 - `_LIBQ_Q16Sqrt` function 5-1281
 - `_LIBQ_Q2_13_acos_Q15` function 5-1292
 - `_LIBQ_Q2_13_asin_Q15` function 5-1289
 - `_LIBQ_Q2_13_atan_Q7_8` function 5-1294
 - `_LIBQ_Q2_13_atan2_Q7_8` function 5-1295
 - `_LIBQ_Q2_29_acos_Q31` function 5-1292
 - `_LIBQ_Q2_29_acos_Q31_Fast` function 5-1293
 - `_LIBQ_Q2_29_asin_Q31` function 5-1290
 - `_LIBQ_Q2_29_asin_Q31_Fast` function 5-1291
 - `_LIBQ_Q2_29_atan_Q16` function 5-1294
 - `_LIBQ_Q2_29_atan2_Q16` function 5-1296
 - `_LIBQ_Q3_12_log10_Q16` function 5-1282
 - `_LIBQ_Q31_cos_Q2_29` function 5-1287
 - `_LIBQ_Q31_sin_Q2_29` function 5-1286
 - `_LIBQ_Q31FromFloat` function 5-1299
 - `_LIBQ_Q31Rand` function 5-1297
 - `_LIBQ_Q4_11_ln_Q16` function 5-1283
 - `_LIBQ_Q5_10_log2_Q16` function 5-1283
 - `_LIBQ_Q7_8_tan_Q2_13` function 5-1289
 - `_LIBQ_ToFloatQ15` function 5-1300
 - `_LIBQ_ToFloatQ31` function 5-1300
 - `_LIBQ_ToStringQ15` function 5-1302
 - `_PARM_EQUAL_FILTER` structure 5-1265
 - `_PARM_EQUAL_FILTER_16` structure 5-1266
 - `_PARM_EQUAL_FILTER_32` structure 5-1266
 - `_Q0_15` type 5-1305
 - `_Q0_31` type 5-1305
 - `_Q15` type 5-1305
 - `_Q15_16` type 5-1306
 - `_Q15_MAX` macro 5-1302
 - `_Q15_MIN` macro 5-1303
 - `_Q16` type 5-1306
 - `_Q16_MAX` macro 5-1303
 - `_Q16_MIN` macro 5-1303
 - `_Q2_13` type 5-1306
 - `_Q2_13_MAX` macro 5-1303
 - `_Q2_13_MIN` macro 5-1303
 - `_Q2_29` type 5-1306
 - `_Q2_29_MAX` macro 5-1303
 - `_Q2_29_MIN` macro 5-1303
 - `_Q3_12` type 5-1306
 - `_Q3_12_MAX` macro 5-1304
 - `_Q3_12_MIN` macro 5-1304
 - `_Q31` type 5-1306
 - `_Q31_MAX` macro 5-1304
 - `_Q31_MIN` macro 5-1304
 - `_Q4_11` type 5-1307
 - `_Q4_11_MAX` macro 5-1304
 - `_Q4_11_MIN` macro 5-1304
 - `_Q5_10` type 5-1307
 - `_Q5_10_MAX` macro 5-1305
 - `_Q5_10_MIN` macro 5-1305
 - `_Q7_8` type 5-1307
 - `_Q7_8_MAX` macro 5-1305
 - `_Q7_8_MIN` macro 5-1305
 - `_SYS_MSG_H` macro 5-3462
 - `_SYS_MSG_OBJECT` macro 5-3462
 - `_SYS_TASKS_PRIORITIES` enumeration 5-3341

2

2. Creating Your Own Applications 1-21

3

3. Release Notes & Information 1-23

4

4. Previous Releases 1-32

5

5. Tips and Tricks 1-36

6

6. Glossary 1-37

A

About CAN Protocol 5-1555

Abstract Control Model (ACM) 5-4063

Abstraction Model 5-195, 5-228, 5-255, 5-290, 5-300, 5-310, 5-320, 5-334, 5-344, 5-368, 5-398, 5-419, 5-458, 5-513, 5-553, 5-607, 5-695, 5-1112, 5-1129, 5-1313, 5-3349, 5-3364, 5-3420, 5-3442, 5-3466, 5-3502, 5-3531, 5-3570, 5-3576, 5-3599, 5-3615, 5-3626, 5-3637, 5-3647, 5-3655, 5-3660, 5-3691, 5-3699, 5-3710, 5-3719, 5-3749, 5-3772, 5-3795, 5-3800, 5-3807, 5-3810, 5-3823, 5-3859, 5-3869, 5-3882, 5-3914, 5-3918, 5-3931, 5-3956, 5-4005, 5-4061, 5-4103, 5-4169

accept function 5-3602

Accessing the Result Buffers 5-1373

ADC Driver Library 5-193

ADC Peripheral Library 5-1361

ADC_ACQUISITION_TIME type 5-1438

ADC_BUFFER_MODE enumeration 5-1426

ADC_CHANNEL_GROUP enumeration 5-1423

ADC_CHANNEL123_INPUTS_NEG enumeration 5-1432

ADC_CHANNEL123_INPUTS_POS enumeration 5-1433

ADC_CLOCK_SOURCE enumeration 5-1424

ADC_CONVERSION_CLOCK type 5-1438

ADC_CONVERSION_ORDER enumeration 5-1435

ADC_CONVERSION_TRIGGER_SOURCE enumeration 5-1424

ADC_DMA_ADDRESS_INCREMENT enumeration 5-1435

ADC_DMA_BUFFER_MODE enumeration 5-1431

ADC_DMA_INPUT_BUFFER enumeration 5-1437

ADC_INPUTS_NEGATIVE enumeration 5-1421

ADC_INPUTS_POSITIVE enumeration 5-1433

ADC_INPUTS_SCAN enumeration 5-1428

ADC_MODULE_ID enumeration 5-1420

ADC_MUX enumeration 5-1427

ADC_PAIR enumeration 5-1431

ADC_REFERENCE_INPUT enumeration 5-1430

ADC_RESULT_BUF_STATUS enumeration 5-1427

ADC_RESULT_FORMAT enumeration 5-1430

ADC_RESULT_SIZE enumeration 5-1426

ADC_SAMPLE type 5-1439

ADC_SAMPLES_PER_INTERRUPT enumeration 5-1422

ADC_SAMPLING_MODE enumeration 5-1427

ADC_VOLTAGE_REFERENCE enumeration 5-1420

ADCP Peripheral Library 5-1471

ADCP_CLASS12_INPUT_ID enumeration 5-1507

ADCP_CLOCK_SOURCE enumeration 5-1507

ADCP_DCOMP_ID enumeration 5-1513

ADCP_DSH_ID enumeration 5-1509

ADCP_INPUT_ID enumeration 5-1510

ADCP_MODULE_ID enumeration 5-1506

ADCP_ODFLTR_ID enumeration 5-1513

ADCP_ODFLTR_OSR enumeration 5-1514

ADCP_SCAN_TRG_SRC enumeration 5-1514

ADCP_SH_ID enumeration 5-1508

ADCP_SH_MODE enumeration 5-1509

ADCP_TRG_SRC enumeration 5-1515

ADCP_VREF_SOURCE enumeration 5-1507

adhocMode enumeration 5-655

Advanced Data Transfer 5-517

Advanced Font Features 5-1119

Advanced MPFS2 Settings 5-3544

AF_INET macro 5-3609

Alarm Config 5-479

Alarm Functionality 5-466

Alarm Functionality Examples 5-481

Alarm Mode Operations 5-2710

Alpha Blending 5-1116, 5-1123

ALPHA_DELTA macro 5-294

AN_READY type 5-1517

AN_SELECT union 5-1516

Announce TCP/IP Stack Library 5-3569

ANNOUNCE_MAX_PAYLOAD macro 5-3571
 ANNOUNCE_PORT macro 5-3571
 Application Interaction 5-3380
 Applications Help 3-61
 Applications Overview 3-62
 Architecture 5-3378
 Architecture Overview 5-4132
 Arcs 5-1114
 ARP Changes 5-3551
 ARP TCP/IP Stack Library 5-3574
 arp.h 5-3596
 arp_app_callbacks structure 5-3595
 ARP_CACHE_ENTRIES_97J60 macro 5-3577
 ARP_CACHE_ENTRIES_DEFAULT macro 5-3578
 ARP_CACHE_ENTRIES_ENCJ60 macro 5-3578
 ARP_CACHE_ENTRIES_ENCJ600 macro 5-3578
 ARP_CACHE_ENTRIES_MRF24W macro 5-3578
 ARP_CACHE_ENTRIES_PIC32INT macro 5-3578
 ARP_CACHE_PENDING_ENTRY_TMO macro 5-3578
 ARP_CACHE_PENDING_RETRY_TMO macro 5-3579
 ARP_CACHE_PERMANENT_QUOTA macro 5-3579
 ARP_CACHE_PURGE_QUANTA macro 5-3579
 ARP_CACHE_PURGE_THRESHOLD macro 5-3579
 ARP_CACHE_SOLVED_ENTRY_TMO macro 5-3579
 arp_config.h 5-3597
 arp_manager.h 5-3597
 ARP_MODULE_CONFIG structure 5-3595
 ARP_TASK_PROCESS_RATE macro 5-3579
 Assigning Buffer Memory 5-1558
 Asynchronous Counter 5-3017
 Asynchronous USART Mode 5-3090
 Audio Protocol Interface Mode 5-2817
 audio_speaker 3-166
 AUTH_LOCALIZED_PASSWORD_KEY_LEN macro 5-3826

B

Before You Start 1-2
 berekely_api_config.h 5-3613
 Berkeley TCP/IP Stack Library 5-3598

berkeley_api.h 5-3613
 BERKELEY_MODULE_CONFIG structure 5-3610
 Bevels 5-1115
 bind function 5-3603
 biquad16 structure 5-1264
 BLACK macro 5-941
 Blocking Applications 3-99
 BLUE macro 5-941
 BMX Peripheral Library 5-1519
 BMX_MODULE_ID enumeration 5-1543
 Board Drivers 7-4229
 Board Support Packages Help 7-4227
 Board Support Packages Overview 7-4228
 BRIGHTBLUE macro 5-941
 BRIGHTCYAN macro 5-942
 BRIGHTGREEN macro 5-942
 BRIGHTMAGENTA macro 5-942
 BRIGHTRED macro 5-942
 BRIGHTYELLOW macro 5-942
 BROWN macro 5-942
 BSP_ANALOG_PIN macro 7-4242
 bsp_config.h 7-4245
 BSP_DIGITAL_PIN macro 7-4242
 BSP_Initialize function 7-4238
 BSP_INPUT macro 7-4242
 BSP_LED enumeration 7-4243
 BSP_OUTPUT macro 7-4242
 BSP_ReadCoreTimer function 7-4241
 BSP_ReadSwitch function 7-4241
 BSP_SWITCH enumeration 7-4243
 BSP_SWITCH_STATE enumeration 7-4244
 BSP_SwitchOFFLED function 7-4239
 BSP_SwitchONLED function 7-4239
 BSP_SwitchToggleLED function 7-4240
 BSP_ToggleLED function 7-4239
 Building MPFS2 Images 5-3542
 Building Projects 4-178
 Building Projects Help 4-177
 Building the Application 3-109, 3-126, 3-145

Building the Library 5-202, 5-236, 5-262, 5-291, 5-301, 5-311, 5-326, 5-335, 5-352, 5-377, 5-400, 5-428, 5-482, 5-519, 5-574, 5-609, 5-701, 5-1124, 5-1130, 5-1324, 5-2256, 5-3368, 5-3444, 5-3474, 5-3517, 5-3572, 5-3580, 5-3601, 5-3618, 5-3630, 5-3640, 5-3648, 5-3657, 5-3692, 5-3700, 5-3711, 5-3723, 5-3753, 5-3773, 5-3796, 5-3804, 5-3808, 5-3814, 5-3832, 5-3862, 5-3875, 5-3888, 5-3916, 5-3921, 5-3935, 5-3957

Building the Library 5-3532

Bulk and Interrupt Transfers 5-570

BURLYWOOD macro 5-942

Bus Arbiter 5-1522

C

Cache Control 5-1522

Cache Control Operations 5-2431

Cache Line Operations 5-2432

Cache Status Operations 5-2433

Can Bit Time Quanta 5-1556

CAN Peripheral Library 5-1553

CAN_CHANNEL enumeration 5-1620

CAN_CHANNEL_EVENT enumeration 5-1621

CAN_CHANNEL_MASK enumeration 5-1622

CAN_DNET_FILTER_SIZE enumeration 5-1623

CAN_ERROR_STATE enumeration 5-1624

CAN_FILTER enumeration 5-1625

CAN_FILTER_MASK enumeration 5-1627

CAN_FILTER_MASK_TYPE enumeration 5-1627

CAN_FUNCTIONAL_MODES enumeration 5-1628

CAN_ID_TYPE enumeration 5-1628

CAN_MODULE_EVENT enumeration 5-1629

CAN_MODULE_ID enumeration 5-1629

CAN_MSG_EID structure 5-1630

CAN_OPERATION_MODES enumeration 5-1630

CAN_RECEIVE_CHANNEL enumeration 5-1631

CAN_RECEIVE_MODES enumeration 5-1631

CAN_RX_DATA_MODE enumeration 5-1632

CAN_RX_DATA_ONLY_SIZE_BYTES macro 5-1636

CAN_RX_MSG_BUFFER union 5-1632

CAN_RX_MSG_SID structure 5-1633

CAN_TIME_SEGMENT_LENGTH enumeration 5-1633

CAN_TX_CHANNEL_STATUS enumeration 5-1634

CAN_TX_MSG_BUFFER union 5-1634

CAN_TX_MSG_SID structure 5-1635

CAN_TX_RTR enumeration 5-1635

CAN_TX_RX_MESSAGE_SIZE_BYTES macro 5-1636

CAN_TXCHANNEL_PRIORITY enumeration 5-1635

cdc_basic 3-167

cdc_com_port_dual 3-154

cdc_com_port_single 3-152

cdc_serial_emulator 3-156

CH_CLR0 macro 5-943

CH_CLR1 macro 5-943

CH_CLR10 macro 5-943

CH_CLR11 macro 5-943

CH_CLR12 macro 5-943

CH_CLR13 macro 5-943

CH_CLR14 macro 5-944

CH_CLR15 macro 5-944

CH_CLR2 macro 5-944

CH_CLR3 macro 5-944

CH_CLR4 macro 5-944

CH_CLR5 macro 5-944

CH_CLR6 macro 5-944

CH_CLR7 macro 5-945

CH_CLR8 macro 5-945

CH_CLR9 macro 5-945

Change Notification 5-3472

Changing the Clock 5-3351

Changing the Default Font 5-1119

Channel Configuration 5-1559

Channel Events 5-1562

Channel Management 5-1743

CHART_MARGIN macro 5-945

CHART_YGRIDCOUNT macro 5-945

CHARTPARAM structure 5-888

Client Transfer - Core 5-423

ClearPaletteChangeError function 5-827

Client Access 5-422, 5-516

Client Access Operation 5-346

Client Basic Operation 5-346

Client Block Data Operation 5-348	Configuring the Application 3-109, 3-127, 3-146
Client Config 5-480	Configuring the Comparator Interrupts 5-1671
Client Configuration 5-428	Configuring the Driver Library 5-311
Client Core Functionality 5-196	Configuring the Hardware 3-67, 3-69, 3-72, 3-74, 3-76, 3-86, 3-88, 3-89, 3-90, 3-109, 3-114, 3-115, 3-116, 3-117, 3-126, 3-131, 3-133, 3-135, 3-145, 3-153, 3-154, 3-156, 3-158, 3-160, 3-161, 3-162, 3-164, 3-165, 3-167, 3-168, 3-169
Client Data Handling 5-4111	Configuring the Libraries 3-110, 3-127, 3-146
Client Functionality 5-608	Configuring the Library 5-202, 5-234, 5-259, 5-290, 5-300, 5-321, 5-335, 5-349, 5-376, 5-400, 5-428, 5-474, 5-519, 5-574, 5-608, 5-701, 5-1123, 5-1130, 5-1322, 5-1380, 5-1480, 5-1526, 5-1566, 5-1748, 5-1866, 5-2089, 5-2175, 5-2205, 5-2256, 5-2294, 5-2336, 5-2434, 5-2488, 5-2601, 5-2666, 5-2690, 5-2712, 5-2766, 5-2823, 5-2903, 5-3021, 5-3101, 5-3182, 5-3322, 5-3352, 5-3366, 5-3424, 5-3443, 5-3474, 5-3516, 5-3532, 5-3571, 5-3577, 5-3600, 5-3616, 5-3627, 5-3638, 5-3648, 5-3656, 5-3661, 5-3691, 5-3700, 5-3710, 5-3720, 5-3749, 5-3773, 5-3796, 5-3801, 5-3807, 5-3814, 5-3824, 5-3860, 5-3872, 5-3884, 5-3914, 5-3920, 5-3933, 5-3957, 5-3985
Client Interaction 5-462	Configuring the Stack 5-3974
Client Operation 5-373	connect function 5-3604
Client Setup 5-4105	Constants 5-190
Client Subclass Specific 5-4120	Control Transfers 5-565
Client Transfer - Advanced 5-425	Controlling the Conversion Process 5-1371
CLK_PERIPH_BUS_MZ enumeration 5-3361	Controlling the Sampling Process 5-1370
CLK_SOURCE_PERIPHERAL enumeration 5-3361	Conversion Sequence Examples 5-1375
Clock Sources 5-2334	ConvertColor25 macro 5-945
Clock System Service Library 5-3347	ConvertColor50 macro 5-946
closesocket function 5-3603	ConvertColor75 macro 5-946
Closing a File 5-3395	CopyPageWindow macro 5-946
CMP_CLOCK_DIVIDE enumeration 5-1716	Core Functionality 5-465, 5-515, 5-1112, 5-1315, 5-2896, 5-3366, 5-3883, 5-3932, 5-3956
CMP_CVREF_BANDGAP_SELECT enumeration 5-1716	Counter Modification 5-465
CMP_CVREF_REFERENCE_SELECT enumeration 5-1717	Creating Your Own USB Device 5-3970
CMP_CVREF_VALUE enumeration 5-1717	Critical Section Operation 5-1320
CMP_CVREF_VOLTAGE_SOURCE enumeration 5-1718	Critical Sections 5-3422
CMP_FILTER_CLOCK enumeration 5-1718	CRYPT_AES_CBC_Decrypt function 5-1156
CMP_INTERRUPT_EVENT enumeration 5-1719	CRYPT_AES_CBC_Encrypt function 5-1157
CMP_INVERTING_INPUT enumeration 5-1719	CRYPT_AES_CTR_Encrypt function 5-1157
CMP_MASK_A enumeration 5-1719	CRYPT_AES_CTX structure 5-1161
CMP_MASK_B enumeration 5-1720	CRYPT_AES_DIRECT_Decrypt function 5-1158
CMP_MASK_C enumeration 5-1721	CRYPT_AES_DIRECT_Encrypt function 5-1158
CMP_MODULE_ID enumeration 5-1721	CRYPT_AES_IvSet function 5-1158
CMP_NON_INVERTING_INPUT enumeration 5-1722	CRYPT_AES_KeySet function 5-1159
CMP_SPEED_POWER enumeration 5-1722	
Code Examples 5-197	
COLOR_DEPTH macro 5-325	
Colors 5-1123	
Communication Mode 5-2822	
Comparator Peripheral Library 5-1667	
Comparison of API Names 5-3386	
Compression Library 4-181	
Configuration 7-4228	

CRYPT_ECC_CTX structure 5-1162
 CRYPT_ECC_DHE_KeyMake function 5-1146
 CRYPT_ECC_DHE_SharedSecretMake function 5-1147
 CRYPT_ECC_DSA_HashSign function 5-1147
 CRYPT_ECC_DSA_HashVerify function 5-1148
 CRYPT_ECC_Free function 5-1148
 CRYPT_ECC_Initialize function 5-1148
 CRYPT_ECC_KeySizeGet function 5-1149
 CRYPT_ECC_PrivateImport function 5-1149
 CRYPT_ECC_PublicExport function 5-1150
 CRYPT_ECC_PublicImport function 5-1150
 CRYPT_ECC_SignatureSizeGet function 5-1150
 CRYPT_ERROR_StringGet function 5-1132
 CRYPT_HMAC_CTX structure 5-1162
 CRYPT_HMAC_DataAdd function 5-1135
 CRYPT_HMAC_Finalize function 5-1136
 CRYPT_HMAC_SetKey function 5-1136
 CRYPT_HUFFMAN_Compress function 5-1145
 CRYPT_HUFFMAN_DeCompress function 5-1146
 CRYPT_MD5_CTX structure 5-1162
 CRYPT_MD5_DataAdd function 5-1159
 CRYPT_MD5_Finalize function 5-1160
 CRYPT_MD5_Initialize function 5-1161
 CRYPT_RNG_BlockGenerate function 5-1143
 CRYPT_RNG_CTX structure 5-1162
 CRYPT_RNG_Get function 5-1144
 CRYPT_RNG_Initialize function 5-1144
 CRYPT_RSA_CTX structure 5-1163
 CRYPT_RSA_EncryptSizeGet function 5-1154
 CRYPT_RSA_Free function 5-1154
 CRYPT_RSA_Initialize function 5-1154
 CRYPT_RSA_PrivateDecrypt function 5-1155
 CRYPT_RSA_PrivateKeyDecode function 5-1155
 CRYPT_RSA_PublicEncrypt function 5-1156
 CRYPT_RSA_PublicKeyDecode function 5-1156
 CRYPT_SHA_CTX structure 5-1163
 CRYPT_SHA_DataAdd function 5-1141
 CRYPT_SHA_Finalize function 5-1142
 CRYPT_SHA_Initialize function 5-1142
 CRYPT_SHA256_CTX structure 5-1163
 CRYPT_SHA256_DataAdd function 5-1139
 CRYPT_SHA256_Finalize function 5-1139
 CRYPT_SHA256_Initialize function 5-1140
 CRYPT_SHA384_CTX structure 5-1163
 CRYPT_SHA384_DataAdd function 5-1137
 CRYPT_SHA384_Finalize function 5-1137
 CRYPT_SHA384_Initialize function 5-1138
 CRYPT_SHA512_CTX structure 5-1164
 CRYPT_SHA512_DataAdd function 5-1133
 CRYPT_SHA512_Finalize function 5-1134
 CRYPT_SHA512_Initialize function 5-1134
 CRYPT_TDES_CBC_Decrypt function 5-1151
 CRYPT_TDES_CBC_Encrypt function 5-1152
 CRYPT_TDES_CTX structure 5-1164
 CRYPT_TDES_IvSet function 5-1152
 CRYPT_TDES_KeySet function 5-1153
 Crypto Library 4-181
 crypto.h 5-1165
 Cryptographic (Crypto) Library 5-1127
 Customizing the Application 3-110, 3-126, 3-147
 CYAN macro 5-946

D

DARKGRAY macro 5-946
 DARKORANGE macro 5-947
 Data Transfer 5-516
 Data Types 5-188
 Data Types and Constants 5-305, 5-317, 5-340, 5-4174, 5-4209, 7-4241
 DATASERIES structure 5-889
 ddns.h 5-3653
 DDNS_CHECKIP_SERVER macro 5-3648
 ddns_config.h 5-3654
 DDNS_DEFAULT_PORT macro 5-3648
 DDNS_MODULE_CONFIG structure 5-3650
 DDNS_POINTERS structure 5-3651
 DDNS_SERVICES enumeration 5-3652
 DDNS_STATUS enumeration 5-3652

-
- Defining Colors 5-1118
 - DELAY_FIRST_PROBE_TIME macro 5-3802
 - Delays 5-3512
 - Demonstration 5-3957
 - Demonstration Application Configurations 3-151
 - Demonstration Board Information 3-77, 3-91, 3-118, 3-136, 3-170
 - Demonstrations 3-65, 3-85, 3-114, 3-131, 3-152
 - Descriptor Table Example 5-1917
 - Descriptors 5-4010
 - Developing Vendor-specific Function Drivers 5-4017
 - Development Information (Advance Information) 5-3565
 - Device 3-152
 - Device Control System Service Library 5-3363
 - Device Events 5-4015
 - DHCP Server TCP/IP Stack Library 5-3625
 - DHCP TCP/IP Stack Library 5-3614
 - dhcp.h 5-3624
 - DHCP_CLIENT_ENABLED macro 5-3617
 - DHCP_CLIENT_PORT macro 5-3617
 - dhcp_config.h 5-3624
 - DHCP_MODULE_CONFIG structure 5-3622
 - DHCP_SERVER_GATEWAY_ADDRESS macro 5-3628
 - DHCP_SERVER_IP_ADDRESS macro 5-3628
 - DHCP_SERVER_NETMASK_ADDRESS macro 5-3628
 - DHCP_SERVER_POOL_ENTRY_TYPE enumeration 5-3633
 - DHCP_SERVER_PORT macro 5-3617
 - DHCP_SERVER_PRIMARY_DNS_ADDRESS macro 5-3628
 - DHCP_SERVER_SECONDARY_DNS_ADDRESS macro 5-3628
 - DHCP_TASK_TICK_RATE macro 5-3617
 - DHCP_TIMEOUT macro 5-3617
 - dhcps.h 5-3635
 - dhcps_config.h 5-3635
 - DHCPS_IP_ADDRESS_RANGE_START macro 5-3629
 - DHCPS_LEASE_DURATION macro 5-3629
 - DHCPS_LEASE_ENTRIES_DEFAULT macro 5-3629
 - DHCPS_LEASE_ENTRY structure 5-3633
 - DHCPS_LEASE_HANDLE type 5-3634
 - DHCPS_LEASE_REMOVED_BEFORE_ACK macro 5-3629
 - DHCPS_LEASE_SOLVED_ENTRY_TMO macro 5-3629
 - DHCPS_MODULE_CONFIG structure 5-3634
 - DHCPS_TASK_PROCESS_RATE macro 5-3629
 - DisablePalette function 5-827
 - DisplayUpdatePending variable 5-887
 - DMA Mode 5-2898
 - DMA Peripheral Library Help 5-1733
 - DMA_ADDRESS_OFFSET_TYPE enumeration 5-1821
 - DMA_CHANNEL enumeration 5-1821
 - DMA_CHANNEL_ADDRESSING_MODE enumeration 5-1822
 - DMA_CHANNEL_COLLISION enumeration 5-1822
 - DMA_CHANNEL_DATA_SIZE enumeration 5-1823
 - DMA_CHANNEL_PRIORITY enumeration 5-1823
 - DMA_CHANNEL_TRANSFER_DIRECTION enumeration 5-1824
 - DMA_CHANNEL_TRIGGER_TYPE enumeration 5-1824
 - DMA_CRC_BIT_ORDER enumeration 5-1824
 - DMA_CRC_BYTE_ORDER enumeration 5-1825
 - DMA_CRC_TYPE enumeration 5-1825
 - DMA_DESTINATION_ADDRESSING_MODE enumeration 5-1826
 - DMA_INT_TYPE enumeration 5-1826
 - DMA_ISR_TASK enumeration 5-307
 - DMA_MODULE_ID enumeration 5-1827
 - DMA_PATTERN_LENGTH enumeration 5-1827
 - DMA_PING_PONG_MODE enumeration 5-1828
 - DMA_SOURCE_ADDRESSING_MODE enumeration 5-1828
 - DMA_TRANSFER_MODE enumeration 5-1829
 - DMA_TRIGGER_SOURCE enumeration 5-1829
 - DNS TCP/IP Stack Library 5-3636
 - dns.h 5-3645
 - DNS_CLIENT_ARP_TMO macro 5-3639
 - DNS_CLIENT_MODULE_CONFIG structure 5-3643
 - DNS_CLIENT_OPEN_TMO macro 5-3639
 - DNS_CLIENT_PORT macro 5-3639
 - DNS_CLIENT_SERVER_TMO macro 5-3639
 - DNS_CLIENT_TASK_PROCESS_RATE macro 5-3640
 - DNS_CLIENT_VERSION_NO macro 5-3640
 - dns_config.h 5-3645
 - DNS_RESOLVE_TYPE enumeration 5-3643
-

DNS_RESULT enumeration 5-3644	DRV_ADC_INSTANCES_NUMBER macro 5-222
DNS_SERVER_MODULE_CONFIG structure 5-3644	DRV_ADC_INTERNAL_BUFFER_SIZE macro 5-223
Double Buffering 5-1121	DRV_ADC_INTERRUPT_MODE macro 5-223
doubleBuffEnabled variable 5-888	DRV_ADC_INTERRUPT_SOURCE macro 5-223
DrawArc function 5-827	DRV_ADC_Open function 5-211
Driver Initialization 5-554	DRV_ADC_PERIPHERAL_ID macro 5-223
Driver Library Help 5-185	DRV_ADC_POWER_STATE macro 5-224
Driver Library Overview 5-185	DRV_ADC_Reinitialize function 5-207
driver.h 5-192	DRV_ADC_RESULT_FORMAT macro 5-224
driver_common.h 5-193	DRV_ADC_SampleMaxGet function 5-214
DRIVER_COUNT macro 5-325	DRV_ADC_SampleMinGet function 5-215
DriverInterfaceInit function 5-291	DRV_ADC_SAMPLES_PER_INTERRUPT macro 5-224
drv_adc.c 5-202	DRV_ADC_SamplesAvailable function 5-212
drv_adc.h 5-225	DRV_ADC_SamplesRead function 5-215
DRV_ADC_ACQUISITION_TIME macro 5-219	DRV_ADC_SamplesReadLatest function 5-216
DRV_ADC_ALTERNATE_INPUT_SAMPLING_ENABLE macro 5-219	DRV_ADC_Start function 5-213
DRV_ADC_ANALOG_INPUT macro 5-220	DRV_ADC_Status function 5-208
DRV_ADC_AUTO_SAMPLING_ENABLE macro 5-220	DRV_ADC_Stop function 5-214
drv_adc_client_multi.c 5-203	DRV_ADC_STOP_ON_CONVERSION_ENABLE macro 5-224
drv_adc_client_single.c 5-203	DRV_ADC_Tasks function 5-209
DRV_ADC_CLIENT_STATUS enumeration 5-217	DRV_ADC_VOLTAGE_REFERENCE macro 5-225
DRV_ADC_CLIENTS_NUMBER macro 5-220	DRV_CLIENT_STATUS enumeration 5-188
DRV_ADC_ClientStatus function 5-210	DRV_CONFIG_NOT_SUPPORTED macro 5-191
DRV_ADC_Close function 5-210	drv_ethernet_flags.h 5-252, 5-288
DRV_ADC_CONVERSION_CLOCK_PRESCALER macro 5-220	drv_ethmac.h 5-252
DRV_ADC_CONVERSION_CLOCK_SOURCE macro 5-221	DRV_ETHMAC_CLIENTS_NUMBER macro 5-235
DRV_ADC_CONVERSION_TRIGGER_SOURCE macro 5-221	drv_ethmac_config.h 5-253
DRV_ADC_Deinitialize function 5-205	DRV_ETHMAC_INDEX macro 5-235
drv_adc_hw_dynamic.c 5-203	DRV_ETHMAC_INDEX_0 macro 8-4247
drv_adc_hw_static.c 5-203	DRV_ETHMAC_INDEX_1 macro 5-251
DRV_ADC_INDEX macro 5-221	DRV_ETHMAC_INDEX_COUNT macro 8-4247
DRV_ADC_INDEX_0 macro 5-222	DRV_ETHMAC_INSTANCES_NUMBER macro 5-235
DRV_ADC_INDEX_1 macro 5-222	DRV_ETHMAC_INTERRUPT_MODE macro 5-235
DRV_ADC_INDEX_2 macro 5-222	DRV_ETHMAC_INTERRUPT_SOURCE macro 5-236
DRV_ADC_INDEX_COUNT macro 5-222	DRV_ETHMAC_PERIPHERAL_ID macro 5-236
DRV_ADC_INIT structure 5-218	DRV_ETHMAC_PIC32MACClose function 5-238
DRV_ADC_INIT_FLAGS enumeration 5-219	DRV_ETHMAC_PIC32MACEventAcknowledge function 5-243
DRV_ADC_Initialize function 5-206	DRV_ETHMAC_PIC32MACEventMaskSet function 5-244
DRV_ADC_InputsRegister function 5-211	DRV_ETHMAC_PIC32MACEventPendingGet function 5-245

DRV_ETHMAC_PIC32MACGetConfig function 5-238	DRV_ETHPHY_SMIIsBusy function 5-277
DRV_ETHMAC_PIC32MACLinkCheck function 5-239	DRV_ETHPHY_SMIReadResultGet function 5-277
DRV_ETHMAC_PIC32MACOpen function 5-239	DRV_ETHPHY_SMIReadStart function 5-278
DRV_ETHMAC_PIC32MACPacketRx function 5-241	DRV_ETHPHY_SMIScanDataGet function 5-278
DRV_ETHMAC_PIC32MACPacketTx function 5-242	DRV_ETHPHY_SMIScanStart function 5-279
DRV_ETHMAC_PIC32MACPowerMode function 5-246	DRV_ETHPHY_SMIScanStatusGet function 5-279
DRV_ETHMAC_PIC32MACProcess function 5-246	DRV_ETHPHY_SMIScanStop function 5-280
DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet function 5-242	DRV_ETHPHY_SMIWriteStart function 5-280
DRV_ETHMAC_PIC32MACSetup function 5-240	DRV_ETHPHY_Status function 5-266
DRV_ETHMAC_PIC32MACTeardown function 5-240	DRV_ETHPHY_Tasks function 5-267
DRV_ETHMAC_POWER_STATE macro 5-236	DRV_ETHPHY_VersionGet function 5-283
drv_ethphy.h 5-286	DRV_ETHPHY_VersionStrGet function 5-284
DRV_ETHPHY_CLIENT_STATUS enumeration 5-284	drv_gfx_config_template.h 5-299
DRV_ETHPHY_CLIENTS_NUMBER macro 5-260	drv_gfx_display.h 5-295
DRV_ETHPHY_ClientStatus function 5-268	drv_gfx_lcc.h 5-296
DRV_ETHPHY_Close function 5-268	drv_gfx_lcc_config_template.h 5-308
drv_ethphy_config.h 5-288	drv_gfx_otm2201a.h 5-318
DRV_ETHPHY_Deinitialize function 5-264	drv_gfx_s1d13517.h 5-297
DRV_ETHPHY_HWConfigFlagsGet function 5-269	drv_gfx_s1d13517_config_template.h 5-332
DRV_ETHPHY_INDEX macro 5-260	drv_gfx_ssd1926.h 5-298
DRV_ETHPHY_INDEX_0 macro 5-260	drv_gfx_ssd1926_config_template.h 5-341
DRV_ETHPHY_INDEX_1 macro 5-251	DRV_HANDLE type 5-189
DRV_ETHPHY_INDEX_COUNT macro 5-261	DRV_HANDLE_INVALID macro 5-191
DRV_ETHPHY_INIT structure 5-285	DRV_IO_BUFFER_TYPES enumeration 5-189
DRV_ETHPHY_Initialize function 5-264	DRV_IO_INTENT enumeration 5-189
DRV_ETHPHY_INSTANCES_NUMBER macro 5-261	DRV_IO_ISBLOCKING macro 5-191
DRV_ETHPHY_LinkScanRead function 5-270	DRV_IO_ISEXCLUSIVE macro 5-191
DRV_ETHPHY_LinkScanStart function 5-270	DRV_IO_ISNONBLOCKING macro 5-192
DRV_ETHPHY_LinkScanStop function 5-271	drv_nvm.h 5-365
DRV_ETHPHY_LinkStatusGet function 5-271	DRV_NVM_BUFFER_HANDLE type 5-363
DRV_ETHPHY_NegotiationIsComplete function 5-272	DRV_NVM_BUFFER_HANDLE_INVALID macro 5-362
DRV_ETHPHY_NegotiationResultGet function 5-273	DRV_NVM_BUFFER_OBJECT_NUMBER macro 5-349
DRV_ETHPHY_Open function 5-273	DRV_NVM_BUFFER_STATUS enumeration 5-363
DRV_ETHPHY_PERIPHERAL_ID macro 5-261	DRV_NVM_BufferStatus function 5-361
DRV_ETHPHY_Reinitialize function 5-265	DRV_NVM_CLIENT_STATUS enumeration 5-364
DRV_ETHPHY_Reset function 5-274	DRV_NVM_CLIENTS_NUMBER macro 5-350
DRV_ETHPHY_RestartNegotiation function 5-275	DRV_NVM_ClientStatus function 5-355
DRV_ETHPHY_Setup function 5-275	DRV_NVM_Close function 5-356
DRV_ETHPHY_SMIClockSet function 5-276	drv_nvm_config_template.h 5-366

DRV_NVM_Deinitialize function 5-353	DRV_PMP_QUEUE_SIZE macro 5-376
DRV_NVM_Erase function 5-357	DRV_PMP_Read function 5-386
DRV_NVM_ERASE_SIZE macro 5-350	DRV_PMP_Reinitialize function 5-380
DRV_NVM_INDEX_0 macro 5-362	DRV_PMP_Status function 5-381
DRV_NVM_INDEX_1 macro 5-365	DRV_PMP_Tasks function 5-382
DRV_NVM_INDEX_COUNT macro 5-363	DRV_PMP_TRANSFER_STATUS enumeration 5-392
DRV_NVM_INIT structure 5-364	DRV_PMP_TRANSFER_TYPE enumeration 5-393
DRV_NVM_Initialize function 5-353	DRV_PMP_TransferStatus function 5-387
DRV_NVM_INSTANCES_NUMBER macro 5-350	DRV_PMP_VersionGet function 5-394
DRV_NVM_INTERRUPT_MODE macro 5-350	DRV_PMP_VersionStrGet function 5-394
DRV_NVM_Open function 5-357	DRV_PMP_WAIT_STATES structure 5-392
DRV_NVM_Read function 5-359	DRV_PMP_Write function 5-386
DRV_NVM_ROW_SIZE macro 5-351	drv_sdcard.h 5-416
DRV_NVM_Status function 5-354	DRV_SDCARD_BUFFER_HANDLE type 5-414
DRV_NVM_Tasks function 5-360	DRV_SDCARD_BufferStatusGet function 5-409
DRV_NVM_UNLOCK_KEY1 macro 5-351	DRV_SDCARD_CLIENT_STATUS enumeration 5-414
DRV_NVM_UNLOCK_KEY2 macro 5-351	DRV_SDCARD_ClientStatus function 5-405
DRV_NVM_Write function 5-360	DRV_SDCARD_Close function 5-406
drv_pmp.h 5-395	DRV_SDCARD_Deinitialize function 5-401
DRV_PMP_CHIPX_STROBE_MODE enumeration 5-388	DRV_SDCARD_INDEX_0 macro 5-413
DRV_PMP_CLIENT_STATUS enumeration 5-388	DRV_SDCARD_INDEX_1 macro 5-416
DRV_PMP_CLIENTS_NUMBER macro 5-376	DRV_SDCARD_INDEX_2 macro 5-416
DRV_PMP_ClientStatus function 5-383	DRV_SDCARD_INDEX_3 macro 5-416
DRV_PMP_Close function 5-383	DRV_SDCARD_INDEX_COUNT macro 5-414
drv_pmp_config.h 5-396	DRV_SDCARD_INIT type 5-415
DRV_PMP_Deinitialize function 5-378	DRV_SDCARD_Initialize function 5-402
DRV_PMP_ENDIAN_MODE enumeration 5-389	DRV_SDCARD_MediaStatusGet function 5-410
DRV_PMP_INDEX enumeration 5-389	DRV_SDCARD_Open function 5-407
DRV_PMP_INDEX_COUNT macro 5-388	DRV_SDCARD_Reinitialize function 5-403
DRV_PMP_INIT structure 5-389	DRV_SDCARD_SectorRead function 5-407
DRV_PMP_Initialize function 5-379	DRV_SDCARD_SectorsCountGet function 5-410
DRV_PMP_INSTANCES_NUMBER macro 5-376	DRV_SDCARD_SectorSizeGet function 5-411
DRV_PMP_MODE_CONFIG structure 5-390	DRV_SDCARD_SectorWrite function 5-408
DRV_PMP_ModeConfig function 5-384	DRV_SDCARD_Status function 5-404
DRV_PMP_Open function 5-385	DRV_SDCARD_Tasks function 5-405
DRV_PMP_POLARITY_OBJECT structure 5-390	DRV_SDCARD_VersionGet function 5-412
DRV_PMP_PORT_CONTROL enumeration 5-391	DRV_SDCARD_VersionStrGet function 5-413
DRV_PMP_PORTS structure 5-391	DRV_SDCARD_WriteProtectionIsEnabled function 5-412
DRV_PMP_QUEUE_ELEMENT_OBJ type 5-392	drv_spi.h 5-455

DRV_SPI_BAUD_RATE macro 5-443	DRV_SPI_INTERRUPT_MODE macro 5-448
DRV_SPI_BAUD_RATE_CLOCK macro 5-444	DRV_SPI_INTERRUPT_SOURCE_ERROR macro 5-448
DRV_SPI_BUFFER_EVENT enumeration 5-450	DRV_SPI_INTERRUPT_SOURCE_RX macro 5-448
DRV_SPI_BUFFER_EVENT_HANDLER type 5-451	DRV_SPI_INTERRUPT_SOURCE_TX macro 5-449
DRV_SPI_BUFFER_HANDLE type 5-452	DRV_SPI_MODE enumeration 5-454
DRV_SPI_BUFFER_HANDLE_INVALID macro 5-444	DRV_SPI_Open function 5-437
DRV_SPI_BUFFER_SIZE macro 5-444	DRV_SPI_PERIPHERAL_ID macro 5-449
DRV_SPI_BUFFER_TYPE enumeration 5-452	DRV_SPI_PORTS_REMAP_USAGE macro 5-449
DRV_SPI_BUFFER_USAGE_TYPE macro 5-444	DRV_SPI_POWER_STATE macro 5-450
DRV_SPI_BufferAddRead function 5-438	DRV_SPI_PROTOCOL macro 5-450
DRV_SPI_BufferAddWrite function 5-439	DRV_SPI_PROTOCOL_TYPE enumeration 5-454
DRV_SPI_BufferAddWriteRead function 5-440	DRV_SPI_Read function 5-434
DRV_SPI_BufferStatus function 5-435	DRV_SPI_RX_FIFO_INTERRUPT_MODE macro 5-443
DRV_SPI_CLIENT_SETUP type 5-453	DRV_SPI_Status function 5-432
DRV_SPI_CLIENTS_NUMBER macro 5-445	DRV_SPI_Tasks function 5-433
DRV_SPI_ClientSetup function 5-436	DRV_SPI_TX_FIFO_INTERRUPT_MODE macro 5-443
DRV_SPI_CLOCK_MODE enumeration 5-453	DRV_SPI_USAGE_MODE macro 5-450
DRV_SPI_CLOCK_OPERATION_MODE macro 5-445	DRV_SPI_VersionGet function 5-441
DRV_SPI_Close function 5-437	DRV_SPI_VersionStrGet function 5-442
DRV_SPI_COMMUNICATION_WIDTH macro 5-445	DRV_SPI_Write function 5-434
drv_spi_config_template.h 5-456	drv_tmr.h 5-508
DRV_SPI_Deinitialize function 5-430	DRV_TMR_ALARM_CONFIG structure 5-504
DRV_SPI_FRAME_SYNC_PULSE_COUNT macro 5-445	DRV_TMR_ALARM_ENABLE macro 5-479
DRV_SPI_FRAME_SYNC_PULSE_DIRECTION macro 5-446	DRV_TMR_ALARM_PERIODIC macro 5-480
DRV_SPI_FRAME_SYNC_PULSE_EDGE macro 5-446	DRV_TMR_ALARM_TYPE enumeration 5-504
DRV_SPI_FRAME_SYNC_PULSE_POLARITY macro 5-446	DRV_TMR_AlarmCountClear function 5-493
DRV_SPI_FRAME_SYNC_PULSE_WIDTH macro 5-446	DRV_TMR_AlarmCountGet function 5-494
DRV_SPI_INDEX macro 5-447	DRV_TMR_AlarmSet function 5-494
DRV_SPI_INDEX_0 macro 5-447	DRV_TMR_ASYNC_WRITE_ENABLE macro 5-481
DRV_SPI_INDEX_1 macro 5-442	DRV_TMR_CALLBACK type 5-505
DRV_SPI_INDEX_2 macro 5-442	DRV_TMR_CLIENT_STATUS enumeration 5-505
DRV_SPI_INDEX_3 macro 5-443	DRV_TMR_CLIENTS_NUMBER macro 5-480
DRV_SPI_INDEX_4 macro 5-443	DRV_TMR_ClientStatus function 5-488
DRV_SPI_INDEX_5 macro 5-443	DRV_TMR_CLOCK_SOURCE macro 5-474
DRV_SPI_INDEX_COUNT macro 5-447	DRV_TMR_Close function 5-489
DRV_SPI_INIT type 5-453	drv_tmr_config_alarm.h 5-481
DRV_SPI_Initialize function 5-431	drv_tmr_config_template.h 5-509
DRV_SPI_INPUT_SAMPLE_PHASE macro 5-447	DRV_TMR_COUNT_WIDTH macro 5-477
DRV_SPI_INSTANCES_NUMBER macro 5-448	DRV_TMR_Counter16BitGet function 5-498

DRV_TMR_Counter16BitSet function 5-499	DRV_TMR_SYNCHRONIZATION_MODE macro 5-476
DRV_TMR_Counter32BitGet function 5-499	DRV_TMR_SyncModeGet function 5-492
DRV_TMR_Counter32BitSet function 5-500	DRV_TMR_SyncModeSet function 5-492
DRV_TMR_Counter8BitGet function 5-500	DRV_TMR_Tasks function 5-487
DRV_TMR_Counter8BitSet function 5-501	DRV_TMR_TickFrequencyGet function 5-502
DRV_TMR_Deinitialize function 5-483	DRV_TMR_TIMER_PERIOD macro 5-476
DRV_TMR_ElapsedStatusGetAndClear function 5-489	DRV_TMR_UNIFIED macro 5-507
DRV_TMR_INDEX macro 5-477	DRV_TMR_VersionGet function 5-503
DRV_TMR_INDEX_0 macro 5-477	DRV_TMR_VersionStrGet function 5-503
DRV_TMR_INDEX_1 macro 5-477	drv_usart.h 5-549
DRV_TMR_INDEX_2 macro 5-478	DRV_USART_AUTO_BAUD_ENABLE macro 5-536
DRV_TMR_INDEX_3 macro 5-478	DRV_USART_BAUD_RATE macro 5-537
DRV_TMR_INDEX_4 macro 5-478	DRV_USART_BUFFER_FLAGS enumeration 5-543
DRV_TMR_INDEX_5 macro 5-478	DRV_USART_BUFFER_STATUS enumeration 5-543
DRV_TMR_INDEX_COUNT macro 5-478	DRV_USART_BufferAdd function 5-526
DRV_TMR_INIT structure 5-506	DRV_USART_BufferStatus function 5-527
DRV_TMR_Initialize function 5-484	DRV_USART_BufferTransferStatus function 5-532
DRV_TMR_INSTANCES_NUMBER macro 5-478	DRV_USART_BYTE_Q_SIZE_RX macro 5-537
DRV_TMR_INTERRUPT_MODE macro 5-479	DRV_USART_BYTE_Q_SIZE_TX macro 5-537
DRV_TMR_INTERRUPT_SOURCE macro 5-507	DRV_USART_CALLBACK type 5-544
DRV_TMR_Open function 5-490	DRV_USART_CLIENT_STATUS enumeration 5-544
DRV_TMR_OperatingFrequencyGet function 5-502	DRV_USART_CLIENTS_NUMBER macro 5-537
DRV_TMR_Period16BitGet function 5-495	DRV_USART_ClientStatus function 5-533
DRV_TMR_Period16BitSet function 5-495	DRV_USART_Close function 5-534
DRV_TMR_Period32BitGet function 5-496	drv_usart_config_template.h 5-551
DRV_TMR_Period32BitSet function 5-497	DRV_USART_Deinitialize function 5-521
DRV_TMR_Period8BitGet function 5-497	DRV_USART_HANDSHAKE_MODE macro 5-537
DRV_TMR_Period8BitSet function 5-498	DRV_USART_HANDSHAKE_MODES enumeration 5-545
DRV_TMR_PERIPHERAL_ID macro 5-475	DRV_USART_INDEX macro 5-538
DRV_TMR_POSTSCALE macro 5-475	DRV_USART_INDEX_0 macro 5-538
DRV_TMR_POWER_STATE macro 5-507	DRV_USART_INDEX_1 macro 5-538
DRV_TMR_PRESCALE macro 5-475	DRV_USART_INDEX_2 macro 5-538
DRV_TMR_PRESCALER_ENABLE macro 5-481	DRV_USART_INDEX_3 macro 5-538
DRV_TMR_Reinitialize function 5-485	DRV_USART_INDEX_4 macro 5-539
DRV_TMR_SOURCE_EDGE macro 5-475	DRV_USART_INDEX_5 macro 5-539
DRV_TMR_Start function 5-491	DRV_USART_INDEX_COUNT macro 5-539
DRV_TMR_Status function 5-486	DRV_USART_INIT structure 5-545
DRV_TMR_Stop function 5-491	DRV_USART_INIT_FLAGS enumeration 5-546
DRV_TMR_SYNC_MODE enumeration 5-506	DRV_USART_Initialize function 5-522

DRV_USART_INSTANCES_NUMBER macro 5-539	DRV_USB_DEVICE_EndpointDisableAll function 5-585
DRV_USART_INTERRUPT_MODE macro 5-539	DRV_USB_DEVICE_EndpointEnable function 5-586
DRV_USART_INTERRUPT_SOURCE_ERROR macro 5-540	DRV_USB_DEVICE_EndpointIsEnabled function 5-587
DRV_USART_INTERRUPT_SOURCE_RX macro 5-540	DRV_USB_DEVICE_EndpointIsStalled function 5-588
DRV_USART_INTERRUPT_SOURCE_TX macro 5-540	DRV_USB_DEVICE_EndpointStall function 5-589
DRV_USART_IO_BUFFER structure 5-546	DRV_USB_DEVICE_EndpointStallClear function 5-589
DRV_USART_LINE_CONTROL macro 5-540	DRV_USB_DEVICE_IRPCancelAll function 5-590
DRV_USART_LINE_CONTROL_MODES enumeration 5-547	DRV_USB_DEVICE_IRPSubmit function 5-591
DRV_USART_Open function 5-534	DRV_USB_ENDPOINT_TABLE_ENTRY_SIZE macro 5-595
DRV_USART_OPERATION_MODE macro 5-541	DRV_USB_EVENT enumeration 5-596
DRV_USART_OPERATION_MODE_INIT union 5-547	DRV_USB_EVENT_CALLBACK type 5-597
DRV_USART_OPERATION_MODES enumeration 5-548	DRV_USB_HOST_BusResetControl function 5-593
DRV_USART_PERIPHERAL_ID macro 5-541	DRV_USB_HOST_DeviceCurrentSpeedGet function 5-593
DRV_USART_POLARITY_INVERTED_RX macro 5-541	DRV_USB_HOST_IRPCancel function 5-593
DRV_USART_POLARITY_INVERTED_TX macro 5-542	DRV_USB_HOST_IRPSubmit function 5-593
DRV_USART_POWER_STATE macro 5-542	DRV_USB_HOST_OperationEnable function 5-593
DRV_USART_Read function 5-528	DRV_USB_HOST_OperationIsEnabled function 5-578
DRV_USART_ReadByte function 5-529	DRV_USB_HOST_PIPE_HANDLE type 5-597
DRV_USART_Reinitialize function 5-523	DRV_USB_HOST_PIPE_HANDLE_INVALID macro 5-595
DRV_USART_Status function 5-535	DRV_USB_HOST_PipeClose function 5-581
DRV_USART_TasksError function 5-524	DRV_USB_HOST_PipeSetup function 5-580
DRV_USART_TasksRX function 5-525	DRV_USB_HOST_StartOfFrameControl function 5-594
DRV_USART_TasksTX function 5-525	DRV_USB_HOST_TimerIsComplete function 5-594
DRV_USART_TRANSFER_STATUS enumeration 5-548	DRV_USB_HOST_TimerReset function 5-594
DRV_USART_TransferStatus function 5-530	DRV_USB_HOST_TimerStart function 5-594
DRV_USART_WAKE_ON_START_ENABLE macro 5-542	DRV_USB_INDEX macro 5-595
DRV_USART_Write function 5-530	DRV_USB_INDEX_0 macro 5-596
DRV_USART_WriteByte function 5-531	DRV_USB_INIT structure 5-597
DRV_USART_XFER_BUFFER_NUMBER macro 5-542	DRV_USB_Initialize function 5-576
DRV_USART_XFER_HANDLE type 5-549	DRV_USB_INTERRUPT_SOURCE macro 5-596
drv_usb.h 5-598	DRV_USB_Open function 5-579
DRV_USB_ClientEventCallbackSet function 5-581	DRV_USB_PERIPHERAL_ID macro 5-596
DRV_USB_Close function 5-579	DRV_USB_Status function 5-577
drv_usb_config_template.h 5-600	DRV_USB_Tasks function 5-594
DRV_USB_DEVICE_AddressSet function 5-582	DRV_USB_Tasks_ISR function 5-578
DRV_USB_DEVICE_Attach function 5-582	drv_wifi.h 5-674
DRV_USB_DEVICE_CurrentSpeedGet function 5-583	DRV_WIFI_ADHOC_CONNECT_ONLY enumeration member 5-655
DRV_USB_DEVICE_Detach function 5-584	DRV_WIFI_ADHOC_CONNECT_THEN_START enumeration member 5-655
DRV_USB_DEVICE_EndpointDisable function 5-585	

DRV_WIFI_ADHOC_MODES enumeration 5-655	DRV_WIFI_DOMAIN_CODES enumeration 5-657
DRV_WIFI_ADHOC_NETWORK_CONTEXT structure 5-655	DRV_WIFI_ENABLED macro 5-650
DRV_WIFI_ADHOC_START_ONLY enumeration member 5-655	DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES enumeration 5-658
DRV_WIFI_AdhocContextSet function 5-636	DRV_WIFI_EVENT_INFO enumeration 5-658
DRV_WIFI_BSSID_LENGTH macro 5-647	DRV_WIFI_EVENTS enumeration 5-659
DRV_WIFI_BssidGet function 5-613	DRV_WIFI_GENERAL_ERRORS enumeration 5-660
DRV_WIFI_BssidSet function 5-614	DRV_WIFI_GratuitousArpStart function 5-638
DRV_WIFI_ChannelListGet function 5-615	DRV_WIFI_GratuitousArpStop function 5-638
DRV_WIFI_ChannelListSet function 5-615	DRV_WIFI_HIBERNATE_STATE structure 5-660
DRV_WIFI_ConfigDataErase function 5-643	DRV_WIFI_HIBERNATE_STATES enumeration 5-660
DRV_WIFI_ConfigDataLoad function 5-643	DRV_WIFI_HibernateEnable function 5-639
DRV_WIFI_ConfigDataPrint function 5-644	DRV_WIFI_HWMulticastFilterGet function 5-617
DRV_WIFI_ConfigDataSave function 5-644	DRV_WIFI_HWMulticastFilterSet function 5-618
DRV_WIFI_Connect function 5-641	DRV_WIFI_Initialize function 5-642
DRV_WIFI_ConnectContextGet function 5-616	DRV_WIFI_isHibernateEnable function 5-639
DRV_WIFI_CONNECTION_CONTEXT structure 5-656	DRV_WIFI_MAC_STATS structure 5-660
DRV_WIFI_CONNECTION_STATES enumeration 5-656	DRV_WIFI_MacAddressGet function 5-619
DRV_WIFI_ConnectionStateGet function 5-616	DRV_WIFI_MacAddressSet function 5-619
DRV_WIFI_DEAUTH_REASONCODE_MASK macro 5-647	DRV_WIFI_MacStatsGet function 5-620
DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD macro 5-647	DRV_WIFI_MAX_CHANNEL_LIST_LENGTH macro 5-650
DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID macro 5-648	DRV_WIFI_MAX_NUM_RATES macro 5-650
DRV_WIFI_DEFAULT_ADHOC_MODE macro 5-648	DRV_WIFI_MAX_SECURITY_KEY_LENGTH macro 5-650
DRV_WIFI_DEFAULT_PS_DTIM_ENABLED macro 5-648	DRV_WIFI_MAX_SSID_LENGTH macro 5-650
DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL macro 5-648	DRV_WIFI_MAX_WEP_KEY_LENGTH macro 5-650
DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL macro 5-648	DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH macro 5-650
DRV_WIFI_DEFAULT_SCAN_COUNT macro 5-648	DRV_WIFI_MAX_WPA2_PASS_PHRASE_LENGTH macro 5-651
DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME macro 5-649	DRV_WIFI_MGMT_ERRORS enumeration 5-661
DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME macro 5-649	DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT structure 5-662
DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY macro 5-649	DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH macro 5-651
DRV_WIFI_DEFAULT_WEP_KEY_TYPE macro 5-649	DRV_WIFI_MULTICAST_FILTER_IDS enumeration 5-663
DRV_WIFI_Deinitialize function 5-641	DRV_WIFI_MULTICAST_FILTERS enumeration 5-663
DRV_WIFI_DEVICE_INFO structure 5-657	DRV_WIFI_NETWORK_TYPE_ADHOC macro 5-651
DRV_WIFI_DEVICE_TYPES enumeration 5-657	DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE macro 5-651
DRV_WIFI_DeviceInfoGet function 5-617	DRV_WIFI_NETWORK_TYPE_P2P macro 5-651
DRV_WIFI_DISABLED macro 5-649	DRV_WIFI_NETWORK_TYPE_SOFT_AP macro 5-651
DRV_WIFI_DISASSOC_REASONCODE_MASK macro 5-649	DRV_WIFI_NetworkTypeGet function 5-620
DRV_WIFI_Disconnect function 5-641	

DRV_WIFI_NetworkTypeSet function 5-621	5-654
DRV_WIFI_NO_ADDITIONAL_INFO macro 5-652	DRV_WIFI_SECURITY_WPS_PIN macro 5-654
DRV_WIFI_POWER_SAVE_STATES enumeration 5-664	DRV_WIFI_SECURITY_WPS_PUSH_BUTTON macro 5-654
DRV_WIFI_PowerSaveStateGet function 5-621	DRV_WIFI_SecurityGet function 5-627
DRV_WIFI_ProcessEvent function 5-644	DRV_WIFI_SecurityOpenSet function 5-628
DRV_WIFI_PS_POLL_CONTEXT structure 5-664	DRV_WIFI_SecurityWepSet function 5-628
DRV_WIFI_PsPollDisable function 5-640	DRV_WIFI_SecurityWpaSet function 5-629
DRV_WIFI_PsPollEnable function 5-640	DRV_WIFI_SecurityWpsSet function 5-630
DRV_WIFI_REASON_CODES enumeration 5-665	DRV_WIFI_SetLinkDownThreshold function 5-645
DRV_WIFI_RECONNECT_MODES enumeration 5-665	DRV_WIFI_SetPSK function 5-646
DRV_WIFI_ReconnectModeGet function 5-622	DRV_WIFI_SOFT_AP_EVENT_REASON_CODES enumeration 5-667
DRV_WIFI_ReconnectModeSet function 5-623	DRV_WIFI_SOFT_AP_STATES enumeration 5-668
DRV_WIFI_RegionalDomainGet function 5-624	DRV_WIFI_SoftApEventInfoGet function 5-630
DRV_WIFI_RETRY_ADHOC macro 5-652	DRV_WIFI_SsidGet function 5-631
DRV_WIFI_RETRY_FOREVER macro 5-652	DRV_WIFI_SsidSet function 5-631
DRV_WIFI_RssiGet function 5-637	DRV_WIFI_STATUS_CODES enumeration 5-668
DRV_WIFI_RssiSet function 5-637	DRV_WIFI_SWMULTICAST_CONFIG structure 5-668
DRV_WIFI_RTS_THRESHOLD_MAX macro 5-652	DRV_WIFI_SWMultiCastFilterEnable function 5-646
DRV_WIFI_RtsThresholdGet function 5-625	DRV_WIFI_SWMulticastFilterSet function 5-632
DRV_WIFI_RtsThresholdSet function 5-625	DRV_WIFI_TX_MODES enumeration 5-669
DRV_WIFI_Scan function 5-642	DRV_WIFI_TxModeGet function 5-633
DRV_WIFI_SCAN_CONTEXT structure 5-665	DRV_WIFI_TxModeSet function 5-633
DRV_WIFI_SCAN_RESULT structure 5-666	DRV_WIFI_TxPowerFactoryMaxGet function 5-634
DRV_WIFI_SCAN_TYPES enumeration 5-667	DRV_WIFI_TxPowerMaxGet function 5-634
DRV_WIFI_ScanContextGet function 5-626	DRV_WIFI_TxPowerMaxSet function 5-635
DRV_WIFI_ScanContextSet function 5-627	DRV_WIFI_VERSION_NUMBER macro 5-654
DRV_WIFI_ScanGetResult function 5-645	DRV_WIFI_WEP_CONTEXT structure 5-669
DRV_WIFI_SECURITY_CONTEXT union 5-667	DRV_WIFI_WEP_KEY_TYPE enumeration 5-670
DRV_WIFI_SECURITY_EAP macro 5-652	DRV_WIFI_WEP104_KEY_LENGTH macro 5-654
DRV_WIFI_SECURITY_OPEN macro 5-652	DRV_WIFI_WEP40_KEY_LENGTH macro 5-654
DRV_WIFI_SECURITY_WEP_104 macro 5-653	DRV_WIFI_WepKeyTypeGet function 5-635
DRV_WIFI_SECURITY_WEP_40 macro 5-653	DRV_WIFI_WPA_CONTEXT structure 5-670
DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY macro 5-653	DRV_WIFI_WPA_KEY_INFO structure 5-671
DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE macro 5-653	DRV_WIFI_WPA_KEY_LENGTH macro 5-655
DRV_WIFI_SECURITY_WPA_WITH_KEY macro 5-653	DRV_WIFI_WPA2_KEY_LENGTH macro 5-655
DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE macro 5-653	DRV_WIFI_WPS_AUTH_TYPES enumeration 5-671
DRV_WIFI_SECURITY_WPA2_WITH_KEY macro 5-654	DRV_WIFI_WPS_CONTEXT structure 5-671
DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE macro	DRV_WIFI_WPS_CREDENTIAL structure 5-672
	DRV_WIFI_WPS_ENCODE_TYPES enumeration 5-673

DRV_WIFI_WPS_PIN_LENGTH macro 5-655	DSP_TransformWindow_Bart32 function 5-1210
DRV_WIFI_WPSCredentialsGet function 5-636	DSP_TransformWindow_Black16 function 5-1211
DRV_WIFI_YieldPassphraseToHost function 5-647	DSP_TransformWindow_Black32 function 5-1212
DSP Fixed-Point Library 5-1167	DSP_TransformWindow_Cosine16 function 5-1213
DSP Math Library 4-181	DSP_TransformWindow_Cosine32 function 5-1213
dsp.h 5-1268	DSP_TransformWindow_Hamm16 function 5-1214
DSP_ComplexAdd32 function 5-1175	DSP_TransformWindow_Hamm32 function 5-1215
DSP_ComplexConj16 function 5-1175	DSP_TransformWindow_Hann16 function 5-1216
DSP_ComplexConj32 function 5-1176	DSP_TransformWindow_Hann32 function 5-1217
DSP_ComplexDotProd32 function 5-1177	DSP_TransformWindow_Kaiser16 function 5-1217
DSP_ComplexMult32 function 5-1178	DSP_TransformWindow_Kaiser32 function 5-1218
DSP_ComplexScalarMult32 function 5-1179	DSP_TransformWinInit_Bart16 function 5-1219
DSP_ComplexSub32 function 5-1180	DSP_TransformWinInit_Bart32 function 5-1220
DSP_FilterFIR32 function 5-1180	DSP_TransformWinInit_Black16 function 5-1220
DSP_FilterFIRDecim32 function 5-1182	DSP_TransformWinInit_Black32 function 5-1221
DSP_FilterFIRInterp32 function 5-1183	DSP_TransformWinInit_Cosine16 function 5-1222
DSP_FilterIIR16 function 5-1184	DSP_TransformWinInit_Cosine32 function 5-1223
DSP_FilterIIRBQ16 function 5-1185	DSP_TransformWinInit_Hamm16 function 5-1223
DSP_FilterIIRBQ16_cascade8 function 5-1187	DSP_TransformWinInit_Hamm32 function 5-1224
DSP_FilterIIRBQ16_cascade8_fast function 5-1188	DSP_TransformWinInit_Hann16 function 5-1225
DSP_FilterIIRBQ16_fast function 5-1190	DSP_TransformWinInit_Hann32 function 5-1225
DSP_FilterIIRBQ16_parallel8 function 5-1191	DSP_TransformWinInit_Kaiser16 function 5-1226
DSP_FilterIIRBQ16_parallel8_fast function 5-1193	DSP_TransformWinInit_Kaiser32 function 5-1227
DSP_FilterIIRBQ32 function 5-1194	DSP_VectorAbs16 function 5-1228
DSP_FilterIIRSetup16 function 5-1195	DSP_VectorAbs32 function 5-1228
DSP_FilterLMS16 function 5-1196	DSP_VectorAdd16 function 5-1229
DSP_MatrixAdd32 function 5-1197	DSP_VectorAdd32 function 5-1230
DSP_MatrixEqual32 function 5-1199	DSP_VectorAddc16 function 5-1231
DSP_MatrixInit32 function 5-1199	DSP_VectorAddc32 function 5-1232
DSP_MatrixMul32 function 5-1200	DSP_VectorAutocorr16 function 5-1233
DSP_MatrixScale32 function 5-1202	DSP_VectorBexp16 function 5-1234
DSP_MatrixSub32 function 5-1202	DSP_VectorBexp32 function 5-1234
DSP_MatrixTranspose32 function 5-1204	DSP_VectorChkEqu32 function 5-1235
DSP_TransformFFT16 function 5-1205	DSP_VectorCopy function 5-1236
DSP_TransformFFT16_setup function 5-1206	DSP_VectorCopyReverse32 function 5-1237
DSP_TransformFFT32 function 5-1206	DSP_VectorDivC function 5-1238
DSP_TransformFFT32_setup function 5-1207	DSP_VectorDotp16 function 5-1238
DSP_TransformIFFT16 function 5-1208	DSP_VectorDotp32 function 5-1239
DSP_TransformWindow_Bart16 function 5-1209	DSP_VectorExp function 5-1240

DSP_VectorFill function 5-1241
 DSP_VectorLn function 5-1242
 DSP_VectorLog10 function 5-1243
 DSP_VectorLog2 function 5-1244
 DSP_VectorMax32 function 5-1244
 DSP_VectorMaxIndex32 function 5-1245
 DSP_VectorMean32 function 5-1246
 DSP_VectorMin32 function 5-1247
 DSP_VectorMinIndex32 function 5-1247
 DSP_VectorMul16 function 5-1248
 DSP_VectorMul32 function 5-1249
 DSP_VectorMulc16 function 5-1250
 DSP_VectorMulc32 function 5-1251
 DSP_VectorNegate function 5-1252
 DSP_VectorRecip function 5-1253
 DSP_VectorRMS16 function 5-1254
 DSP_VectorShift function 5-1254
 DSP_VectorSqrt function 5-1255
 DSP_VectorStdDev16 function 5-1256
 DSP_VectorSub16 function 5-1257
 DSP_VectorSub32 function 5-1258
 DSP_VectorSumSquares16 function 5-1259
 DSP_VectorSumSquares32 function 5-1259
 DSP_VectorVari16 function 5-1260
 DSP_VectorVariance function 5-1261
 DSP_VectorZeroPad function 5-1262
 Dual Compare Continuous Mode 5-2292
 Dynamic Configuration 5-3426
 Dynamic DNS TCP/IP Stack Library 5-3646

E

EBI Peripheral Library 5-1864
 EMAC_RX_BUFF_SIZE macro 5-3750
 EMAC_RX_DESCRIPTOR macro 5-3750
 EMAC_RX_FRAGMENTS macro 5-3750
 EMAC_RX_MAX_FRAME macro 5-3751
 EMAC_TX_DESCRIPTOR macro 5-3751
 ENABLE_P2P_PRINTS macro 5-673
 ENABLE_WPS_PRINTS macro 5-673

EnablePalette function 5-828
 Enabling 5-3956
 Enabling Events 5-1560
 END_OF_SNMP_READ_COMMUNITIES macro 5-3826
 END_OF_SNMP_WRITE_COMMUNITIES macro 5-3826
 Endpoint Pipes 5-557
 Enhanced Buffer Master Mode 5-2807
 Enhanced Buffer Slave Mode 5-2809
 Enhanced Buffer SPI Mode 5-2807
 Error Operation 5-1321
 ETH_AUTOPAD_OPTION enumeration 5-2050
 ETH_CFG_10 macro 5-3751
 ETH_CFG_100 macro 5-3751
 ETH_CFG_AUTO macro 5-3751
 ETH_CFG_AUTO_MDIX macro 5-3752
 ETH_CFG_FDUPLEX macro 5-3752
 ETH_CFG_HDUPLEX macro 5-3752
 ETH_CFG_LINK macro 5-3752
 ETH_CFG_LINK_INIT_DELAY macro 5-3752
 ETH_CFG_SWAP_MDIX macro 5-3752
 ETH_CLOSE_FLAGS enumeration 5-247
 ETH_INTERRUPT_SOURCES enumeration 5-2050
 ETH_LINK_STATUS enumeration 5-247
 ETH_MIIM_CLK enumeration 5-2050
 ETH_MODULE_STATUS enumeration 5-248
 ETH_OPEN_FLAGS enumeration 5-248
 ETH_PATTERN_MATCH_MODE type 5-2051
 ETH_PAUSE_TYPE enumeration 5-249
 ETH_RECEIVE_FILTER enumeration 5-2051
 ETH_RESULT_CODE enumeration 5-250
 ETH_RMII_SPEED enumeration 5-2052
 Ethernet Controller Operation 5-1909
 Ethernet DMA and Buffer Management Engine 5-1912
 Ethernet Frame Overview 5-1909
 Ethernet MAC Driver Library 5-226
 Ethernet Peripheral Library 5-1902
 Ethernet PHY Driver Library 5-254
 ETHPHY_CONFIG_FLAGS enumeration 5-251
 Event Handling 5-563, 5-3975, 5-3981, 5-4066, 5-4135

Events 5-1317
 Example - Channel Scanning 5-1476
 Example - Digital Comparator 5-1478
 Example - Digital Oversampling Filter 5-1477
 Example - Simultaneous Sampling Three Class 1 Inputs 5-1475
 Example Code for Complete Operation 5-374
 Example Usage of the Timer Driver 5-467
 Examples 5-3514
 Examples - Sample Functionality 5-609, 5-1324, 5-3366
 Exception Generator 5-1522
 Explorer 16 Development Board 3-78, 3-104, 3-124, 3-143, 3-171
 Explorer 16 PIC32MX795F512L BSP Library 7-4230
 Extended ID Message Format 5-1556
 EXTPHY_MDI_CONFIGURE type 5-281
 EXTPHY_MI_CONFIGURE type 5-281
 EXTPHY_SMI_ADDRESS_GET type 5-282
 EXTPHY_SMICLOCK_GET type 5-282

F

Fail-Safe Clock Monitor 5-2335
 FAT_FS_MAX_LFN macro 5-3417
 FAT_FS_MAX_SS macro 5-3417
 FAT_FS_USE_LFN macro 5-3417
 File EOF 5-3395
 File Seek 5-3396
 File System Demonstrations 3-64
 File System Service Library 5-3376
 File Tell 5-3396
 FileFeof type 5-889
 FileRead type 5-890

Files 3-111, 3-128, 3-148, 5-192, 5-225, 5-252, 5-286, 5-295, 5-308, 5-318, 5-332, 5-341, 5-365, 5-395, 5-416, 5-454, 5-508, 5-549, 5-598, 5-608, 5-674, 5-1062, 5-1126, 5-1164, 5-1268, 5-1307, 5-1358, 5-1466, 5-1518, 5-1551, 5-1662, 5-1730, 5-1858, 5-1900, 5-2052, 5-2166, 5-2200, 5-2250, 5-2285, 5-2328, 5-2423, 5-2475, 5-2588, 5-2659, 5-2686, 5-2702, 5-2759, 5-2796, 5-2887, 5-3006, 5-3081, 5-3164, 5-3312, 5-3330, 5-3345, 5-3362, 5-3374, 5-3418, 5-3437, 5-3463, 5-3498, 5-3528, 5-3534, 5-3573, 5-3596, 5-3613, 5-3623, 5-3634, 5-3645, 5-3653, 5-3658, 5-3687, 5-3696, 5-3707, 5-3716, 5-3745, 5-3769, 5-3793, 5-3797, 5-3805, 5-3808, 5-3821, 5-3853, 5-3866, 5-3879, 5-3910, 5-3916, 5-3928, 5-3952, 5-3962, 5-4002, 5-4057, 5-4098, 5-4129, 5-4166,

5-4178, 5-4198, 5-4223, 7-4245
 Files to Include 5-3979, 5-4007, 5-4064, 5-4104, 5-4133
 FileSeek type 5-890
 FileTell type 5-890
 Filters and Masks Configuration 5-1559
 Fixed-Point Math Library 4-182
 Flash ECC Operations 5-2434
 Flash Operations 5-2256
 Flow Control Overview 5-1911
 FONT_ORIENTATION enumeration 5-890
 Fonts 5-1123
 Forming Transfers 5-2084
 Framed SPI Modes 5-2810
 Framework Help 5-184
 FreeRTOS RTOS with Graphics 3-116
 FreeRTOS_basic_demo 3-114
 FTP TCP/IP Stack Library 5-3654
 ftp.h 5-3658
 ftp_config.h 5-3659
 FTP_MODULE_CONFIG structure 5-3658
 Functions 5-301, 5-311, 5-326, 5-335, 5-4201
 Functions Registration Table 5-4013

G

Gated Timer 5-3019
 General 5-1367
 General Configuration 5-1736
 Generating Server Certificates 5-3870
 generic_device 3-164
 GENERIC_TRAP_NOTIFICATION_TYPE enumeration 5-3847
 gethostname function 5-3604
 GetPaletteChangeError function 5-828
 Getting Help 5-3542
 Getting Started 5-3378
 gfx.h 5-1062
 GFX_ActivePageGet macro 5-947
 GFX_ALIGNMENT enumeration 5-890
 GFX_ALPHA_PARAMS structure 5-295
 GFX_AlphaBlendingValueGet function 5-828

GFX_AlphaBlendingValueSet function 5-829	GFX_DOUBLE_BUFFERING_MODE structure 5-892
GFX_AlphaBlendWindow function 5-829	GFX_DoubleBufferAreaGet function 5-840
GFX_AlphaParamsSet macro 5-947	GFX_DoubleBufferAreaMark function 5-840
GFX_AnalogClockCreate function 5-718	GFX_DoubleBufferDisable function 5-840
GFX_BACKGROUND structure 5-891	GFX_DoubleBufferEnable function 5-840
GFX_BACKGROUND_TYPE enumeration 5-891	GFX_DoubleBufferStatusGet function 5-841
GFX_BackgroundColorGet function 5-830	GFX_DoubleBufferSyncAllStatusClear function 5-841
GFX_BackgroundImageGet function 5-830	GFX_DoubleBufferSyncAllStatusGet function 5-841
GFX_BackgroundImageLeftGet function 5-830	GFX_DoubleBufferSyncAllStatusSet function 5-841
GFX_BackgroundImageTopGet function 5-831	GFX_DoubleBufferSyncAreaCountGet function 5-841
GFX_BackgroundSet function 5-831	GFX_DoubleBufferSyncAreaCountSet function 5-841
GFX_BackgroundTypeGet function 5-832	GFX_DoubleBufferSynchronize function 5-841
GFX_BackgroundTypeSet function 5-832	GFX_DoubleBufferSynchronizeCancel function 5-841
GFX_BarAlphaDraw function 5-832	GFX_DoubleBufferSynchronizeRequest function 5-841
GFX_BarDraw function 5-833	GFX_DoubleBufferSynchronizeSet function 5-841
GFX_BarGradientDraw function 5-835	GFX_DoubleBufferSynchronizeStatusGet function 5-841
GFX_BEVEL_RENDER_TYPE enumeration 5-892	GFX_DoubleBufferVisualPageUpdate function 5-842
GFX_BevelDraw function 5-836	GFX_DrawBufferGet function 5-842
GFX_BevelDrawTypeGet function 5-836	GFX_DrawBufferSelect function 5-842
GFX_BevelDrawTypeSet function 5-836	GFX_DrawBufferSet function 5-842
GFX_BevelGradientDraw function 5-836	GFX_DRV_DATA structure 5-292
GFX_BUFFER1 macro 5-948	GFX_DRV_FontSet function 5-842
GFX_BUFFER2 macro 5-948	GFX_DRV_FUNCTIONS structure 5-293
GFX_CircleDraw function 5-838	GFX_DRV_lcc_BrightnessSet function 5-302
GFX_CircleFillDraw function 5-838	GFX_DRV_lcc_Close function 5-301
GFX_COLOR type 5-892	GFX_DRV_lcc_COMMAND structure 5-306
GFX_ColorGet function 5-839	GFX_DRV_lcc_COMMANDQUEUESIZE macro 5-307
gfx_colors.h 5-1063	GFX_DRV_lcc_Initialize function 5-302
gfx_colors_w3.h 5-1064	GFX_DRV_lcc_Layer function 5-305
gfx_colors_x11.h 5-1068	GFX_DRV_lcc_Open function 5-302
GFX_ColorSet function 5-839	GFX_DRV_lcc_PixelArrayGet function 5-303
GFX_COMP_MASK macro 5-948	GFX_DRV_lcc_PixelArrayPut function 5-303
GFX_CONFIG_DRIVER_COUNT macro 5-294	GFX_DRV_lcc_PixelPut function 5-303
GFX_CONFIG_OBJECT_METER_DEGREECOUNT macro 5-948	GFX_DRV_lcc_PixelsPut function 5-304
GFX_CONFIG_OBJECT_METER_RESOLUTION macro 5-948	GFX_DRV_lcc_SetColor function 5-304
GFX_CONFIG_OBJECT_METER_SCALE_COUNT macro 5-949	GFX_DRV_lcc_SetInstance function 5-304
gfx_config_template.h 5-1071	GFX_DRV_lcc_SetPage function 5-305
GFX_CosineGet macro 5-1125	GFX_DRV_lcc_Tasks function 5-302
	GFX_DRV_OTM2201A_AddressSet function 5-312

GFX_DRV_OTM2201A_BrightnessSet function 5-312	GFX_DRV_SSD1926_PixelArrayGet function 5-338
GFX_DRV_OTM2201A_Busy function 5-312	GFX_DRV_SSD1926_PixelArrayPut function 5-338
GFX_DRV_OTM2201A_Close function 5-313	GFX_DRV_SSD1926_PixelPut function 5-339
GFX_DRV_OTM2201A_ColorSet function 5-313	GFX_DRV_SSD1926_PixelsPut function 5-339
GFX_DRV_OTM2201A_COMMAND structure 5-317	GFX_DRV_SSD1926_SetColor function 5-339
GFX_DRV_OTM2201A_Initialize function 5-313	GFX_DRV_SSD1926_SetInstance function 5-339
GFX_DRV_OTM2201A_InstanceSet function 5-314	GFX_DRV_SSD1926_SetReg function 5-340
GFX_DRV_OTM2201A_Open function 5-314	GFX_DRV_SSD1926_TASK structure 5-341
GFX_DRV_OTM2201A_PixelArrayGet function 5-314	GFX_DRV_SSD1926_Tasks function 5-340
GFX_DRV_OTM2201A_PixelArrayPut function 5-315	GFX_DRV_TextStringWidthGet function 5-842
GFX_DRV_OTM2201A_PixelPut function 5-315	GFX_ExternalCharInfoGet function 5-843
GFX_DRV_OTM2201A_PixelsPut function 5-316	GFX_ExternalResourceCallback function 5-843
GFX_DRV_OTM2201A_RegGet function 5-316	GFX_FEATURE_STATUS enumeration 5-893
GFX_DRV_OTM2201A_RegSet function 5-316	GFX_FILL_STYLE enumeration 5-893
GFX_DRV_OTM2201A_Tasks function 5-317	GFX_FillStyleGet function 5-844
GFX_DRV_S1D13517_AlphaBlendWindow function 5-327	GFX_FillStyleSet function 5-844
GFX_DRV_S1D13517_BrightnessSet function 5-327	GFX_FlashCharInfoGet function 5-845
GFX_DRV_S1D13517_Close function 5-327	GFX_FONT_ANTIALIAS_TYPE enumeration 5-893
GFX_DRV_S1D13517_COMMAND structure 5-331	GFX_FONT_CURRENT structure 5-894
GFX_DRV_S1D13517_GetReg function 5-327	GFX_FONT_GLYPH_ENTRY structure 5-894
GFX_DRV_S1D13517_Initialize function 5-328	GFX_FONT_GLYPH_ENTRY_EXTENDED structure 5-895
GFX_DRV_S1D13517_Layer function 5-328	GFX_FONT_HEADER structure 5-895
GFX_DRV_S1D13517_Open function 5-328	GFX_FONT_OUTCHAR structure 5-895
GFX_DRV_S1D13517_PixelArrayPut function 5-329	GFX_FONT_SPACE macro 5-949
GFX_DRV_S1D13517_PixelPut function 5-329	GFX_FontAlignmentSet macro 5-949
GFX_DRV_S1D13517_PixelsPut function 5-329	GFX_FontAntiAliasGet function 5-845
GFX_DRV_S1D13517_SetColor function 5-330	GFX_FontAntiAliasSet function 5-845
GFX_DRV_S1D13517_SetInstance function 5-330	GFX_FontGet function 5-846
GFX_DRV_S1D13517_SetPage function 5-330	GFX_FontSet function 5-846
GFX_DRV_S1D13517_SetReg function 5-331	GFX_FrameBufferGet function 5-846
GFX_DRV_S1D13517_Tasks function 5-331	GFX_FrameBufferSelect function 5-846
GFX_DRV_SSD1926_BarFill function 5-336	GFX_free macro 5-326
GFX_DRV_SSD1926_BrightnessSet function 5-336	gfx_gol.h 5-1072
GFX_DRV_SSD1926_Busy function 5-336	gfx_gol_analog_clock.h 5-1074
GFX_DRV_SSD1926_Close function 5-336	GFX_GOL_ANALOGCLOCK structure 5-896
GFX_DRV_SSD1926_COMMAND structure 5-340	GFX_GOL_ANALOGCLOCK_DISABLED macro 5-949
GFX_DRV_SSD1926_GetReg function 5-337	GFX_GOL_ANALOGCLOCK_DRAW macro 5-950
GFX_DRV_SSD1926_Initialize function 5-337	GFX_GOL_ANALOGCLOCK_HIDE macro 5-950
GFX_DRV_SSD1926_Open function 5-337	GFX_GOL_ANALOGCLOCK_PRESSED macro 5-950

GFX_GOL_ANALOGCLOCK_TICK macro 5-950	GFX_GOL_ChartActionGet function 5-733
GFX_GOL_ANALOGCLOCK_UPDATE_HOUR macro 5-950	GFX_GOL_ChartAxisLabelFontGet macro 5-955
GFX_GOL_ANALOGCLOCK_UPDATE_MINUTE macro 5-950	GFX_GOL_ChartAxisLabelFontSet macro 5-956
GFX_GOL_ANALOGCLOCK_UPDATE_SECOND macro 5-951	GFX_GOL_ChartColorTableGet macro 5-956
GFX_GOL_AnalogClockDraw function 5-719	GFX_GOL_ChartColorTableSet macro 5-957
GFX_GOL_AnalogClockHandsDraw function 5-720	GFX_GOL_ChartCreate function 5-734
GFX_GOL_AnalogClockHourSet function 5-721	GFX_GOL_ChartDataSeriesAdd function 5-736
GFX_GOL_AnalogClockMinuteSet function 5-721	GFX_GOL_ChartDataSeriesFree function 5-736
GFX_GOL_AnalogClockSecondSet function 5-722	GFX_GOL_ChartDataSeriesRemove function 5-737
GFX_GOL_BUTTON structure 5-896	GFX_GOL_ChartDataSeriesSet function 5-737
gfx_gol_button.h 5-1075	GFX_GOL_ChartDraw function 5-738
GFX_GOL_BUTTON_STATE enumeration 5-897	GFX_GOL_ChartGridLabelFontGet macro 5-957
GFX_GOL_ButtonActionGet function 5-722	GFX_GOL_ChartGridLabelFontSet macro 5-958
GFX_GOL_ButtonActionSet function 5-723	GFX_GOL_ChartPercentMaxGet macro 5-958
GFX_GOL_ButtonCreate function 5-724	GFX_GOL_ChartPercentMinGet macro 5-959
GFX_GOL_ButtonDraw function 5-726	GFX_GOL_ChartPercentRangeGet macro 5-959
GFX_GOL_ButtonPressStateImageGet macro 5-951	GFX_GOL_ChartPercentRangeSet function 5-738
GFX_GOL_ButtonPressStateImageSet macro 5-951	GFX_GOL_ChartSampleEndGet macro 5-960
GFX_GOL_ButtonReleaseStateImageGet macro 5-952	GFX_GOL_ChartSampleLabelGet macro 5-960
GFX_GOL_ButtonReleaseStateImageSet macro 5-952	GFX_GOL_ChartSampleLabelSet macro 5-960
GFX_GOL_ButtonTextAlignmentGet function 5-726	GFX_GOL_ChartSampleRangeGet macro 5-961
GFX_GOL_ButtonTextAlignmentSet function 5-727	GFX_GOL_ChartSampleRangeSet function 5-739
GFX_GOL_ButtonTextGet macro 5-953	GFX_GOL_ChartSampleStartGet macro 5-961
GFX_GOL_ButtonTextSet function 5-727	GFX_GOL_ChartSeriesHide macro 5-962
GFX_GOL_CHART structure 5-898	GFX_GOL_ChartShowSeriesCountGet macro 5-962
gfx_gol_chart.h 5-1076	GFX_GOL_ChartShowSeriesSet macro 5-963
GFX_GOL_CHART_3D_ENABLE macro 5-953	GFX_GOL_ChartShowSeriesStatusGet macro 5-963
GFX_GOL_CHART_BAR macro 5-953	GFX_GOL_ChartTitleFontGet macro 5-964
GFX_GOL_CHART_BAR_HOR macro 5-953	GFX_GOL_ChartTitleFontSet macro 5-964
GFX_GOL_CHART_DISABLED macro 5-954	GFX_GOL_ChartTitleGet macro 5-965
GFX_GOL_CHART_DONUT macro 5-954	GFX_GOL_ChartTitleSet macro 5-965
GFX_GOL_CHART_DRAW macro 5-954	GFX_GOL_ChartValueLabelGet macro 5-966
GFX_GOL_CHART_DRAW_DATA macro 5-954	GFX_GOL_ChartValueLabelSet macro 5-966
GFX_GOL_CHART_HIDE macro 5-954	GFX_GOL_ChartValueMaxGet macro 5-967
GFX_GOL_CHART_LEGEND macro 5-954	GFX_GOL_ChartValueMinGet macro 5-967
GFX_GOL_CHART_NUMERIC macro 5-955	GFX_GOL_ChartValueRangeGet macro 5-968
GFX_GOL_CHART_PERCENT macro 5-955	GFX_GOL_ChartValueRangeSet function 5-740
GFX_GOL_CHART_PIE macro 5-955	gfx_gol_check_box.h 5-1080
GFX_GOL_CHART_VALUE macro 5-955	GFX_GOL_CHECKBOX structure 5-898

-
GFX_GOL_CHECKBOX_STATE enumeration 5-899	GFX_GOL_EDITBOX_STATE enumeration 5-1056
GFX_GOL_CheckBoxActionGet function 5-728	GFX_GOL_EditBoxActionGet function 5-747
GFX_GOL_CheckBoxActionSet function 5-729	GFX_GOL_EditBoxActionSet function 5-748
GFX_GOL_CheckBoxCreate function 5-730	GFX_GOL_EditBoxCharAdd function 5-749
GFX_GOL_CheckBoxDraw function 5-731	GFX_GOL_EditBoxCharRemove function 5-749
GFX_GOL_CheckBoxTextAlignmentGet function 5-732	GFX_GOL_EditBoxCreate function 5-750
GFX_GOL_CheckBoxTextAlignmentSet function 5-732	GFX_GOL_EditBoxDraw function 5-751
GFX_GOL_CheckBoxTextGet macro 5-968	GFX_GOL_EditBoxTextAlignmentGet macro 5-1058
GFX_GOL_CheckBoxTextSet function 5-732	GFX_GOL_EditBoxTextAlignmentSet macro 5-1059
gfx_gol_custom_control.h 5-1080	GFX_GOL_EditBoxTextGet macro 5-1059
GFX_GOL_CUSTOMCONTROL structure 5-899	GFX_GOL_EditBoxTextSet function 5-751
GFX_GOL_CUSTOMCONTROL_DISABLED macro 5-969	gfx_gol_font_default.h 5-1083
GFX_GOL_CUSTOMCONTROL_DRAW macro 5-969	GFX_GOL_FONT_DEFAULT_H_FILE macro 5-971
GFX_GOL_CUSTOMCONTROL_DRAW_BAR macro 5-969	gfx_gol_grid.h 5-1083
GFX_GOL_CUSTOMCONTROL_HIDE macro 5-969	gfx_gol_group_box.h 5-1085
GFX_GOL_CustomControlActionGet function 5-740	GFX_GOL_GROUPBOX structure 5-902
GFX_GOL_CustomControlActionSet function 5-741	GFX_GOL_GROUPBOX_STATE enumeration 5-903
GFX_GOL_CustomControlCreate function 5-741	GFX_GOL_GroupboxActionGet function 5-759
GFX_GOL_CustomControlDraw function 5-742	GFX_GOL_GroupboxCreate function 5-760
GFX_GOL_CustomControlGetPos macro 5-969	GFX_GOL_GroupboxDraw function 5-761
GFX_GOL_CustomControlSetPos macro 5-970	GFX_GOL_GroupboxTextAlignmentGet function 5-761
gfx_gol_digital_meter.h 5-1081	GFX_GOL_GroupboxTextAlignmentSet function 5-762
GFX_GOL_DIGITALMETER structure 5-900	GFX_GOL_GroupboxTextGet macro 5-971
GFX_GOL_DIGITALMETER_INDENT macro 5-970	GFX_GOL_GroupboxTextSet function 5-762
GFX_GOL_DIGITALMETER_STATE enumeration 5-901	gfx_gol_list_box.h 5-1086
GFX_GOL_DIGITALMETER_WIDTH macro 5-970	GFX_GOL_LISTBOX structure 5-903
GFX_GOL_DigitalMeterActionGet function 5-742	GFX_GOL_LISTBOX_ITEM_STATUS enumeration 5-903
GFX_GOL_DigitalMeterCreate function 5-743	GFX_GOL_LISTBOX_STATE enumeration 5-904
GFX_GOL_DigitalMeterDecrement function 5-744	GFX_GOL_ListBoxActionGet function 5-763
GFX_GOL_DigitalMeterDraw function 5-745	GFX_GOL_ListBoxActionSet function 5-764
GFX_GOL_DigitalMeterIncrement function 5-745	GFX_GOL_ListBoxCreate function 5-765
GFX_GOL_DigitalMeterTextAlignmentGet macro 5-1057	GFX_GOL_ListBoxDraw function 5-766
GFX_GOL_DigitalMeterTextAlignmentSet macro 5-1058	GFX_GOL_ListBoxItemAdd function 5-767
GFX_GOL_DigitalMeterValueGet macro 5-970	GFX_GOL_ListBoxItemCountGet macro 5-972
GFX_GOL_DigitalMeterValueSet function 5-746	GFX_GOL_ListBoxItemFocusGet function 5-768
GFX_GOL_DRAW_CALLBACK_FUNC type 5-901	GFX_GOL_ListBoxItemFocusSet function 5-769
GFX_GOL_DrawCallbackSet function 5-702	GFX_GOL_ListBoxItemImageGet macro 5-972
gfx_gol_edit_box.h 5-1082	GFX_GOL_ListBoxItemImageSet macro 5-973
GFX_GOL_EDITBOX structure 5-902	GFX_GOL_ListBoxItemListGet macro 5-973
-

GFX_GOL_ListBoxItemListRemove function 5-769	GFX_GOL_ObjectDrawDisable function 5-705
GFX_GOL_ListBoxItemRemove function 5-770	GFX_GOL_ObjectDrawEnable function 5-706
GFX_GOL_ListBoxItemSelectStatusClear macro 5-974	GFX_GOL_ObjectFind function 5-707
GFX_GOL_ListBoxItemSelectStatusSet macro 5-974	GFX_GOL_ObjectFocusGet function 5-707
GFX_GOL_ListBoxSelectionChange function 5-770	GFX_GOL_ObjectFocusNextGet function 5-708
GFX_GOL_ListBoxSelectionGet function 5-771	GFX_GOL_ObjectFocusPrevGet function 5-708
GFX_GOL_ListBoxTextAlignmentGet macro 5-1060	GFX_GOL_ObjectFocusSet function 5-709
GFX_GOL_ListBoxTextAlignmentSet macro 5-1060	GFX_GOL_ObjectHideDraw function 5-709
GFX_GOL_ListBoxVisibleItemCountGet function 5-772	GFX_GOL_ObjectIDGet function 5-709
GFX_GOL_LISTITEM structure 5-904	GFX_GOL_ObjectIsRedrawSet function 5-710
GFX_GOL_MESSAGE structure 5-905	GFX_GOL_ObjectListDraw function 5-711
GFX_GOL_MessageCallbackSet function 5-702	GFX_GOL_ObjectListFree function 5-711
GFX_GOL_METER structure 5-906	GFX_GOL_ObjectListGet function 5-712
gfx_gol_meter.h 5-1087	GFX_GOL_ObjectListNew function 5-712
GFX_GOL_METER_DRAW_TYPE enumeration 5-907	GFX_GOL_ObjectListSet function 5-713
GFX_GOL_METER_STATE enumeration 5-907	GFX_GOL_ObjectMessage function 5-714
GFX_GOL_MeterActionGet function 5-772	GFX_GOL_ObjectNextGet function 5-715
GFX_GOL_MeterActionSet function 5-773	GFX_GOL_ObjectRectangleRedraw function 5-715
GFX_GOL_MeterCreate function 5-773	GFX_GOL_ObjectStateClear macro 5-978
GFX_GOL_MeterDecrement function 5-775	GFX_GOL_ObjectStateGet macro 5-978
GFX_GOL_MeterDraw function 5-776	GFX_GOL_ObjectStateSet macro 5-979
GFX_GOL_MeterIncrement function 5-776	GFX_GOL_ObjectTypeGet function 5-716
GFX_GOL_MeterMaximumValueGet macro 5-975	GFX_GOL_PanelAlphaParameterSet function 5-717
GFX_GOL_MeterMinimumValueGet macro 5-975	GFX_GOL_PanelBackgroundSet function 5-717
GFX_GOL_MeterRangeSet function 5-777	GFX_GOL_PanelDraw function 5-717
GFX_GOL_MeterScaleColorsSet function 5-777	GFX_GOL_PanelGradientParameterSet function 5-717
GFX_GOL_MeterTitleFontSet macro 5-976	GFX_GOL_PanelParameterSet function 5-717
GFX_GOL_MeterTypeSet macro 5-976	gfx_gol_picture.h 5-1087
GFX_GOL_MeterValueFontSet macro 5-976	GFX_GOL_PICTURECONTROL structure 5-1056
GFX_GOL_MeterValueGet macro 5-977	GFX_GOL_PictureControlActionGet function 5-783
GFX_GOL_MeterValueSet function 5-778	GFX_GOL_PICTURECONTROLCONTROL_STATE enumeration 5-1057
GFX_GOL_OBJ_HEADER structure 5-908	GFX_GOL_PictureControlCreate function 5-784
GFX_GOL_OBJ_SCHEME structure 5-908	GFX_GOL_PictureControlDraw function 5-785
GFX_GOL_OBJ_TYPE enumeration 5-910	GFX_GOL_PictureControlImageGet macro 5-1061
GFX_GOL_ObjectAdd function 5-703	GFX_GOL_PictureControlImageSet macro 5-1061
GFX_GOL_ObjectBackGroundSet function 5-704	GFX_GOL_PictureControlPartialSet function 5-785
GFX_GOL_ObjectByIDDelete function 5-704	GFX_GOL_PictureControlScaleSet macro 5-1062
GFX_GOL_ObjectCanBeFocused function 5-704	gfx_gol_progress_bar.h 5-1088
GFX_GOL_ObjectDelete function 5-705	GFX_GOL_PROGRESSBAR structure 5-911

GFX_GOL_PROGRESSBAR_STATE enumeration 5-911	gfx_gol_scheme.h 5-1092
GFX_GOL_ProgressBarActionGet function 5-779	GFX_GOL_SchemeAlphaPrecentSet function 5-878
GFX_GOL_ProgressBarCreate function 5-780	GFX_GOL_SchemeBackgroundColorGet function 5-878
GFX_GOL_ProgressBarDraw function 5-781	GFX_GOL_SchemeBackgroundColorSet function 5-878
GFX_GOL_ProgressBarPositionGet macro 5-980	GFX_GOL_SchemeBackgroundImageSet function 5-879
GFX_GOL_ProgressBarPositionSet function 5-782	GFX_GOL_SchemeBackgroundTypeGet function 5-879
GFX_GOL_ProgressBarRangeGet macro 5-980	GFX_GOL_SchemeBackgroundTypeSet function 5-879
GFX_GOL_ProgressBarRangeSet function 5-782	GFX_GOL_SchemeColor0Get function 5-879
gfx_gol_radio_button.h 5-1089	GFX_GOL_SchemeColor0Set function 5-879
GFX_GOL_RADIOBUTTON structure 5-912	GFX_GOL_SchemeColor1Get function 5-880
GFX_GOL_RADIOBUTTON_STATE enumeration 5-912	GFX_GOL_SchemeColor1Set function 5-880
GFX_GOL_RadioButtonActionGet function 5-787	GFX_GOL_SchemeColorDisabledGet function 5-880
GFX_GOL_RadioButtonActionSet function 5-788	GFX_GOL_SchemeColorDisabledSet function 5-880
GFX_GOL_RadioButtonCheckGet function 5-788	GFX_GOL_SchemeColorSet function 5-880
GFX_GOL_RadioButtonCheckSet function 5-790	GFX_GOL_SchemeEmbossDarkColorGet function 5-881
GFX_GOL_RadioButtonCreate function 5-790	GFX_GOL_SchemeEmbossDarkColorSet function 5-881
GFX_GOL_RadioButtonDraw function 5-792	GFX_GOL_SchemeEmbossLightColorGet function 5-881
GFX_GOL_RadioButtonTextAlignmentGet function 5-792	GFX_GOL_SchemeEmbossLightColorSet function 5-881
GFX_GOL_RadioButtonTextAlignmentSet function 5-793	GFX_GOL_SchemeEmbossSet function 5-881
GFX_GOL_RadioButtonTextGet macro 5-980	GFX_GOL_SchemeEmbossSizeGet function 5-882
GFX_GOL_RadioButtonTextSet function 5-793	GFX_GOL_SchemeEmbossSizeSet function 5-882
gfx_gol_round_dial.h 5-1090	GFX_GOL_SchemeFillStyleGet function 5-882
GFX_GOL_RoundDialActionGet function 5-794	GFX_GOL_SchemeFillStyleSet function 5-882
GFX_GOL_RoundDialActionSet function 5-795	GFX_GOL_SchemeFontGet function 5-882
GFX_GOL_ROUNDSDIAL structure 5-913	GFX_GOL_SchemeFontSet function 5-883
GFX_GOL_ROUNDSDIAL_DISABLED macro 5-981	GFX_GOL_SchemeGradientColorSet function 5-883
GFX_GOL_ROUNDSDIAL_DRAW macro 5-981	GFX_GOL_SchemeGradientEndColorGet function 5-883
GFX_GOL_ROUNDSDIAL_HIDE macro 5-981	GFX_GOL_SchemeGradientEndColorSet function 5-883
GFX_GOL_ROUNDSDIAL_MAX_POSITIONS macro 5-981	GFX_GOL_SchemeGradientStartColorGet function 5-883
GFX_GOL_ROUNDSDIAL_QUADRANT_POSITIONS macro 5-981	GFX_GOL_SchemeGradientStartColorSet function 5-884
GFX_GOL_ROUNDSDIAL_ROT_CCW macro 5-982	GFX_GOL_SchemeTextColor0Get function 5-884
GFX_GOL_ROUNDSDIAL_ROT_CW macro 5-982	GFX_GOL_SchemeTextColor0Set function 5-884
GFX_GOL_RoundDialCreate function 5-796	GFX_GOL_SchemeTextColor1Get function 5-884
GFX_GOL_RoundDialDraw function 5-797	GFX_GOL_SchemeTextColor1Set function 5-884
GFX_GOL_RoundDialValueDecrement macro 5-982	GFX_GOL_SchemeTextColorDisableGet function 5-885
GFX_GOL_RoundDialValueGet macro 5-982	GFX_GOL_SchemeTextColorDisableSet function 5-885
GFX_GOL_RoundDialValueIncrement macro 5-983	GFX_GOL_SchemeTextColorSet function 5-885
GFX_GOL_RoundDialValueSet macro 5-983	gfx_gol_scroll_bar.h 5-1093
gfx_gol_scan_codes.h 5-1091	GFX_GOL_SCROLLBAR structure 5-914

- GFX_GOL_SCROLLBAR_STATE enumeration 5-914
- GFX_GOL_ScrollBarActionGet function 5-798
- GFX_GOL_ScrollBarActionSet function 5-799
- GFX_GOL_ScrollBarCreate function 5-800
- GFX_GOL_ScrollBarDraw function 5-802
- GFX_GOL_ScrollBarPageGet function 5-802
- GFX_GOL_ScrollBarPageSet function 5-803
- GFX_GOL_ScrollBarPositionDecrement function 5-804
- GFX_GOL_ScrollBarPositionGet function 5-804
- GFX_GOL_ScrollBarPositionIncrement function 5-805
- GFX_GOL_ScrollBarPositionSet function 5-806
- GFX_GOL_ScrollBarRangeGet function 5-807
- GFX_GOL_ScrollBarRangeSet function 5-807
- gfx_gol_static_text.h 5-1094
- GFX_GOL_STATICTEXT structure 5-915
- GFX_GOL_STATICTEXT_STATE enumeration 5-915
- GFX_GOL_StaticTextActionGet function 5-808
- GFX_GOL_StaticTextAlignmentGet function 5-809
- GFX_GOL_StaticTextAlignmentSet function 5-809
- GFX_GOL_StaticTextCreate function 5-810
- GFX_GOL_StaticTextDraw function 5-811
- GFX_GOL_StaticTextGet function 5-811
- GFX_GOL_StaticTextSet function 5-812
- gfx_gol_text_entry.h 5-1095
- GFX_GOL_TEXTENTRY structure 5-916
- GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE enumeration 5-916
- GFX_GOL_TEXTENTRY_STATE enumeration 5-917
- GFX_GOL_TextEntryActionGet function 5-812
- GFX_GOL_TextEntryActionSet function 5-814
- GFX_GOL_TextEntryBufferClear function 5-815
- GFX_GOL_TextEntryBufferGet function 5-815
- GFX_GOL_TextEntryBufferSet function 5-816
- GFX_GOL_TextEntryCharAdd function 5-816
- GFX_GOL_TextEntryCreate function 5-817
- GFX_GOL_TextEntryDraw function 5-818
- GFX_GOL_TextEntryKeyCommandGet function 5-819
- GFX_GOL_TextEntryKeyCommandSet function 5-819
- GFX_GOL_TextEntryKeysPressed function 5-820
- GFX_GOL_TextEntryKeyListCreate function 5-820
- GFX_GOL_TextEntryKeyMemberListDelete function 5-821
- GFX_GOL_TextEntryKeyTextSet function 5-821
- GFX_GOL_TextEntryLastCharDelete function 5-822
- GFX_GOL_TextEntrySpaceCharAdd function 5-822
- GFX_GOL_TRANSLATED_ACTION enumeration 5-917
- GFX_GOL_TRANSLATED_ACTION, \GFX_GOL_OBJ_HEADER *, \GFX_GOL_MESSAGE * \) type 5-919
- GFX_GOL_TwoTonePanelDraw function 5-718
- GFX_GOL_WINDOW structure 5-920
- gfx_gol_window.h 5-1096
- GFX_GOL_WINDOW_STATE enumeration 5-920
- GFX_GOL_WindowActionGet function 5-823
- GFX_GOL_WindowCreate function 5-823
- GFX_GOL_WindowDraw function 5-824
- GFX_GOL_WindowImageGet macro 5-984
- GFX_GOL_WindowTextAlignmentGet function 5-825
- GFX_GOL_WindowTextAlignmentSet function 5-825
- GFX_GOL_WindowTextGet macro 5-984
- GFX_GOL_WindowTextSet function 5-826
- GFX_GRADIENT_STYLE structure 5-921
- GFX_GradientColorSet function 5-847
- GFX_GradientEndColorGet function 5-847
- GFX_GradientStartColorGet function 5-847
- GFX_GradientTypeSet macro 5-1125
- gfx_imageDecoder.h 5-1096
- GFX_ImageDraw function 5-848
- GFX_ImageHeaderGet function 5-848
- GFX_ImageHeightGet function 5-849
- GFX_ImageOffsetAddressGet function 5-849
- GFX_ImagePartialDraw function 5-850
- GFX_ImageStretchSet macro 5-985
- GFX_ImageWidthGet function 5-851
- GFX_Initialize function 5-852
- GFX_Layer function 5-852
- GFX_LAYER_PARAMS structure 5-295
- GFX_LINE_STYLE enumeration 5-921
- GFX_LineDraw function 5-853
- GFX_LinePositionRelativeSet function 5-853
- GFX_LinePositionSet function 5-854

GFX_LinePositionXGet function 5-854	GFX_RESOURCE_HDR structure 5-927
GFX_LinePositionYGet function 5-855	GFX_RESOURCE_IMAGE structure 5-928
GFX_LineStyleGet function 5-855	GFX_RESOURCE_PALETTE structure 5-929
GFX_LineStyleSet function 5-855	GFX_RGBConvert macro 5-987
GFX_LineToDraw function 5-856	GFX_ScreenClear function 5-863
GFX_LineToRelativeDraw function 5-856	GFX_SetFontOrientation macro 5-987
GFX_malloc macro 5-326	GFX_SetTransitionParameters function 5-863
GFX_MAX_INVALIDATE_AREAS macro 5-985	GFX_SineCosineGet function 5-1124
GFX_MaxXGet macro 5-985	GFX_SineGet macro 5-1125
GFX_MaxYGet macro 5-985	GFX_SolidLineDraw function 5-864
GFX_MCHP_BITMAP_HEADER structure 5-922	GFX_STATUS enumeration 5-929
GFX_MEM_MASK macro 5-986	GFX_STATUS_BIT enumeration 5-929
GFX_PageSet function 5-857	GFX_StyledLineDraw function 5-864
gfx_palette.h 5-1098	GFX_TextAreaBottomGet function 5-864
GFX_PALETTE_ENTRY union 5-922	GFX_TextAreaBottomSet function 5-864
GFX_PARTIAL_IMAGE_PARAM structure 5-923	GFX_TextAreaLeftGet function 5-865
GFX_PixelPut macro 5-986	GFX_TextAreaLeftSet function 5-865
GFX_PixelsPut macro 5-986	GFX_TextAreaRightGet function 5-865
GFX_PolygonDraw function 5-857	GFX_TextAreaRightSet function 5-865
GFX_PrevAlphaColorGet macro 5-986	GFX_TextAreaTopGet function 5-865
GFX_PrevAlphaColorSet macro 5-986	GFX_TextAreaTopSet function 5-865
GFX_PRIM_SetPIPWindow function 5-305	GFX_TextCharDraw function 5-866
gfx_primitive.h 5-1100	GFX_TextCursorPositionSet function 5-866
GFX_Primitive_DATA structure 5-923	GFX_TextCursorPositionXGet function 5-867
GFX_Primitive_Initialize function 5-858	GFX_TextCursorPositionYGet function 5-867
GFX_Primitive_instance variable 5-888	GFX_TextStringBoxDraw function 5-868
gfx_primitive_local.h 5-1104	GFX_TextStringDraw function 5-869
GFX_RectangleDraw function 5-858	GFX_TextStringHeightGet function 5-869
GFX_RectangleFillDraw function 5-859	GFX_TextStringWidthGet function 5-870
GFX_RectangleRoundDraw function 5-860	GFX_ThickBevelDraw function 5-871
GFX_RectangleRoundFillDraw function 5-861	GFX_Transition function 5-871
GFX_RECTANGULAR_AREA structure 5-924	GFX_TRANSITION_DIRECTION enumeration 5-930
GFX_RenderingBufferGet function 5-862	GFX_TRANSITION_PARAMS structure 5-930
GFX_RenderToDisplayBufferDisable function 5-862	GFX_TRANSITION_TYPE enumeration 5-930
GFX_RenderToDisplayBufferDisableFlagGet function 5-862	gfx_transitions.h 5-1105
GFX_RenderToDisplayBufferEnable function 5-862	GFX_TransparentColorDisable function 5-871
GFX_RESOURCE enumeration 5-924	GFX_TransparentColorEnable function 5-872
GFX_RESOURCE_BINARY structure 5-926	GFX_TransparentColorGet function 5-873
GFX_RESOURCE_FONT structure 5-927	GFX_TransparentColorStatusGet function 5-873

GFX_TRIG_FUNCTION_TYPE enumeration 5-931	GFX_W3_DARKGREY macro 5-993
GFX_TYPE_MASK macro 5-988	GFX_W3_DARKHAKI macro 5-993
gfx_types_font.h 5-1107	GFX_W3_DARKMAGENTA macro 5-993
GFX_TYPES_FONTS_H macro 5-988	GFX_W3_DARKOLIVEGREEN macro 5-993
gfx_types_image.h 5-1107	GFX_W3_DARKORANGE macro 5-993
GFX_TYPES_IMAGE_H macro 5-988	GFX_W3_DARKORCHID macro 5-994
gfx_types_macros.h 5-1108	GFX_W3_DARKRED macro 5-994
gfx_types_palette.h 5-1109	GFX_W3_DARKSALMON macro 5-994
GFX_TYPES_PALETTE_H macro 5-988	GFX_W3_DARKSEAGREEN macro 5-994
gfx_types_resource.h 5-1109	GFX_W3_DARKSLATEBLUE macro 5-994
GFX_TYPES_RESOURCE_H macro 5-988	GFX_W3_DARKSLATEGRAY macro 5-994
GFX_UXCHAR macro 5-988	GFX_W3_DARKSLATEGREY macro 5-994
GFX_VisualPageGet macro 5-989	GFX_W3_DARKTURQUOISE macro 5-995
GFX_W3_ALICEBLUE macro 5-989	GFX_W3_DARKVIOLET macro 5-995
GFX_W3_ANTIQUWHITE macro 5-989	GFX_W3_DEEPPINK macro 5-995
GFX_W3_AQUA macro 5-989	GFX_W3_DEEPSKYBLUE macro 5-995
GFX_W3_AQUAMARINE macro 5-989	GFX_W3_DIMGRAY macro 5-995
GFX_W3_AZURE macro 5-989	GFX_W3_DIMGREY macro 5-995
GFX_W3_BEIGE macro 5-990	GFX_W3_DODGERBLUE macro 5-996
GFX_W3_BISQUE macro 5-990	GFX_W3_FIREBRICK macro 5-996
GFX_W3_BLACK macro 5-990	GFX_W3_FLORALWHITE macro 5-996
GFX_W3_BLANCHEDALMOND macro 5-990	GFX_W3_FORESTGREEN macro 5-996
GFX_W3_BLUE macro 5-990	GFX_W3_FUCHSIA macro 5-996
GFX_W3_BLUEVIOLET macro 5-990	GFX_W3_GAINSBORO macro 5-996
GFX_W3_BROWN macro 5-990	GFX_W3_GHOSTWHITE macro 5-996
GFX_W3_BURLYWOOD macro 5-991	GFX_W3_GOLD macro 5-997
GFX_W3_CADETBLUE macro 5-991	GFX_W3_GOLDENROD macro 5-997
GFX_W3_CHARTREUSE macro 5-991	GFX_W3_GRAY macro 5-997
GFX_W3_CHOCOLATE macro 5-991	GFX_W3_GREEN macro 5-997
GFX_W3_CORAL macro 5-991	GFX_W3_GREENYELLOW macro 5-997
GFX_W3_CORNFLOWERBLUE macro 5-991	GFX_W3_GREY macro 5-997
GFX_W3_CORNSILK macro 5-992	GFX_W3_HONEYDEW macro 5-998
GFX_W3_CRIMSON macro 5-992	GFX_W3_HOTPINK macro 5-998
GFX_W3_CYAN macro 5-992	GFX_W3_INDIANRED macro 5-998
GFX_W3_DARKBLUE macro 5-992	GFX_W3_INDIGO macro 5-998
GFX_W3_DARKCYAN macro 5-992	GFX_W3_IVORY macro 5-998
GFX_W3_darkgoldenrod macro 5-992	GFX_W3_LAVENDER macro 5-998
GFX_W3_DARKGRAY macro 5-992	GFX_W3_LAVENDERBLUSH macro 5-998
GFX_W3_DARKGREEN macro 5-993	GFX_W3_LAWNGREEN macro 5-999

GFX_W3_LEMONCHIFFON macro 5-999	GFX_W3_ORANGE macro 5-1005
GFX_W3_LIGHTBLUE macro 5-999	GFX_W3_ORANGERED macro 5-1005
GFX_W3_LIGHTCORAL macro 5-999	GFX_W3_ORCHID macro 5-1005
GFX_W3_LIGHTCYAN macro 5-999	GFX_W3_PALEGOLDENROD macro 5-1005
GFX_W3_LIGHTGOLDENRODYELLOW macro 5-999	GFX_W3_PALEGREEN macro 5-1005
GFX_W3_LIGHTGRAY macro 5-1000	GFX_W3_PALETURQUOISE macro 5-1006
GFX_W3_LIGHTGREEN macro 5-1000	GFX_W3_PALEVIOLETRED macro 5-1006
GFX_W3_LIGHTGREY macro 5-1000	GFX_W3_PAPAYAWHIP macro 5-1006
GFX_W3_LIGHTPINK macro 5-1000	GFX_W3_PEACHPUFF macro 5-1006
GFX_W3_LIGHTSALMON macro 5-1000	GFX_W3_PERU macro 5-1006
GFX_W3_LIGHTSKYBLUE macro 5-1000	GFX_W3_PINK macro 5-1006
GFX_W3_LIGHTSLATEGRAY macro 5-1000	GFX_W3_PLUM macro 5-1006
GFX_W3_LIGHTSLATEGREY macro 5-1001	GFX_W3_POWDERBLUE macro 5-1007
GFX_W3_LIGHTSTEELBLUE macro 5-1001	GFX_W3_PURPLE macro 5-1007
GFX_W3_LIGHTYELLOW macro 5-1001	GFX_W3_RED macro 5-1007
GFX_W3_LIGHSEAGREEN macro 5-1001	GFX_W3_ROSYBROWN macro 5-1007
GFX_W3_LIME macro 5-1001	GFX_W3_ROYALBLUE macro 5-1007
GFX_W3_LIMEGREEN macro 5-1001	GFX_W3_SADDLEBROWN macro 5-1007
GFX_W3_LINEN macro 5-1002	GFX_W3_SALMON macro 5-1008
GFX_W3_MAGENTA macro 5-1002	GFX_W3_SANDYGREEN macro 5-1008
GFX_W3_MAROON macro 5-1002	GFX_W3_SEAGREEN macro 5-1008
GFX_W3_MEDIUMAQUAMARINE macro 5-1002	GFX_W3_SEASHELL macro 5-1008
GFX_W3_MEDIUMBLUE macro 5-1002	GFX_W3_SIENNA macro 5-1008
GFX_W3_MEDIUMORCHID macro 5-1002	GFX_W3_SILVER macro 5-1008
GFX_W3_MEDIUMPURPLE macro 5-1002	GFX_W3_SKYBLUE macro 5-1008
GFX_W3_MEDIUMSEAGREEN macro 5-1003	GFX_W3_SLATEBLUE macro 5-1009
GFX_W3_MEDIUMSLATEBLUE macro 5-1003	GFX_W3_SLATEGRAY macro 5-1009
GFX_W3_MEDIUMSPRINGGREEN macro 5-1003	GFX_W3_SLATEGREY macro 5-1009
GFX_W3_MEDIUMTURQUOISE macro 5-1003	GFX_W3_SNOW macro 5-1009
GFX_W3_MEDIUMVIOLETRED macro 5-1003	GFX_W3_SPRINGGREEN macro 5-1009
GFX_W3_MIDNIGHTBLUE macro 5-1003	GFX_W3_STEELBLUE macro 5-1009
GFX_W3_MINTCREAM macro 5-1004	GFX_W3_TAN macro 5-1010
GFX_W3_MISTYROSE macro 5-1004	GFX_W3_TEAL macro 5-1010
GFX_W3_MOCCASIN macro 5-1004	GFX_W3_THISTLE macro 5-1010
GFX_W3_NAVAJOWHITE macro 5-1004	GFX_W3_TOMATO macro 5-1010
GFX_W3_NAVY macro 5-1004	GFX_W3_TURQUOISE macro 5-1010
GFX_W3_OLDLACE macro 5-1004	GFX_W3_VIOLET macro 5-1010
GFX_W3_OLIVE macro 5-1004	GFX_W3_WHEAT macro 5-1010
GFX_W3_OLIVEDRAB macro 5-1005	GFX_W3_WHITE macro 5-1011

GFX_W3_WHITESMOKE macro 5-1011	GFX_X11_DARK_TURQUOISE macro 5-1017
GFX_W3_YELLOW macro 5-1011	GFX_X11_DARK_VIOLET macro 5-1017
GFX_W3_YELLOWGREEN macro 5-1011	GFX_X11_DEEP_PINK macro 5-1017
GFX_X11_ALICE_BLUE macro 5-1011	GFX_X11_DEEP_SKY_BLUE macro 5-1017
GFX_X11_ANTIQUUE_WHITE macro 5-1011	GFX_X11_DIM_GREY macro 5-1017
GFX_X11_AQUA macro 5-1012	GFX_X11_DODGER_BLUE macro 5-1018
GFX_X11_AQUAMARINE macro 5-1012	GFX_X11_FIRE_BRICK macro 5-1018
GFX_X11_AZURE macro 5-1012	GFX_X11_FLORAL_WHITE macro 5-1018
GFX_X11_BEIGE macro 5-1012	GFX_X11_FOREST_GREEN macro 5-1018
GFX_X11_BISQUE macro 5-1012	GFX_X11_FUCHSIA macro 5-1018
GFX_X11_BLACK macro 5-1012	GFX_X11_GAINSBORO macro 5-1018
GFX_X11_BLANCHED_ALMOND macro 5-1012	GFX_X11_GHOST_WHITE macro 5-1018
GFX_X11_BLUE macro 5-1013	GFX_X11_GOLD macro 5-1019
GFX_X11_BLUE_VIOLET macro 5-1013	GFX_X11_GOLDENROD macro 5-1019
GFX_X11_BROWN macro 5-1013	GFX_X11_GREEN macro 5-1019
GFX_X11_BURLY_WOOD macro 5-1013	GFX_X11_GREEN_YELLOW macro 5-1019
GFX_X11_CADEL_BLUE macro 5-1013	GFX_X11_GREY macro 5-1019
GFX_X11_CHARTEUSE macro 5-1013	GFX_X11_HONEYDEW macro 5-1019
GFX_X11_CHOCOLATE macro 5-1014	GFX_X11_HOT_PINK macro 5-1020
GFX_X11_CORAL macro 5-1014	GFX_X11_INDIAN_RED macro 5-1020
GFX_X11_CORNFLOWER_BLUE macro 5-1014	GFX_X11_INDIGO macro 5-1020
GFX_X11_CORNSILK macro 5-1014	GFX_X11_IVORY macro 5-1020
GFX_X11_CRIMSON macro 5-1014	GFX_X11_KHAKI macro 5-1020
GFX_X11_CYAN macro 5-1014	GFX_X11_LAVENDER macro 5-1020
GFX_X11_DARK_BLUE macro 5-1014	GFX_X11_LAVENDOR_BLUSH macro 5-1020
GFX_X11_DARK_CYAN macro 5-1015	GFX_X11_LAWN_GREEN macro 5-1021
GFX_X11_DARK_GOLDENROD macro 5-1015	GFX_X11_LEMON_CHIFFON macro 5-1021
GFX_X11_DARK_GREEN macro 5-1015	GFX_X11_LIGHT_BLUE macro 5-1021
GFX_X11_DARK_GREY macro 5-1015	GFX_X11_LIGHT_CAROL macro 5-1021
GFX_X11_DARK_KHAKI macro 5-1015	GFX_X11_LIGHT_CYAN macro 5-1021
GFX_X11_DARK_MAGENTA macro 5-1015	GFX_X11_LIGHT_GOLDENROD_YELLOW macro 5-1021
GFX_X11_DARK_OLIVE_GREEN macro 5-1016	GFX_X11_LIGHT_GRAY macro 5-1022
GFX_X11_DARK_ORANGE macro 5-1016	GFX_X11_LIGHT_GREEN macro 5-1022
GFX_X11_DARK_ORCHID macro 5-1016	GFX_X11_LIGHT_PINK macro 5-1022
GFX_X11_DARK_RED macro 5-1016	GFX_X11_LIGHT_SALMON macro 5-1022
GFX_X11_DARK_SALMON macro 5-1016	GFX_X11_LIGHT_SEA_GREEN macro 5-1022
GFX_X11_DARK_SEA_GREEN macro 5-1016	GFX_X11_LIGHT_SKY_BLUE macro 5-1022
GFX_X11_DARK_SLATE_BLUE macro 5-1016	GFX_X11_LIGHT_SLATE_GREY macro 5-1022
GFX_X11_DARK_SLATE_GREY macro 5-1017	GFX_X11_LIGHT_STEEL_BLUE macro 5-1023

GFX_X11_LIGHT_YELLOW macro 5-1023	GFX_X11_ROSY_BROWN macro 5-1029
GFX_X11_LIME macro 5-1023	GFX_X11_ROYAL_BLUE macro 5-1029
GFX_X11_LIME_GREEN macro 5-1023	GFX_X11_SADDLE_BROWN macro 5-1029
GFX_X11_LINEN macro 5-1023	GFX_X11_SALMON macro 5-1029
GFX_X11_MAGENTA macro 5-1023	GFX_X11_SANDY_BROWN macro 5-1029
GFX_X11_MARRON macro 5-1024	GFX_X11_SEA_GREEN macro 5-1030
GFX_X11_MEDIUM_AQUAMARINE macro 5-1024	GFX_X11_SEASHELL macro 5-1030
GFX_X11_MEDIUM_BLUE macro 5-1024	GFX_X11_SIENNA macro 5-1030
GFX_X11_MEDIUM_ORCHID macro 5-1024	GFX_X11_SILVER macro 5-1030
GFX_X11_MEDIUM_PURPLE macro 5-1024	GFX_X11_SKY_BLUE macro 5-1030
GFX_X11_MEDIUM_SEA_GREEN macro 5-1024	GFX_X11_SLATE_BLUE macro 5-1030
GFX_X11_MEDIUM_SLATE_BLUE macro 5-1024	GFX_X11_SLATE_GREY macro 5-1030
GFX_X11_MEDIUM_SPRING_GREEN macro 5-1025	GFX_X11_SNOW macro 5-1031
GFX_X11_MEDIUM_TURQUOISE macro 5-1025	GFX_X11_SPRING_GREEN macro 5-1031
GFX_X11_MEDIUM_VIOLET_RED macro 5-1025	GFX_X11_STEEL_BLUE macro 5-1031
GFX_X11_MIDNIGHT_BLUE macro 5-1025	GFX_X11_TAN macro 5-1031
GFX_X11_MINT_CREAM macro 5-1025	GFX_X11_TEAL macro 5-1031
GFX_X11_MISTY_ROSE macro 5-1025	GFX_X11_THISTLE macro 5-1031
GFX_X11_MOCCASIN macro 5-1026	GFX_X11_TOMATO macro 5-1032
GFX_X11_NAVAJO_WHITE macro 5-1026	GFX_X11_TURQUOISE macro 5-1032
GFX_X11_NAVY macro 5-1026	GFX_X11_VIOLET macro 5-1032
GFX_X11_OLD_LACE macro 5-1026	GFX_X11_WHEAT macro 5-1032
GFX_X11_OLIVE macro 5-1026	GFX_X11_WHITE macro 5-1032
GFX_X11_OLIVE_DRAB macro 5-1026	GFX_X11_WHITE_SMOKE macro 5-1032
GFX_X11_ORANGE macro 5-1026	GFX_X11_YELLOW macro 5-1032
GFX_X11_ORANGE_RED macro 5-1027	GFX_X11_YELLOW_GREEN macro 5-1033
GFX_X11_ORCHID macro 5-1027	GFX_XCHAR macro 5-1033
GFX_X11_PALE_GOLDENROD macro 5-1027	GFXCirclePointGet function 5-1125
GFX_X11_PALE_GREEN macro 5-1027	GOL_PANEL_PARAM structure 5-932
GFX_X11_PALE_TURQUOISE macro 5-1027	GOLD macro 5-1033
GFX_X11_PALE_VIOLET_RED macro 5-1027	Gradients 5-1114
GFX_X11_PAPAYA_WHIP macro 5-1028	Graphic Display Driver Layer Changes 5-683
GFX_X11_PEACH_PUFF macro 5-1028	Graphic Object Layer (GOL) Changes 5-683
GFX_X11_PERU macro 5-1028	Graphic Primitive Layer Changes 5-682
GFX_X11_PINK macro 5-1028	Graphics (GFX) Driver Library 5-289
GFX_X11_PLUM macro 5-1028	Graphics Demonstrations 3-84
GFX_X11_POWDER_BLUE macro 5-1028	Graphics Library Help 5-680
GFX_X11_PURPLE macro 5-1028	Graphics Library Porting Guide 5-680
GFX_X11_RED macro 5-1029	Graphics Object Library 5-694

Graphics Primitive Library 5-1110
 GRAPHICS_LIBRARY_VERSION macro 5-1033
 GRAY000 macro 5-1034
 GRAY001 macro 5-1034
 GRAY002 macro 5-1034
 GRAY003 macro 5-1034
 GRAY004 macro 5-1034
 GRAY005 macro 5-1034
 GRAY006 macro 5-1034
 GRAY007 macro 5-1035
 GRAY008 macro 5-1035
 GRAY009 macro 5-1035
 GRAY010 macro 5-1035
 GRAY011 macro 5-1035
 GRAY012 macro 5-1035
 GRAY013 macro 5-1036
 GRAY014 macro 5-1036
 GRAY015 macro 5-1036
 GRAY032 macro 5-1036
 GRAY096 macro 5-1036
 GRAY128 macro 5-1036
 GRAY160 macro 5-1036
 GRAY192 macro 5-1037
 GRAY204 macro 5-1037
 GRAY224 macro 5-1037
 GRAY229 macro 5-1037
 GRAY242 macro 5-1037
 GREEN macro 5-1037
 GRID structure 5-932
 GRID_DISABLED macro 5-1038
 GRID_DRAW_ALL macro 5-1038
 GRID_DRAW_ITEMS macro 5-1038
 GRID_FOCUSED macro 5-1038
 GRID_HEIGHT macro 5-1038
 GRID_HIDE macro 5-1038
 GRID_MAX_COLUMNS macro 5-1039
 GRID_MAX_ROWS macro 5-1039
 GRID_OUT_OF_BOUNDS macro 5-1039
 GRID_SHOW_BORDER_ONLY macro 5-1039

GRID_SHOW_FOCUS macro 5-1039
 GRID_SHOW_LINES macro 5-1039
 GRID_SHOW_SEPARATORS_ONLY macro 5-1039
 GRID_SUCCESS macro 5-1040
 GRID_TYPE_MASK macro 5-1040
 GRID_WIDTH macro 5-1040
 GridClearCellState function 5-752
 GridCreate function 5-753
 GridDraw function 5-754
 GridFreeItems function 5-754
 GridGetCell function 5-755
 GridGetFocusX macro 5-1040
 GridGetFocusY macro 5-1041
 GRIDITEM structure 5-933
 GRIDITEM_DRAW macro 5-1041
 GRIDITEM_IS_BITMAP macro 5-1041
 GRIDITEM_IS_TEXT macro 5-1041
 GRIDITEM_SELECTED macro 5-1041
 GRIDITEM_TEXTBOTTOM macro 5-1042
 GRIDITEM_TEXTLEFT macro 5-1042
 GRIDITEM_TEXTRIGHT macro 5-1042
 GRIDITEM_TEXTTOP macro 5-1042
 GridMsgDefault function 5-755
 GridSetCell function 5-756
 GridSetCellState function 5-757
 GridSetFocus function 5-757
 GridTranslateMsg function 5-758

H

Handling Endpoint 0 (EP0) Packets 5-3976
 Handling Errors 5-2085, 5-2487
 Handling Events 5-1562
 Hardware 3-103, 3-124, 3-143
 Hardware Abstraction Model 5-1362, 5-1473, 5-1521, 5-1564,
 5-1669, 5-1735, 5-1905, 5-2087, 5-2172, 5-2203, 5-2253,
 5-2289, 5-2332, 5-2429, 5-2479, 5-2595, 5-2663, 5-2689,
 5-2705, 5-2763, 5-2801, 5-2893, 5-3012, 5-3087, 5-3322
 Hardware Abstraction Models 5-3171
 Hardware Drivers 5-309
 Hardware Information 3-103, 3-124, 3-143

help_sys_ports.h 5-3498	HTTP_USE_POST macro 5-3665
hid_basic 3-158	HTTPCheckAuth function 5-3679
hid_joystick 3-161	HTTPExecuteGet function 5-3680
hid_keyboard 3-161	HTTPExecutePost function 5-3681
hid_mouse 3-160	HTTPNeedsAuth function 5-3682
HIDE_DATA macro 5-1042	HTTPPrint_varname function 5-3683
Host 3-167	HTTPReadPostPair macro 5-3687
How the Library Works 5-195, 5-345, 5-369, 5-399, 5-421, 5-459, 5-514, 5-554, 5-607, 5-697, 5-1112, 5-1315, 5-1367, 5-1474, 5-1522, 5-1555, 5-1670, 5-1736, 5-1866, 5-1906, 5-2061, 5-2173, 5-2205, 5-2255, 5-2290, 5-2333, 5-2431, 5-2480, 5-2598, 5-2666, 5-2690, 5-2707, 5-2764, 5-2802, 5-2896, 5-3015, 5-3088, 5-3321, 5-3350, 5-3365, 5-3422, 5-3443, 5-3470, 5-3506, 5-3531, 5-3576, 5-3616, 5-3627, 5-3869, 5-3883, 5-3932, 5-4007, 5-4170	HTTPS_DEFAULT_FILE macro 5-3665
HTTP Changes 5-3551	HTTPS_PORT macro 5-3665
HTTP Server TCP/IP Stack Library 5-3659	
http.h 5-3688	I2C Peripheral Library 5-2059
HTTP_CACHE_LEN macro 5-3662	I2C_MODULE_ID enumeration 5-2165
http_config.h 5-3689	IC_BUFFER_SIZE enumeration 5-2188
HTTP_CONFIG_FLAGS macro 5-3662	IC_CLOCK_SOURCES enumeration 5-2188
HTTP_CONN_HANDLE type 5-3684	IC_EDGE_TYPES enumeration 5-2189
HTTP_DEFAULT_FILE macro 5-3662	IC_EVENTS_PER_INTERRUPT enumeration 5-2189
HTTP_DEFAULT_LEN macro 5-3663	IC_INPUT_CAPTURE_MODES enumeration 5-2189
HTTP_FILE_TYPE enumeration 5-3684	IC_MODULE_ID enumeration 5-2190
HTTP_FILE_UPLOAD macro 5-3663	IC_SYNC_MODE_INPUTS enumeration 5-2191
HTTP_IO_RESULT enumeration 5-3684	IC_SYNC_MODES enumeration 5-2192
HTTP_MAX_CONNECTIONS macro 5-3663	IC_TIMERS enumeration 5-2192
HTTP_MAX_DATA_LEN macro 5-3663	ICMP TCP/IP Stack Library 5-3690
HTTP_MAX_HEADER_LEN macro 5-3663	icmp.h 5-3696
HTTP_MIN_CALLBACK_FREE macro 5-3663	icmp_config.h 5-3697
HTTP_MODULE_CONFIG structure 5-3685	ICMP_ECHO_RESULT enumeration 5-3695
HTTP_MODULE_FLAGS enumeration 5-3685	ICMP_HANDLE type 5-3695
HTTP_PORT macro 5-3664	ICMP_MODULE_CONFIG structure 5-3696
HTTP_READ_STATUS enumeration 5-3685	ICMP_TIMEOUT macro 5-3692
HTTP_SKT_RX_BUFF_SIZE macro 5-3664	ICMPv6 TCP/IP Stack Library 5-3697
HTTP_SKT_TX_BUFF_SIZE macro 5-3664	icmpv6.h 5-3707
HTTP_SSL_ONLY_CHAR macro 5-3664	icmpv6_config.h 5-3708
HTTP_STATUS enumeration 5-3686	ICMPV6_ERR_DU_CODE enumeration 5-3704
HTTP_TIMEOUT macro 5-3664	ICMPV6_ERR_PP_CODE enumeration 5-3705
HTTP_USE_AUTHENTICATION macro 5-3664	ICMPV6_ERR_PTB_CODE macro 5-3703
HTTP_USE_COOKIES macro 5-3665	ICMPV6_ERR_TE_CODE enumeration 5-3705
	ICMPV6_HANDLE type 5-3705
	ICMPV6_HEADER_ECHO structure 5-3706
	ICMPV6_HEADER_ERROR structure 5-3706

-
- ICMPV6_INFO_EREQ_CODE macro 5-3704
 - ICMPV6_INFO_ERPL_CODE macro 5-3704
 - ICMPV6_PACKET_TYPES enumeration 5-3706
 - ICMPV6_TIMEOUT macro 5-3700
 - IMAGE_STRETCH enumeration 5-933
 - ImageAbort macro 5-1042
 - ImageDecoderInit function 5-874
 - ImageDecodeTask function 5-874
 - ImageFullScreenDecode macro 5-1043
 - ImageLoopCallbackRegister function 5-875
 - Images 5-1117, 5-1124
 - IMG_ALIGN_CENTER macro 5-1044
 - IMG_bAlignCenter variable 5-885
 - IMG_bDownScalingFactor variable 5-885
 - IMG_bAbortImageDecoding variable 5-886
 - IMG_DECODE_ABORTED macro 5-1044
 - IMG_DOWN_SCALE macro 5-1044
 - IMG_FCLOSE macro 5-1044
 - IMG_FEOF macro 5-1044
 - IMG_FILE macro 5-1044
 - IMG_FILE_FORMAT type 5-933
 - IMG_FILE_SYSTEM_API type 5-933
 - IMG_FOPEN macro 5-1045
 - IMG_FREAD macro 5-1045
 - IMG_FSEEK macro 5-1045
 - IMG_FTELL macro 5-1045
 - IMG_LOOP_CALLBACK type 5-933
 - IMG_pCurrentFile variable 5-886
 - IMG_pFileAPIs variable 5-886
 - IMG_PIXEL_OUTPUT type 5-934
 - IMG_PIXEL_XY_RGB_888 type 5-934
 - IMG_PixelXYColor variable 5-886
 - IMG_pLoopCallbackFn variable 5-886
 - IMG_pPixelOutput variable 5-886
 - IMG_SCREEN_HEIGHT macro 5-1045
 - IMG_SCREEN_WIDTH macro 5-1045
 - IMG_vCheckAndAbort macro 5-1045
 - IMG_vLoopCallback macro 5-1046
 - IMG_vPutPixel macro 5-1046
 - IMG_vSetboundaries function 5-875
 - IMG_vSetColor macro 5-1046
 - IMG_wHeight variable 5-886
 - IMG_wImageHeight variable 5-887
 - IMG_wImageWidth variable 5-887
 - IMG_wStartX variable 5-887
 - IMG_wStartY variable 5-887
 - IMG_wWidth variable 5-887
 - in_addr structure 5-3611
 - INADDR_ANY macro 5-3609
 - Initator Initialization 5-2764
 - Initialization 5-1367, 5-1670, 5-1915, 5-2481, 5-3350, 5-3387
 - Initialization Functions 7-4238
 - Initialization of CAN 5-1557
 - Initialization Overrides 5-474, 5-574, 5-608, 5-1323
 - Initializing and Communicating with the Endpoint 5-3975
 - Initializing the Data Structure 5-4014
 - Initializing the I2C 5-2061
 - Initializing the Library 5-3980, 5-4007
 - Initializing the USB Device Stack 5-3973
 - Input Capture Module Setup 5-2173
 - Input Capture Peripheral Library 5-2170
 - INPUT_DEVICE_EVENT enumeration 5-934
 - INPUT_DEVICE_TYPE enumeration 5-935
 - Installing the Configurator Plug-in 2-42
 - INT_EXTERNAL_SOURCES enumeration 5-2232
 - INT_PRIORITY_LEVEL enumeration 5-2233
 - INT_SOURCE enumeration 5-2233
 - INT_SUBPRIORITY_LEVEL enumeration 5-2237
 - INT_TRAP_SOURCE enumeration 5-2238
 - INT_VECTOR enumeration 5-2238
 - int16_gfx_image_prog type 5-935
 - int16_prog type 5-935
 - int16_prog_pack type 5-935
 - int16c structure 5-1264
 - int32_gfx_image_prog type 5-936
 - int32_prog type 5-936
 - int32_prog_pack type 5-936
 - int32c structure 5-1265
-

int8_gfx_image_prog type 5-936	IPV4_QUEUED_PACKET_TIMEOUT macro 5-3711
int8_prog type 5-936	IPV4_TASK_PROCESS_RATE macro 5-3711
int8_prog_pack type 5-936	IPv6 TCP/IP Stack Library 5-3718
Internal FRC Oscillator Tuning 5-2335	ipv6.h 5-3745
Interrupt Control and Management 5-1738	IPV6_ACTION enumeration 5-3739
Interrupt Operation 5-1320	IPV6_ADDRESS_POLICY structure 5-3739
Interrupt Peripheral Library 5-2202	IPV6_ADDRESS_PREFERENCE enumeration 5-3739
Interrupt State Machine 5-2076	IPV6_ADDRESS_TYPE union 5-3739
Interrupt System Service Library 5-3419	IPV6_DATA_DYNAMIC_BUFFER macro 5-3735
Interrupt System Setup 5-3422	IPV6_DATA_NETWORK_FIFO macro 5-3736
Interrupts 5-2823	IPV6_DATA_NONE macro 5-3736
Introduction 2-40, 3-64, 3-84, 3-98, 3-102, 3-113, 3-123, 3-130, 3-142, 3-149, 4-178, 5-193, 5-226, 5-254, 5-289, 5-299, 5-309, 5-319, 5-333, 5-342, 5-367, 5-397, 5-418, 5-457, 5-510, 5-552, 5-600, 5-680, 5-694, 5-1110, 5-1127, 5-1167, 5-1275, 5-1311, 5-1361, 5-1471, 5-1519, 5-1553, 5-1667, 5-1733, 5-1864, 5-1902, 5-2059, 5-2170, 5-2202, 5-2252, 5-2287, 5-2330, 5-2427, 5-2478, 5-2593, 5-2662, 5-2687, 5-2703, 5-2762, 5-2798, 5-2891, 5-3011, 5-3085, 5-3167, 5-3319, 5-3347, 5-3363, 5-3376, 5-3419, 5-3438, 5-3465, 5-3500, 5-3529, 5-3536, 5-3546, 5-3569, 5-3574, 5-3598, 5-3614, 5-3625, 5-3636, 5-3646, 5-3654, 5-3659, 5-3690, 5-3697, 5-3709, 5-3718, 5-3747, 5-3771, 5-3794, 5-3798, 5-3806, 5-3809, 5-3822, 5-3858, 5-3867, 5-3881, 5-3913, 5-3917, 5-3930, 5-3955, 5-3963, 5-3973, 5-3977, 5-4004, 5-4060, 5-4100, 5-4131, 5-4168, 5-4179, 5-4199, 6-4226, 7-4228, 7-4230, 7-4232, 7-4234, 7-4236	IPV6_DATA_PIC_RAM macro 5-3736
INVALID_SOCKET macro 5-3910	IPV6_DATA_SEGMENT_HEADER type 5-3740
INVALID_TCP_PORT macro 5-3609	IPV6_DEFAULT_BASE_REACHABLE_TIME macro 5-3722
INVALID_UDP_SOCKET macro 5-3950	IPV6_DEFAULT_CUR_HOP_LIMIT macro 5-3722
IP_ADDR_ANY macro 5-3609	IPV6_DEFAULT_LINK_MTU macro 5-3722
ip_config.h 5-3716, 5-3747	IPV6_DEFAULT_RETRANSMIT_TIME macro 5-3722
IP_DEFAULT_ALLOCATION_BLOCK_SIZE macro 5-3711	IPV6_EVENT_HANDLER type 5-3740
IP_VERSION_4 macro 5-3735	IPV6_EVENT_TYPE enumeration 5-3740
IP_VERSION_6 macro 5-3735	IPV6_FRAGMENT_HEADER structure 5-3741
IPPROTO_IP macro 5-3609	IPV6_HANDLE type 5-3741
IPPROTO_TCP macro 5-3609	IPV6_HEADER structure 5-3741
IPPROTO_UDP macro 5-3609	IPV6_HEADER_OFFSET_DEST_ADDR macro 5-3736
IPv4 TCP/IP Stack Library 5-3708	IPV6_HEADER_OFFSET_NEXT_HEADER macro 5-3736
ipv4.h 5-3717	IPV6_HEADER_OFFSET_PAYLOAD_LENGTH macro 5-3736
IPV4_HEADER structure 5-3715	IPV6_HEADER_OFFSET_SOURCE_ADDR macro 5-3737
IPV4_HEADER_TYPE enumeration 5-3715	IPV6_INIT_TASK_PROCESS_RATE macro 5-3720
IPV4_MODULE_CONFIG structure 5-3715	IPV6_MINIMUM_LINK_MTU macro 5-3721
IPV4_PACKET structure 5-3716	IPV6_MODULE_CONFIG structure 5-3742
IPV4_QUEUED_PACKET_LIMIT macro 5-3711	IPV6_MTU_INCREASE_TIMEOUT macro 5-3802
	IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT macro 5-3721
	IPV6_NEXT_HEADER_TYPE enumeration 5-3742
	IPV6_NO_UPPER_LAYER_CHECKSUM macro 5-3737
	IPV6_PACKET type 5-3743
	IPV6_PACKET_ACK_FNC type 5-3743
	IPV6_QUEUE_MCAST_PACKET_LIMIT macro 5-3721
	IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT macro 5-3721
	IPV6_QUEUED_MCAST_PACKET_TIMEOUT macro 5-3721
	IPV6_RX_FRAGMENT_BUFFER type 5-3743

IPV6_SEGMENT_TYPE enumeration 5-3743
 IPV6_SNMP_TRAP_INFO structure 5-3848
 IPV6_TASK_PROCESS_RATE macro 5-3721
 IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM macro 5-3737
 IPV6_TLV_HBHO_ROUTER_ALERT macro 5-3737
 IPV6_TLV_OPTION_TYPE union 5-3744
 IPV6_TLV_PAD_1 macro 5-3737
 IPV6_TLV_PAD_N macro 5-3737
 IPV6_TLV_UNREC_OPT_DISCARD_PP macro 5-3737
 IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC macro 5-3738
 IPV6_TLV_UNREC_OPT_DISCARD_SILENT macro 5-3738
 IPV6_TLV_UNREC_OPT_SKIP_OPTION macro 5-3738
 IPV6_ULA_FLAGS enumeration 5-3744
 IPV6_ULA_NTP_ACCESS_TMO macro 5-3722
 IPV6_ULA_NTP_VALID_WINDOW macro 5-3722
 IPV6_ULA_RESULT enumeration 5-3745
 Isochronous Transfers 5-571
 IsPaletteEnabled function 5-875
 ISR Implementation 7-4229

K

KEYMEMBER structure 5-936
 KHAKI macro 5-1046

L

LAYER_ACTIONS enumeration 5-937
 LAYER_REGISTERS structure 5-332
 LAYER_TYPE enumeration 5-294
 lcc_demo 3-85
 LCC_TASK enumeration 5-307
 LED Control Functions 7-4238
 LibQ Fixed-Point Math Library 5-1275
 libq.h 5-1307
 Library Architecture 5-2428, 5-2690, 5-2800, 5-3978, 5-4061, 5-4101, 5-4132
 Library Configuration 5-4022, 5-4065, 5-4104, 5-4134
 Library Initialization 5-4065, 5-4134
 Library Interface 5-204, 5-237, 5-262, 5-291, 5-301, 5-311, 5-326, 5-335, 5-352, 5-377, 5-400, 5-428, 5-482, 5-519, 5-574,

5-609, 5-702, 5-1124, 5-1130, 5-1171, 5-1278, 5-1325, 5-1381, 5-1480, 5-1526, 5-1567, 5-1672, 5-1748, 5-1866, 5-1922, 5-2089, 5-2175, 5-2206, 5-2257, 5-2294, 5-2337, 5-2434, 5-2488, 5-2601, 5-2666, 5-2690, 5-2712, 5-2766, 5-2824, 5-2903, 5-3022, 5-3101, 5-3182, 5-3323, 5-3332, 5-3353, 5-3368, 5-3397, 5-3426, 5-3444, 5-3474, 5-3517, 5-3532, 5-3572, 5-3580, 5-3601, 5-3618, 5-3630, 5-3640, 5-3649, 5-3658, 5-3665, 5-3692, 5-3700, 5-3712, 5-3723, 5-3753, 5-3773, 5-3797, 5-3804, 5-3808, 5-3815, 5-3832, 5-3863, 5-3875, 5-3888, 5-3916, 5-3921, 5-3935, 5-3958, 5-3985, 5-4023, 5-4075, 5-4120, 5-4141, 5-4173, 5-4180, 5-4200, 7-4238

Library Overview 5-194, 5-230, 5-257, 5-290, 5-300, 5-310, 5-321, 5-334, 5-344, 5-369, 5-399, 5-420, 5-459, 5-514, 5-553, 5-607, 5-696, 5-1112, 5-1129, 5-1168, 5-1276, 5-1314, 5-1473, 5-1566, 5-2598, 5-3014, 5-3173, 5-3322, 5-3349, 5-3365, 5-3422, 5-3442, 5-3470, 5-3506, 5-3531, 5-3571, 5-3576, 5-3600, 5-3616, 5-3627, 5-3638, 5-3647, 5-3656, 5-3661, 5-3691, 5-3699, 5-3710, 5-3719, 5-3749, 5-3772, 5-3796, 5-3801, 5-3810, 5-3823, 5-3859, 5-3869, 5-3882, 5-3914, 5-3919, 5-3932, 5-3956, 5-4007, 5-4101, 7-4231, 7-4233, 7-4235, 7-4237

Library Usage Model 5-1366, 5-1670, 5-1736, 5-2088, 5-2204, 5-2254, 5-2333, 5-2480, 5-2665, 5-2706, 5-2895, 5-3088

LIGHTBLUE macro 5-1046

LIGHTCYAN macro 5-1047

LIGHTGRAY macro 5-1047

LIGHTGREEN macro 5-1047

LIGHTMAGENTA macro 5-1047

LIGHTORANGE macro 5-1047

LIGHTRED macro 5-1047

LIGHTYELLOW macro 5-1048

Link Local (Zeroconf) 5-3956

listen function 5-3605

LOCAL_TCP_PORT_END_NUMBER macro 5-3885

LOCAL_TCP_PORT_START_NUMBER macro 5-3885

Low-Cost Controllerless (LCC) Graphics Driver Library 5-299

M

MAC TCP/IP Stack Library 5-3747

MAGENTA macro 5-1048

Main Program Changes 5-684, 5-3555

MAIN_RETURN macro 5-3341

MAIN_RETURN_CODE macro 5-3342

MAIN_RETURN_CODES enumeration 5-3334

Manager TCP/IP Stack Library 5-3770

Managing Slave Addresses 5-2082	More Information 1-20
Master Descriptor Table 5-4012	Mounting a Volume 5-3392
Master Mode 5-2819	MPFS2 Command Line Options 5-3544
Math Library Help 5-1167	MPFS2 Utility 5-3542
matrix32 structure 5-1265	MPLAB Harmony Configurator 2-40
MAX_ANYCAST_DELAY_TIME macro 5-3802	MPLAB Harmony Configurator Help 2-39
MAX_BSD_CLIENT_CONNECTIONS macro 5-3601	MPLAB Harmony Graphics Library Access Changes 5-682
MAX_BSD_SERVER_CONNECTIONS macro 5-3601	MPLAB Harmony Graphics Library API Changes 5-682
MAX_MULTICAST_SOLICIT macro 5-3802	MPLAB Harmony Graphics Library BSP Configuration 5-685
MAX_NEIGHBOR_ADVERTISEMENT macro 5-3802	MPLAB Harmony Graphics Library Configuration 5-685
MAX_NONBUFFERED_BYTE_COUNT macro 5-393	MPLAB Harmony Graphics Library Configuration Changes 5-685
MAX_RTR_SOLICITATION_DELAY macro 5-3802	MPLAB Harmony Graphics Library Design 5-682
MAX_RTR_SOLICITATIONS macro 5-3803	MPLAB Harmony Graphics Library File Name Compliance 5-686
MAX_SMTP_CONNECTIONS macro 5-3814	MPLAB Harmony Graphics Library Function Name Compliance 5-687
MAX_TELNET_CONNECTIONS macro 5-3915	MPLAB Harmony Graphics Library Initialization Changes 5-684
MAX_UNICAST_SOLICIT macro 5-3803	MPLAB Harmony Graphics Library Key Features 5-681
MAX16 macro 5-1267	MPLAB Harmony Graphics Library Structure 5-681
MAX32 macro 5-1267	MPLAB Harmony Graphics Library Utilities 5-685
MC_CRYPTAPI_H macro 5-1164	MPLAB Harmony TCP/IP Stack Access Changes 5-3556
MCHP_BITMAP_NORMAL macro 5-1048	MPLAB Harmony TCP/IP Stack API Changes 5-3548
MCHP_BITMAP_PALETTE_STR macro 5-1048	MPLAB Harmony TCP/IP Stack Configuration 5-3553
mDNS 5-3957	MPLAB Harmony TCP/IP Stack Configuration Changes 5-3553
MDNSD_ERR_CODE enumeration 5-3961	MPLAB Harmony TCP/IP Stack Design 5-3548
Media Driver Registration 5-4173	MPLAB Harmony TCP/IP Stack Function Name Compliance 5-3558
Media Independent Interface (MII) 5-1910	MPLAB Harmony TCP/IP Stack Heap Configuration 5-3553
Memory Access Control 5-1524	MPLAB Harmony TCP/IP Stack Initialization Changes 5-3556
Memory Operation 5-1322	MPLAB Harmony TCP/IP Stack Key Features 5-3546
Messaging Functionality 5-700	MPLAB Harmony TCP/IP Stack SSL/RSA Usage 5-3557
Messaging System Service Library 5-3438	MPLAB Harmony TCP/IP Stack Storage Changes 5-3552
Micrium uC_OS_III with Graphics 3-117	MPLAB Harmony TCP/IP Stack Structure 5-3547
Migration Guide 5-3386	MPLAB Harmony TCP/IP Stack Utilities 5-3557
MIN16 macro 5-1267	MPLAB Harmony vs. the Unified TCP/IP Stack 5-257
MIN32 macro 5-1267	MPLAB Harmony vs. Unified Stack Functions 5-230
Miscellaneous 5-3473, 5-3513	MRF24W Wi-Fi Driver Library 5-600
Miscellaneous Config 5-480	MSD Function Driver Registration 5-4172
Miscellaneous Configuration 5-428	msd_basic 3-164, 3-169
Miscellaneous Functions 5-2256	mul16 function 5-1263
Mode Configuration 5-1557	
Module Events 5-1563	

mul16r function 5-1263
 mul32 function 5-1263
 Multimedia Expansion Board (MEB) 3-96
 Multimedia Expansion Board II (MEB II) 3-82, 3-96, 3-107, 3-140, 3-175
 Mutex Operation 5-1318
 MY_DEFAULT_SNMP_IF macro 5-3826

N

NBNS TCP/IP Stack Library 5-3794
 nbns.h 5-3797
 nbns_config.h 5-3797
 NBNS_MODULE_CONFIG structure 5-3797
 NBNS_PORT macro 5-3796
 NDP TCP/IP Stack Library 5-3798
 ndp.h 5-3805
 ndp_config.h 5-3805
 NDP_TASK_TIMER_RATE macro 5-3803
 NDP_VALID_LIFETIME_TWO_HOURS macro 5-3803
 New MPLAB Harmony Graphics Library API Functions 5-684
 New MPLAB Harmony TCP/IP Stack API Functions 5-3555
 NMI Events Functions 5-2699
 nmv_mpfs_single_disk 3-73
 Non-Volatile Memory (NVM) Driver Library 5-342
 NOTIFY_COMMUNITY_LEN macro 5-3827
 NTP_DEFAULT_CONNECTION_TYPE macro 5-3860
 NTP_DEFAULT_IF macro 5-3861
 NTP_EPOCH macro 5-3861
 NTP_FAST_QUERY_INTERVAL macro 5-3861
 NTP_MAX_RETRIES macro 5-3861
 NTP_MAX_STRATUM macro 5-3861
 NTP_QUERY_INTERVAL macro 5-3861
 NTP_REPLY_TIMEOUT macro 5-3862
 NTP_SERVER macro 5-3862
 NTP_SERVER_PORT macro 5-3862
 NTP_TIME_STAMP_TMO macro 5-3862
 NTP_VERSION macro 5-3862
 NUM_ALPHA_LEVELS macro 5-294
 NVM Driver Demonstration 3-102

NVM Peripheral Library 5-2252
 nvm_fat_single_disk 3-66
 nvm_sdcard_fat_mpfs_multi_disk 3-75
 nvm_sdcard_fat_multi_disk 3-71

O

Object Initialization 5-4171
 object_demo 3-88
 Objects Functionality 5-697
 OC_16BIT_TIMERS enumeration 5-2312
 OC_BUFFER_SIZE enumeration 5-2312
 OC_CLOCK_RESOLUTION enumeration 5-2312
 OC_COMPARE_MODES enumeration 5-2313
 OC_FAULT_MODES enumeration 5-2314
 OC_FAULT_OUT enumeration 5-2314
 OC_FAULT_TRISTATE enumeration 5-2315
 OC_FAULTS enumeration 5-2315
 OC_MODULE_ID enumeration 5-2315
 OC_SYNC_MODES enumeration 5-2316
 OC_SYNC_SOURCES enumeration 5-2316
 OC_TRIGGER_STATUS_MODES enumeration 5-2318
 OID_MAX_LEN macro 5-3827
 One Shot Callback 5-3510
 ONEP25 macro 5-1048
 Opening a File 5-3393
 Opening the Driver 5-556
 Operating as a Master 5-2483
 Operating as a Master Receiver 5-2070
 Operating as a Master Transmitter 5-2065
 Operating as a Slave 5-2484
 Operating as a Slave Receiver 5-2062
 Operating as a Slave Transmitter 5-2063
 Operating System Abstraction Layer (OSAL) Library Help 5-1311
 Operating/Addressing Mode Management 5-1739
 Optional Interfaces 5-467
 ORANGE macro 5-1048
 OSAL Operation 5-1322
 osal.c 5-1325

osal.h 5-1358	OSAL_RESULT_FALSE enumeration member 5-1357
OSAL_CRIT_Enter function 5-1349	OSAL_RESULT_NOT_IMPLEMENTED enumeration member 5-1357
OSAL_CRIT_Leave function 5-1349	OSAL_RESULT_TRUE enumeration member 5-1357
OSAL_CRIT_TYPE enumeration 5-1350	OSAL_SEM_Create function 5-1333
OSAL_CRIT_TYPE_HIGH enumeration member 5-1350	OSAL_SEM_DECLARE macro 5-1337
OSAL_CRIT_TYPE_LOW enumeration member 5-1350	OSAL_SEM_Delete function 5-1334
OSAL_ERROR_CALLBACK_DEFAULT enumeration member 5-1352	OSAL_SEM_GetCount function 5-1334
OSAL_ERROR_CALLBACK_ERROR enumeration member 5-1352	OSAL_SEM_HANDLE_TYPE macro 5-1356
OSAL_ERROR_CALLBACK_OUT_OF_MEMORY enumeration member 5-1352	OSAL_SEM_Pend function 5-1335
OSAL_ERROR_CALLBACK_STACK_OVERFLOW enumeration member 5-1352	OSAL_SEM_Post function 5-1336
OSAL_ERROR_CALLBACK_TYPE enumeration 5-1352	OSAL_SEM_PostISR function 5-1336
OSAL_ErrorCallback function 5-1352	OSAL_SEM_TYPE enumeration 5-1357
OSAL_Free function 5-1353	OSAL_SEM_TYPE_BINARY enumeration member 5-1357
OSAL_Initialize function 5-1355	OSAL_SEM_TYPE_COUNTING enumeration member 5-1357
OSAL_ISR_Enter function 5-1351	OSAL_Start function 5-1356
OSAL_ISR_Exit function 5-1351	OSAL_THREAD_Create function 5-1327
OSAL_Malloc function 5-1354	OSAL_THREAD_CreateDaemon function 5-1328
OSAL_MUTEX_Create function 5-1338	OSAL_THREAD_FUNCTION type 5-1327
OSAL_MUTEX_DECLARE macro 5-1340	OSAL_THREAD_HANDLE_TYPE macro 5-1357
OSAL_MUTEX_Delete function 5-1338	OSAL_THREAD_PriorityGet function 5-1329
OSAL_MUTEX_HANDLE_TYPE macro 5-1356	OSAL_THREAD_PrioritySet function 5-1330
OSAL_MUTEX_Lock function 5-1339	OSAL_THREAD_Resume function 5-1330
OSAL_MUTEX_Unlock function 5-1340	OSAL_THREAD_ResumeAll function 5-1331
OSAL_Name function 5-1355	OSAL_THREAD_Sleep function 5-1331
OSAL_QUEUE_Add function 5-1341	OSAL_THREAD_Suspend function 5-1332
OSAL_QUEUE_AddHead function 5-1342	OSAL_THREAD_SuspendAll function 5-1333
OSAL_QUEUE_AddHeadISR function 5-1342	OSAL_USE_NONE macro 5-1357
OSAL_QUEUE_AddISR function 5-1343	OSAL_WAIT_FOREVER macro 5-1357
OSAL_QUEUE_Create function 5-1344	OSC_AUX_CLOCK_SOURCE enumeration 5-2394
OSAL_QUEUE_DECLARE macro 5-1341	OSC_AUX_MODE enumeration 5-2394
OSAL_QUEUE_Delete function 5-1345	OSC_AUXPLL_IN_DIV enumeration 5-2394
OSAL_QUEUE_HANDLE_TYPE macro 5-1356	OSC_AUXPLL_MULTIPLIER enumeration 5-2395
OSAL_QUEUE_Peek function 5-1345	OSC_AUXPLL_OUT_DIV enumeration 5-2395
OSAL_QUEUE_PeekISR function 5-1346	OSC_DOZE_RATIO enumeration 5-2396
OSAL_QUEUE_Remove function 5-1347	OSC_FRC_DIV enumeration 5-2396
OSAL_QUEUE_RemoveISR function 5-1348	OSC_FRC_TUNE enumeration 5-2397
OSAL_RESULT enumeration 5-1357	OSC_GFX_CLOCK enumeration 5-2398
	OSC_LFSR_TYPE enumeration 5-2398
	OSC_MODULE_ID enumeration 5-2399

OSC_OPERATION_ON_WAIT enumeration 5-2399
 OSC_PB_CLOCK_DIV_TYPE macro 5-2393
 OSC_PERIPHERAL_BUS enumeration 5-2399
 OSC_PLL_SELECT enumeration 5-2400
 OSC_PLLAUX_CLOCK_SOURCE enumeration 5-2400
 OSC_REF_BASECLOCK enumeration 5-2401
 OSC_REF_DIV enumeration 5-2401
 OSC_REF_DIVISOR_TYPE macro 5-2393
 OSC_REFERENCE enumeration 5-2402
 OSC_REFERENCE_MAX_DIV macro 5-2393
 OSC_SYS_CLOCK_DIV enumeration 5-2403
 OSC_SYS_TYPE enumeration 5-2403
 OSC_SYSPLL_FREQ_RANGE enumeration 5-2404
 OSC_SYSPLL_IN_CLK_SOURCE enumeration 5-2404
 OSC_SYSPLL_IN_DIV enumeration 5-2405
 OSC_SYSPLL_MULTIPLIER_TYPE macro 5-2393
 OSC_SYSPLL_OUT_DIV enumeration 5-2405
 OSC_TUNING_MODE enumeration 5-2406
 OSC_TUNING_SEQUENCERS enumeration 5-2406
 OSC_USBCLOCK_SOURCE enumeration 5-2407
 Oscillator Peripheral Library 5-2330
 Oscillator Selection and Switching 5-2333
 Oscillator Tuning 5-3351
 Other Features 5-2086, 5-2487, 5-2822, 5-3020, 5-3099
 Other Functionality 5-608, 5-1316, 5-2711
 Other Functions 7-4240
 Others 5-574, 5-609, 5-1324, 5-3366
 OTM2201A Graphics Controller Driver Library 5-309
 OTM2201A_TASK enumeration 5-318
 Output Compare Peripheral Library 5-2287
 Overview 3-65

P

PA6_IO macro 5-673
 PA6_TRISTATE macro 5-674
 PAGE_TYPE enumeration 5-294
 PALETTE_ENTRY union 5-938
 PALETTE_EXTERNAL type 5-938
 PALETTE_FLASH structure 5-938
 PALETTE_HEADER structure 5-939
 Palettelnit function 5-876
 Parallel Master Port (PMP) Driver Library 5-367
 PARM_EQUAL_FILTER structure 5-1265
 PARM_EQUAL_FILTER_16 structure 5-1266
 PARM_EQUAL_FILTER_32 structure 5-1266
 PARM_FILTER_GAIN structure 5-1267
 PCACHE_MODULE_ID enumeration 5-2463
 Percentage2Alpha function 5-291
 Period Modification 5-464
 Periodic Callback 5-3509
 Peripheral Libraries 4-182
 Peripheral Library Example Applications 3-98
 Peripheral Library Help 5-1361
 Peripheral Pin Select 5-3473
 PERU macro 5-1049
 PGV Error Handling 5-2765
 PHY_NEG_DONE_TMO macro 5-261
 PHY_NEG_INIT_TMO macro 5-262
 PHY_RESET_CLR_TMO macro 5-262
 PIC32 Ethernet Starter Kit 3-136
 PIC32 USB Digital Audio Accessory Board 3-173
 PIC32 USB Starter Kit II 3-77, 3-91, 3-104, 3-118, 3-170
 PIC32 USB Starter Kit II BSP Library 7-4232
 pic32_ethernet_starter_kit 3-131
 PIC32MX USB Audio BSP Library 7-4234
 PIC32MX795F512 Plug-In-Module (PIM) 3-79
 PIC32MX795F512L Plug-in Module (PIM) 3-105, 3-125, 3-144
 PIC32MX795F512L Plug-In-Module (PIM) 3-96, 3-173
 PIC32MZ Embedded Connectivity (EC) Starter Kit 3-80, 3-92, 3-106, 3-119, 3-137, 3-173
 PIC32MZ Embedded Connectivity (EC) Starter Kit BSP Library 7-4236
 PICTail Daughter Board for SD and MMC 3-80
 PICTail Plus LCC Daughter Board 3-95
 PICTail Plus S1D13517 Daughter Board 3-95
 PICTail Plus SSD1926 Daughter Board 3-95
 Pin Control 5-3470
 PIO Mode 5-2900
 PIP_BUFFER macro 5-307

Pipe Close 5-560	PLIB_ADC_ExistsConversionTriggerGroup function 5-1443
Pipe Setup 5-558	PLIB_ADC_ExistsConversionTriggerSource function 5-1444
Pipe Stall 5-561	PLIB_ADC_ExistsDMAAddressIncrement function 5-1444
Pipe Status 5-559	PLIB_ADC_ExistsDMABufferMode function 5-1445
Pipe Transfer Queue 5-560	PLIB_ADC_ExistsDMABuffersPerAnalogInput function 5-1445
plib_adc.h 5-1467	PLIB_ADC_ExistsDMAControl function 5-1446
PLIB_ADC_AsynchrousDedicatedSamplingDisable function 5-1417	PLIB_ADC_ExistsEnableControl function 5-1446
PLIB_ADC_AsynchrousDedicatedSamplingEnable function 5-1417	PLIB_ADC_ExistsGlobalSoftwareTrigger function 5-1447
PLIB_ADC_CalibrationDisable function 5-1389	PLIB_ADC_ExistsInputSelect function 5-1447
PLIB_ADC_CalibrationEnable function 5-1388	PLIB_ADC_ExistsInternalReferenceChannelControl function 5-1448
PLIB_ADC_ChannelGroupSelect function 5-1393	PLIB_ADC_ExistsISRJumpTableBaseAddress function 5-1448
PLIB_ADC_ConversionClockGet function 5-1402	PLIB_ADC_ExistsMuxChannel0NegativeInput function 5-1449
PLIB_ADC_ConversionClockSet function 5-1402	PLIB_ADC_ExistsMuxChannel0PositiveInput function 5-1449
PLIB_ADC_ConversionClockSourceSelect function 5-1403	PLIB_ADC_ExistsMuxChannel123NegativeInput function 5-1450
PLIB_ADC_ConversionHasCompleted function 5-1400	PLIB_ADC_ExistsMuxChannel123PositiveInput function 5-1450
PLIB_ADC_ConversionOrderSelect function 5-1404	PLIB_ADC_ExistsMuxInputScanControl function 5-1451
PLIB_ADC_ConversionStart function 5-1399	PLIB_ADC_ExistsMuxInputScanSelect function 5-1451
PLIB_ADC_ConversionStopSequenceDisable function 5-1401	PLIB_ADC_ExistsPairConversionControl function 5-1452
PLIB_ADC_ConversionStopSequenceEnable function 5-1401	PLIB_ADC_ExistsPairInterruptOnConversion function 5-1453
PLIB_ADC_ConversionTriggerGroupSelect function 5-1404	PLIB_ADC_ExistsPairInterruptRequest function 5-1453
PLIB_ADC_ConversionTriggerSourceSelect function 5-1400	PLIB_ADC_ExistsPairSampleStatus function 5-1454
PLIB_ADC_Disable function 5-1386	PLIB_ADC_ExistsPairTriggerSource function 5-1454
PLIB_ADC_DMAAddressIncrementSelect function 5-1405	PLIB_ADC_ExistsResultBufferFillStatus function 5-1455
PLIB_ADC_DMABufferModeSelect function 5-1405	PLIB_ADC_ExistsResultBufferMode function 5-1455
PLIB_ADC_DMADisable function 5-1406	PLIB_ADC_ExistsResultFormat function 5-1456
PLIB_ADC_DMAEnable function 5-1406	PLIB_ADC_ExistsResultGet function 5-1456
PLIB_ADC_DMAInputBufferSelect function 5-1407	PLIB_ADC_ExistsResultGetByIndex function 5-1457
PLIB_ADC_Enable function 5-1385	PLIB_ADC_ExistsResultSign function 5-1457
PLIB_ADC_ExistsAsynchronousDedicatedSampling function 5-1439	PLIB_ADC_ExistsResultSize function 5-1458
PLIB_ADC_ExistsCalibrationControl function 5-1439	PLIB_ADC_ExistsSampleResolution function 5-1458
PLIB_ADC_ExistsChannelGroup function 5-1440	PLIB_ADC_ExistsSamplesPerInterruptSelect function 5-1459
PLIB_ADC_ExistsConversionClock function 5-1440	PLIB_ADC_ExistsSamplingAcquisitionTime function 5-1459
PLIB_ADC_ExistsConversionClockSource function 5-1441	PLIB_ADC_ExistsSamplingAutoStart function 5-1460
PLIB_ADC_ExistsConversionControl function 5-1441	PLIB_ADC_ExistsSamplingControl function 5-1460
PLIB_ADC_ExistsConversionOrder function 5-1442	PLIB_ADC_ExistsSamplingModeControl function 5-1461
PLIB_ADC_ExistsConversionStatus function 5-1442	PLIB_ADC_ExistsSamplingStatus function 5-1461
PLIB_ADC_ExistsConversionStopSequenceControl function 5-1443	PLIB_ADC_ExistsStopInIdleControl function 5-1462
	PLIB_ADC_ExistsVoltageReference function 5-1462

PLIB_ADC_GlobalSoftwareTriggerSet function 5-1418	PLIB_ADC_SamplingStart function 5-1398
PLIB_ADC_InputScanMaskAdd function 5-1390	PLIB_ADC_SamplingStop function 5-1399
PLIB_ADC_InputScanMaskRemove function 5-1391	PLIB_ADC_StopInIdleDisable function 5-1386
PLIB_ADC_InputSelectNegative function 5-1390	PLIB_ADC_StopInIdleEnable function 5-1387
PLIB_ADC_InputSelectPositive function 5-1389	PLIB_ADC_VoltageReferenceSelect function 5-1387
PLIB_ADC_InternalReferenceChannelDisable function 5-1392	plib_adcp.h 5-1518
PLIB_ADC_InternalReferenceChannelEnable function 5-1392	PLIB_ADCP_AlternateInputSelect function 5-1489
PLIB_ADC_IsrJumpTableBaseAddressGet function 5-1419	PLIB_ADCP_CalibrationStart function 5-1484
PLIB_ADC_IsrJumpTableBaseAddressSet function 5-1419	PLIB_ADCP_ChannelScanConfigure function 5-1492
PLIB_ADC_MuxAInputScanDisable function 5-1463	PLIB_ADCP_Class12TriggerConfigure function 5-1495
PLIB_ADC_MuxAInputScanEnable function 5-1463	PLIB_ADCP_Configure function 5-1482
PLIB_ADC_MuxChannel0InputNegativeSelect function 5-1464	PLIB_ADCP_DefaultInputSelect function 5-1490
PLIB_ADC_MuxChannel0InputPositiveSelect function 5-1465	PLIB_ADCP_DigCmpAidGet function 5-1500
PLIB_ADC_MuxChannel123InputNegativeSelect function 5-1465	PLIB_ADCP_DigCmpConfig function 5-1500
PLIB_ADC_MuxChannel123InputPositiveSelect function 5-1466	PLIB_ADCP_DigCmpDisable function 5-1499
PLIB_ADC_PairConversionIsPending function 5-1408	PLIB_ADCP_DigCmpEnable function 5-1499
PLIB_ADC_PairConversionStart function 5-1408	PLIB_ADCP_Disable function 5-1483
PLIB_ADC_PairInterruptAfterFirstConversion function 5-1409	PLIB_ADCP_Enable function 5-1483
PLIB_ADC_PairInterruptAfterSecondConversion function 5-1409	PLIB_ADCP_ExistsCalibration function 5-1486
PLIB_ADC_PairInterruptRequestDisable function 5-1410	PLIB_ADCP_ExistsChannelScan function 5-1493
PLIB_ADC_PairInterruptRequestEnable function 5-1410	PLIB_ADCP_ExistsConfiguration function 5-1488
PLIB_ADC_PairSamplesAvailable function 5-1411	PLIB_ADCP_ExistsConversionResults function 5-1498
PLIB_ADC_PairSampleStatusClear function 5-1412	PLIB_ADCP_ExistsDigCmp function 5-1502
PLIB_ADC_PairTriggerSourceSelect function 5-1412	PLIB_ADCP_ExistsEnableControl function 5-1486
PLIB_ADC_ResultBufferModeSelect function 5-1413	PLIB_ADCP_ExistsInputSelect function 5-1491
PLIB_ADC_ResultBufferStatusGet function 5-1413	PLIB_ADCP_ExistsLowPowerControl function 5-1487
PLIB_ADC_ResultFormatSelect function 5-1414	PLIB_ADCP_ExistsModeSelect function 5-1491
PLIB_ADC_ResultGet function 5-1415	PLIB_ADCP_ExistsOsampDigFilter function 5-1505
PLIB_ADC_ResultGetByIndex function 5-1415	PLIB_ADCP_ExistsReadyStatus function 5-1489
PLIB_ADC_ResultSignGet function 5-1416	PLIB_ADCP_ExistsTriggering function 5-1495
PLIB_ADC_ResultSizeSelect function 5-1416	PLIB_ADCP_GlobalSoftwareTrigger function 5-1494
PLIB_ADC_SampleAcquisitionTimeSet function 5-1394	PLIB_ADCP_IndividualTrigger function 5-1494
PLIB_ADC_SampleAutoStartDisable function 5-1394	PLIB_ADCP_LowPowerStateEnter function 5-1484
PLIB_ADC_SampleAutoStartEnable function 5-1395	PLIB_ADCP_LowPowerStateExit function 5-1485
PLIB_ADC_SampleMaxGet function 5-1395	PLIB_ADCP_LowPowerStateGet function 5-1485
PLIB_ADC_SampleMinGet function 5-1396	PLIB_ADCP_ModuleIsReady function 5-1488
PLIB_ADC_SamplesPerInterruptSelect function 5-1396	PLIB_ADCP_OsampDigFilterConfig function 5-1505
PLIB_ADC_SamplingIsActive function 5-1397	PLIB_ADCP_OsampDigFilterDataGet function 5-1504
PLIB_ADC_SamplingModeSelect function 5-1397	PLIB_ADCP_OsampDigFilterDataRdy function 5-1503

PLIB_ADCP_OsampDigFilterDisable function 5-1503	PLIB_BMX_ExistsProgramFlashBootSize function 5-1549
PLIB_ADCP_OsampDigFilterEnable function 5-1502	PLIB_BMX_ExistsProgramFlashMemoryCacheDma function 5-1550
PLIB_ADCP_ResultGet function 5-1497	PLIB_BMX_ExistsProgramFlashMemorySize function 5-1550
PLIB_ADCP_ResultReady function 5-1496	PLIB_BMX_ExistsProgramFlashPartition function 5-1551
PLIB_ADCP_ResultReadyGrpIntConfigure function 5-1497	PLIB_BMX_PFM_BLOCK_SIZE macro 5-1544
PLIB_ADCP_SHModeSelect function 5-1492	PLIB_BMX_ProgramFlashBootSizeGet function 5-1540
plib_bmx.h 5-1552	PLIB_BMX_ProgramFlashMemoryCacheDmaDisable function 5-1533
PLIB_BMX_ARB_MODE enumeration 5-1543	PLIB_BMX_ProgramFlashMemoryCacheDmaEnable function 5-1534
PLIB_BMX_ArbitrationModeGet function 5-1535	PLIB_BMX_ProgramFlashMemorySizeGet function 5-1541
PLIB_BMX_ArbitrationModeSet function 5-1535	PLIB_BMX_ProgramFlashPartitionGet function 5-1541
PLIB_BMX_BusExceptionDataDisable function 5-1528	PLIB_BMX_ProgramFlashPartitionSet function 5-1542
PLIB_BMX_BusExceptionDataEnable function 5-1529	plib_can.h 5-1662
PLIB_BMX_BusExceptionDMADisable function 5-1529	PLIB_CAN_AllChannelEventsGet function 5-1595
PLIB_BMX_BusExceptionDMAEnable function 5-1530	PLIB_CAN_AllChannelOverflowStatusGet function 5-1596
PLIB_BMX_BusExceptionICDDisable function 5-1530	PLIB_CAN_BusActivityWakeupDisable function 5-1572
PLIB_BMX_BusExceptionICDEnable function 5-1531	PLIB_CAN_BusActivityWakeupEnable function 5-1572
PLIB_BMX_BusExceptionInstructionDisable function 5-1531	PLIB_CAN_BusLine3TimesSamplingDisable function 5-1583
PLIB_BMX_BusExceptionInstructionEnable function 5-1532	PLIB_CAN_BusLine3TimesSamplingEnable function 5-1584
PLIB_BMX_BusExceptionIXIDisable function 5-1532	PLIB_CAN_ChannelEventClear function 5-1597
PLIB_BMX_BusExceptionIXIEnable function 5-1533	PLIB_CAN_ChannelEventDisable function 5-1592
PLIB_BMX_DATA_RAM_WAIT_STATES enumeration 5-1543	PLIB_CAN_ChannelEventEnable function 5-1598
PLIB_BMX_DataRAMKernelProgramOffsetGet function 5-1536	PLIB_CAN_ChannelEventGet function 5-1599
PLIB_BMX_DataRAMPartitionSet function 5-1536	PLIB_CAN_ChannelForReceiveSet function 5-1589
PLIB_BMX_DataRAMSizeGet function 5-1537	PLIB_CAN_ChannelForTransmitSet function 5-1590
PLIB_BMX_DataRAMUserDataOffsetGet function 5-1538	PLIB_CAN_ChannelReset function 5-1590
PLIB_BMX_DataRAMUserProgramOffsetGet function 5-1538	PLIB_CAN_ChannelResetIsComplete function 5-1579
PLIB_BMX_DataRamWaitStateGet function 5-1539	PLIB_CAN_ChannelUpdate function 5-1591
PLIB_BMX_DataRamWaitStateSet function 5-1540	PLIB_CAN_DeviceNetConfigure function 5-1579
PLIB_BMX_DRM_BLOCK_SIZE macro 5-1544	PLIB_CAN_Disable function 5-1573
PLIB_BMX_EXCEPTION_SRC enumeration 5-1544	PLIB_CAN_Enable function 5-1574
PLIB_BMX_ExistsArbitrationMode function 5-1545	PLIB_CAN_ErrorStateGet function 5-1617
PLIB_BMX_ExistsBusExceptionData function 5-1545	PLIB_CAN_ExistsActiveStatus function 5-1636
PLIB_BMX_ExistsBusExceptionDMA function 5-1546	PLIB_CAN_ExistsAllChannelEvents function 5-1637
PLIB_BMX_ExistsBusExceptionICD function 5-1546	PLIB_CAN_ExistsAllChannelOverflow function 5-1637
PLIB_BMX_ExistsBusExceptionInstruction function 5-1547	PLIB_CAN_ExistsBusActivityWakeup function 5-1638
PLIB_BMX_ExistsBusExceptionIXI function 5-1547	PLIB_CAN_ExistsBusLine3TimesSampling function 5-1638
PLIB_BMX_ExistsDataRAMPartition function 5-1548	PLIB_CAN_ExistsChannelEvent function 5-1639
PLIB_BMX_ExistsDataRAMSize function 5-1548	
PLIB_BMX_ExistsDataRamWaitState function 5-1549	

PLIB_CAN_ExistsChannelEventEnable function 5-1639	PLIB_CAN_ExistsTimeStampValue function 5-1659
PLIB_CAN_ExistsChannelForReceiveSet function 5-1640	PLIB_CAN_ExistsTransmissionIsAborted function 5-1660
PLIB_CAN_ExistsChannelForTransmitSet function 5-1640	PLIB_CAN_ExistsTransmitBufferGet function 5-1660
PLIB_CAN_ExistsChannelReset function 5-1641	PLIB_CAN_ExistsTransmitChannelFlush function 5-1661
PLIB_CAN_ExistsChannelUpdate function 5-1641	PLIB_CAN_ExistsTransmitChannelStatus function 5-1661
PLIB_CAN_ExistsDeviceNet function 5-1642	PLIB_CAN_ExistsTransmitErrorCountGet function 5-1662
PLIB_CAN_ExistsEnableControl function 5-1642	PLIB_CAN_FilterConfigure function 5-1612
PLIB_CAN_ExistsErrorState function 5-1643	PLIB_CAN_FilterDisable function 5-1612
PLIB_CAN_ExistsFilterConfigure function 5-1643	PLIB_CAN_FilterEnable function 5-1613
PLIB_CAN_ExistsFilterEnable function 5-1644	PLIB_CAN_FilterIsDisabled function 5-1614
PLIB_CAN_ExistsFilterMaskConfigure function 5-1644	PLIB_CAN_FilterMaskConfigure function 5-1614
PLIB_CAN_ExistsFilterToChannelLink function 5-1645	PLIB_CAN_FilterToChannelLink function 5-1615
PLIB_CAN_ExistsFunctionalMode function 5-1645	PLIB_CAN_FunctionalModeGet function 5-1580
PLIB_CAN_ExistsLatestFilterMatchGet function 5-1646	PLIB_CAN_FunctionalModeSelect function 5-1583
PLIB_CAN_ExistsMemoryBufferAssign function 5-1646	PLIB_CAN_IsActive function 5-1574
PLIB_CAN_ExistsMessageIsReceived function 5-1647	PLIB_CAN_LatestFilterMatchGet function 5-1616
PLIB_CAN_ExistsModuleEventClear function 5-1647	PLIB_CAN_MemoryBufferAssign function 5-1580
PLIB_CAN_ExistsModuleEventEnable function 5-1648	PLIB_CAN_MessageIsReceived function 5-1609
PLIB_CAN_ExistsModuleInfo function 5-1648	PLIB_CAN_ModuleEventClear function 5-1593
PLIB_CAN_ExistsOperationModeRead function 5-1649	PLIB_CAN_ModuleEventDisable function 5-1593
PLIB_CAN_ExistsOperationModeWrite function 5-1649	PLIB_CAN_ModuleEventEnable function 5-1594
PLIB_CAN_ExistsPendingEventsGet function 5-1650	PLIB_CAN_ModuleEventGet function 5-1595
PLIB_CAN_ExistsPendingTransmissionsAbort function 5-1650	PLIB_CAN_OperationModeGet function 5-1575
PLIB_CAN_ExistsPhaseSegment1Length function 5-1651	PLIB_CAN_OperationModeSelect function 5-1576
PLIB_CAN_ExistsPhaseSegment2Length function 5-1651	PLIB_CAN_PendingEventsGet function 5-1600
PLIB_CAN_ExistsPhaseSegment2LengthFreelyProgrammable function 5-1652	PLIB_CAN_PendingTransmissionsAbort function 5-1601
PLIB_CAN_ExistsPropagationTimeSegment function 5-1652	PLIB_CAN_PhaseSegment1LengthSet function 5-1585
PLIB_CAN_ExistsReceiveBufferClear function 5-1653	PLIB_CAN_PhaseSegment2LengthFreelyProgrammableDisable function 5-1586
PLIB_CAN_ExistsReceiveBufferIsEmpty function 5-1653	PLIB_CAN_PhaseSegment2LengthFreelyProgrammableEnable function 5-1586
PLIB_CAN_ExistsReceivedMessageGet function 5-1654	PLIB_CAN_PhaseSegment2LengthSet function 5-1587
PLIB_CAN_ExistsReceiveErrorCount function 5-1654	PLIB_CAN_PropagationTimeSegmentSet function 5-1588
PLIB_CAN_ExistsReceiveModeSelect function 5-1655	PLIB_CAN_ReceiveBufferClear function 5-1610
PLIB_CAN_ExistsRemoteTransmitReq function 5-1655	PLIB_CAN_ReceiveBufferIsEmpty function 5-1610
PLIB_CAN_ExistsRxFIFOFourLeftNotify function 5-1656	PLIB_CAN_ReceivedMessageGet function 5-1605
PLIB_CAN_ExistsRxFIFOOneLeftNotify function 5-1656	PLIB_CAN_ReceiveErrorCountGet function 5-1616
PLIB_CAN_ExistsStopInIdle function 5-1657	PLIB_CAN_ReceiveModeGet function 5-1608
PLIB_CAN_ExistsSyncJumpWidthSet function 5-1657	PLIB_CAN_ReceiveModeSelect function 5-1608
PLIB_CAN_ExistsTimeStampEnable function 5-1658	
PLIB_CAN_ExistsTimeStampPrescaler function 5-1658	

PLIB_CAN_RemoteTransmitReqsReceived function 5-1611	PLIB_CMP_CVREF_Output2Disable function 5-1677
PLIB_CAN_RxFIFOFourLeftNotify function 5-1607	PLIB_CMP_CVREF_Output2Enable function 5-1677
PLIB_CAN_RxFIFOOneLeftNotify function 5-1607	PLIB_CMP_CVREF_OutputDisable function 5-1678
PLIB_CAN_StopInIdleDisable function 5-1576	PLIB_CMP_CVREF_OutputEnable function 5-1678
PLIB_CAN_StopInIdleEnable function 5-1577	PLIB_CMP_CVREF_ReferenceVoltageSelect function 5-1679
PLIB_CAN_SyncJumpWidthSet function 5-1588	PLIB_CMP_CVREF_SourceNegativeInputSelect function 5-1679
PLIB_CAN_TimeStampDisable function 5-1577	PLIB_CMP_CVREF_SourceVoltageSelect function 5-1680
PLIB_CAN_TimeStampEnable function 5-1578	PLIB_CMP_CVREF_ValueGet function 5-1680
PLIB_CAN_TimeStampPrescalerSet function 5-1581	PLIB_CMP_CVREF_ValueSelect function 5-1681
PLIB_CAN_TimeStampValueGet function 5-1582	PLIB_CMP_CVREF_WideRangeDisable function 5-1681
PLIB_CAN_TimeStampValueSet function 5-1582	PLIB_CMP_CVREF_WideRangeEnable function 5-1682
PLIB_CAN_TotalChannelsGet function 5-1618	PLIB_CMP_CVREF_WideRangeIsEnabled function 5-1682
PLIB_CAN_TotalFiltersGet function 5-1618	PLIB_CMP_Disable function 5-1683
PLIB_CAN_TotalMasksGet function 5-1619	PLIB_CMP_Enable function 5-1683
PLIB_CAN_TransmissionIsAborted function 5-1601	PLIB_CMP_EventHasOccurred function 5-1684
PLIB_CAN_TransmitBufferGet function 5-1602	PLIB_CMP_EventStatusClear function 5-1684
PLIB_CAN_TransmitChannelFlush function 5-1603	PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect function 5-1722
PLIB_CAN_TransmitChannelStatusGet function 5-1604	PLIB_CMP_ExistsCVREFEnableControl function 5-1723
PLIB_CAN_TransmitErrorCountGet function 5-1605	PLIB_CMP_ExistsCVREFOutputEnableControl function 5-1723
plib_cmp.h 5-1730	PLIB_CMP_ExistsCVREFRefVoltageRangeSelect function 5-1724
PLIB_CMP_AndGateAInputDisable function 5-1696	PLIB_CMP_ExistsCVREFValueSelect function 5-1724
PLIB_CMP_AndGateAInputEnable function 5-1697	PLIB_CMP_ExistsCVREFVoltageRangeSelect function 5-1725
PLIB_CMP_AndGateBInputDisable function 5-1697	PLIB_CMP_ExistsCVREFWideRangeControl function 5-1725
PLIB_CMP_AndGateBInputEnable function 5-1698	PLIB_CMP_ExistsEnableControl function 5-1726
PLIB_CMP_AndGateCInputDisable function 5-1704	PLIB_CMP_ExistsInterruptEventSelect function 5-1727
PLIB_CMP_AndGateCInputEnable function 5-1698	PLIB_CMP_ExistsInvertingInputSelect function 5-1727
PLIB_CMP_AndGateInvertedAInputDisable function 5-1699	PLIB_CMP_ExistsInvertOutputControl function 5-1728
PLIB_CMP_AndGateInvertedAInputEnable function 5-1699	PLIB_CMP_ExistsNonInvertingInputSelect function 5-1728
PLIB_CMP_AndGateInvertedBInputDisable function 5-1700	PLIB_CMP_ExistsOutputEnableControl function 5-1729
PLIB_CMP_AndGateInvertedBInputEnable function 5-1700	PLIB_CMP_ExistsOutputLevelControl function 5-1729
PLIB_CMP_AndGateInvertedCInputDisable function 5-1701	PLIB_CMP_ExistsStopInIdle function 5-1730
PLIB_CMP_AndGateInvertedCInputEnable function 5-1701	PLIB_CMP_FilterClockDivideSelect function 5-1704
PLIB_CMP_Cmp1OutputHighSelect function 5-1690	PLIB_CMP_FilterDisable function 5-1705
PLIB_CMP_Cmp1OutputLowSelect function 5-1690	PLIB_CMP_FilterEnable function 5-1705
PLIB_CMP_Cmp2OutputHighSelect function 5-1691	PLIB_CMP_FilterInputClockSelect function 5-1706
PLIB_CMP_Cmp2OutputLowSelect function 5-1691	PLIB_CMP_HighLevelMaskSelect function 5-1715
PLIB_CMP_CVREF_BandGapReferenceSourceSelect function 5-1676	PLIB_CMP_HysteresisDisable function 5-1692
PLIB_CMP_CVREF_Disable function 5-1689	
PLIB_CMP_CVREF_Enable function 5-1676	

PLIB_CMP_HysteresisEnable function 5-1692	PLIB_DMA_BusyActiveReset function 5-1755
PLIB_CMP_InterruptEventSelect function 5-1685	PLIB_DMA_BusyActiveSet function 5-1755
PLIB_CMP_InvertedOutputSelect function 5-1686	PLIB_DMA_ChannelBitsGet function 5-1820
PLIB_CMP_InvertingInputChannelSelect function 5-1686	PLIB_DMA_ChannelPriorityGet function 5-1788
PLIB_CMP_LowLevelMaskSelect function 5-1715	PLIB_DMA_ChannelPrioritySelect function 5-1788
PLIB_CMP_NegativeAndGateOutputDisable function 5-1702	PLIB_DMA_ChannelXAbortIRQSet function 5-1759
PLIB_CMP_NegativeAndGateOutputEnable function 5-1702	PLIB_DMA_ChannelXAddressModeGet function 5-1766
PLIB_CMP_NonInvertedOutputSelect function 5-1687	PLIB_DMA_ChannelXAddressModeSelect function 5-1767
PLIB_CMP_NonInvertingInputChannelSelect function 5-1687	PLIB_DMA_ChannelXAutoDisable function 5-1789
PLIB_CMP_OpAmpModeDisable function 5-1706	PLIB_DMA_ChannelXAutoEnable function 5-1789
PLIB_CMP_OpAmpModeEnable function 5-1707	PLIB_DMA_ChannelXAutsEnabled function 5-1815
PLIB_CMP_OpAmpOutputDisable function 5-1707	PLIB_DMA_ChannelXBufferedDataIsWritten function 5-1816
PLIB_CMP_OpAmpOutputEnable function 5-1708	PLIB_DMA_ChannelXBusyActiveSet function 5-1790
PLIB_CMP_OrGateAInputDisable function 5-1708	PLIB_DMA_ChannelXBusyInActiveSet function 5-1790
PLIB_CMP_OrGateAInputEnable function 5-1709	PLIB_DMA_ChannelXBusyIsBusy function 5-1816
PLIB_CMP_OrGateBInputDisable function 5-1709	PLIB_DMA_ChannelXCellProgressPointerGet function 5-1779
PLIB_CMP_OrGateBInputEnable function 5-1710	PLIB_DMA_ChannelXCellSizeGet function 5-1779
PLIB_CMP_OrGateCInputDisable function 5-1710	PLIB_DMA_ChannelXCellSizeSet function 5-1780
PLIB_CMP_OrGateCInputEnable function 5-1711	PLIB_DMA_ChannelXChainDisable function 5-1791
PLIB_CMP_OrGateInvertedAInputDisable function 5-1711	PLIB_DMA_ChannelXChainEnable function 5-1791
PLIB_CMP_OrGateInvertedAInputEnable function 5-1712	PLIB_DMA_ChannelXChainIsEnabled function 5-1817
PLIB_CMP_OrGateInvertedBInputDisable function 5-1712	PLIB_DMA_ChannelXChainToHigher function 5-1792
PLIB_CMP_OrGateInvertedBInputEnable function 5-1713	PLIB_DMA_ChannelXChainToLower function 5-1792
PLIB_CMP_OrGateInvertedCInputDisable function 5-1713	PLIB_DMA_ChannelXCollisionStatus function 5-1817
PLIB_CMP_OrGateInvertedCInputEnable function 5-1714	PLIB_DMA_ChannelXDataSizeGet function 5-1780
PLIB_CMP_OutputDisable function 5-1688	PLIB_DMA_ChannelXDataSizeSelect function 5-1781
PLIB_CMP_OutputEnable function 5-1688	PLIB_DMA_ChannelXDestinationAddressModeGet function 5-1768
PLIB_CMP_OutputHighSelect function 5-1693	PLIB_DMA_ChannelXDestinationAddressModeSelect function 5-1768
PLIB_CMP_OutputLowSelect function 5-1693	PLIB_DMA_ChannelXDestinationPointerGet function 5-1782
PLIB_CMP_OutputStatusGet function 5-1689	PLIB_DMA_ChannelXDestinationSizeGet function 5-1782
PLIB_CMP_OutputSynchronousWithTimer1Disable function 5-1694	PLIB_DMA_ChannelXDestinationSizeSet function 5-1783
PLIB_CMP_OutputSynchronousWithTimer1Enable function 5-1694	PLIB_DMA_ChannelXDestinationStartAddressGet function 5-1774
PLIB_CMP_PositiveAndGateOutputDisable function 5-1703	PLIB_DMA_ChannelXDestinationStartAddressSet function 5-1774
PLIB_CMP_PositiveAndGateOutputEnable function 5-1703	PLIB_DMA_ChannelXDisable function 5-1793
PLIB_CMP_StopInIdleModeDisable function 5-1695	PLIB_DMA_ChannelXDisabledDisablesEvents function 5-1796
PLIB_CMP_StopInIdleModeEnable function 5-1696	PLIB_DMA_ChannelXDisabledEnablesEvents function 5-1797
plib_dma.h 5-1859	PLIB_DMA_ChannelXEnable function 5-1793
PLIB_DMA_AbortTransferSet function 5-1758	

PLIB_DMA_ChannelXEventIsDetected function 5-1818	PLIB_DMA_ChannelXStartAddressOffsetGet function 5-1777
PLIB_DMA_ChannelXINTSourceDisable function 5-1763	PLIB_DMA_ChannelXStartAddressOffsetSet function 5-1778
PLIB_DMA_ChannelXINTSourceEnable function 5-1763	PLIB_DMA_ChannelXStartIRQSet function 5-1760
PLIB_DMA_ChannelXINTSourceFlagClear function 5-1764	PLIB_DMA_ChannelXTransferCountGet function 5-1786
PLIB_DMA_ChannelXINTSourceFlagGet function 5-1764	PLIB_DMA_ChannelXTransferCountSet function 5-1787
PLIB_DMA_ChannelXINTSourceFlagSet function 5-1765	PLIB_DMA_ChannelXTransferDirectionGet function 5-1795
PLIB_DMA_ChannelXINTSourceIsEnabled function 5-1766	PLIB_DMA_ChannelXTransferDirectionSelect function 5-1796
PLIB_DMA_ChannelXIsEnabled function 5-1819	PLIB_DMA_ChannelXTriggerDisable function 5-1761
PLIB_DMA_ChannelXNullWriteModeDisable function 5-1769	PLIB_DMA_ChannelXTriggerEnable function 5-1761
PLIB_DMA_ChannelXNullWriteModeEnable function 5-1769	PLIB_DMA_ChannelXTriggerIsEnabled function 5-1762
PLIB_DMA_ChannelXNullWriteModelsEnabled function 5-1819	PLIB_DMA_CRCAppendModeDisable function 5-1801
PLIB_DMA_ChannelXOperatingTransferModeGet function 5-1770	PLIB_DMA_CRCAppendModeEnable function 5-1802
PLIB_DMA_ChannelXOperatingTransferModeSelect function 5-1770	PLIB_DMA_CRCAppendModelsEnabled function 5-1802
PLIB_DMA_ChannelXPatternDataGet function 5-1783	PLIB_DMA_CRCBitOrderSelect function 5-1803
PLIB_DMA_ChannelXPatternDataSet function 5-1784	PLIB_DMA_CRCByteOrderGet function 5-1803
PLIB_DMA_ChannelXPatternIgnoreByteDisable function 5-1797	PLIB_DMA_CRCByteOrderSelect function 5-1804
PLIB_DMA_ChannelXPatternIgnoreByteEnable function 5-1798	PLIB_DMA_CRCChannelGet function 5-1804
PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled function 5-1798	PLIB_DMA_CRCChannelSelect function 5-1805
PLIB_DMA_ChannelXPatternIgnoreGet function 5-1799	PLIB_DMA_CRCDataRead function 5-1805
PLIB_DMA_ChannelXPatternIgnoreSet function 5-1799	PLIB_DMA_CRCDataWrite function 5-1806
PLIB_DMA_ChannelXPatternLengthGet function 5-1800	PLIB_DMA_CRCDisable function 5-1806
PLIB_DMA_ChannelXPatternLengthSet function 5-1801	PLIB_DMA_CRCEnable function 5-1807
PLIB_DMA_ChannelXPeripheralAddressGet function 5-1775	PLIB_DMA_CRCIsEnabled function 5-1811
PLIB_DMA_ChannelXPeripheralAddressSet function 5-1775	PLIB_DMA_CRCPolynomialLengthGet function 5-1807
PLIB_DMA_ChannelXPingPongModeGet function 5-1820	PLIB_DMA_CRCPolynomialLengthSet function 5-1808
PLIB_DMA_ChannelXPriorityGet function 5-1794	PLIB_DMA_CRCTypeGet function 5-1808
PLIB_DMA_ChannelXPrioritySelect function 5-1794	PLIB_DMA_CRCTypeSet function 5-1809
PLIB_DMA_ChannelXReloadDisable function 5-1771	PLIB_DMA_CRCWriteByteOrderAlter function 5-1809
PLIB_DMA_ChannelXReloadEnable function 5-1771	PLIB_DMA_CRCWriteByteOrderMaintain function 5-1810
PLIB_DMA_ChannelXReloadIsEnabled function 5-1773	PLIB_DMA_CRCXOREnableGet function 5-1810
PLIB_DMA_ChannelXSourceAddressModeGet function 5-1772	PLIB_DMA_CRCXOREnableSet function 5-1810
PLIB_DMA_ChannelXSourceAddressModeSelect function 5-1772	PLIB_DMA_Disable function 5-1756
PLIB_DMA_ChannelXSourcePointerGet function 5-1785	PLIB_DMA_Enable function 5-1756
PLIB_DMA_ChannelXSourceSizeGet function 5-1785	PLIB_DMA_ExistsAbortTransfer function 5-1835
PLIB_DMA_ChannelXSourceSizeSet function 5-1786	PLIB_DMA_ExistsBusy function 5-1835
PLIB_DMA_ChannelXSourceStartAddressGet function 5-1776	PLIB_DMA_ExistsChannelBits function 5-1836
PLIB_DMA_ChannelXSourceStartAddressSet function 5-1777	PLIB_DMA_ExistsChannelX function 5-1837
	PLIB_DMA_ExistsChannelXAbortIRQ function 5-1837
	PLIB_DMA_ExistsChannelXAuto function 5-1838

PLIB_DMA_ExistsChannelXBusy function 5-1838	PLIB_DMA_IsBusy function 5-1812
PLIB_DMA_ExistsChannelXCellProgressPointer function 5-1839	PLIB_DMA_IsEnabled function 5-1812
PLIB_DMA_ExistsChannelXCellSize function 5-1839	PLIB_DMA_LastBusAccessIsRead function 5-1813
PLIB_DMA_ExistsChannelXChain function 5-1840	PLIB_DMA_LastBusAccessIsWrite function 5-1813
PLIB_DMA_ExistsChannelXChainEnbl function 5-1840	PLIB_DMA_LastChannelActive function 5-1814
PLIB_DMA_ExistsChannelXDestinationPointer function 5-1841	PLIB_DMA_RecentAddressAccessed function 5-1814
PLIB_DMA_ExistsChannelXDestinationSize function 5-1841	PLIB_DMA_StartTransferSet function 5-1759
PLIB_DMA_ExistsChannelXDestinationStartAddress function 5-1842	PLIB_DMA_StopInIdleDisable function 5-1756
PLIB_DMA_ExistsChannelXDisabled function 5-1843	PLIB_DMA_StopInIdleEnable function 5-1757
PLIB_DMA_ExistsChannelXEvent function 5-1843	PLIB_DMA_SuspendDisable function 5-1757
PLIB_DMA_ExistsChannelXINTSource function 5-1844	PLIB_DMA_SuspendEnable function 5-1758
PLIB_DMA_ExistsChannelXINTSourceFlag function 5-1844	PLIB_DMA_SuspendIsEnabled function 5-1814
PLIB_DMA_ExistsChannelXPatternData function 5-1845	plib_ebi.h 5-1900
PLIB_DMA_ExistsChannelXPatternIgnore function 5-1845	PLIB_EBI_AddressHoldTimeGet function 5-1869
PLIB_DMA_ExistsChannelXPatternIgnoreByte function 5-1846	PLIB_EBI_AddressPinEnableBitsSet function 5-1876
PLIB_DMA_ExistsChannelXPatternLength function 5-1846	PLIB_EBI_AddressSetupTimeGet function 5-1870
PLIB_DMA_ExistsChannelXPriority function 5-1847	PLIB_EBI_BaseAddressGet function 5-1870
PLIB_DMA_ExistsChannelXSourcePointer function 5-1847	PLIB_EBI_BaseAddressSet function 5-1876
PLIB_DMA_ExistsChannelXSourceSize function 5-1848	PLIB_EBI_ByteSelectPinSet function 5-1877
PLIB_DMA_ExistsChannelXSourceStartAddress function 5-1848	PLIB_EBI_ChipSelectEnableSet function 5-1878
PLIB_DMA_ExistsChannelXStartIRQ function 5-1849	PLIB_EBI_ControlEnableClear function 5-1869
PLIB_DMA_ExistsChannelXTrigger function 5-1849	PLIB_EBI_ControlEnableGet function 5-1870
PLIB_DMA_ExistsCRC function 5-1850	PLIB_EBI_ControlEnableSet function 5-1886
PLIB_DMA_ExistsCRCAppendMode function 5-1850	PLIB_EBI_DataEnableSet function 5-1878
PLIB_DMA_ExistsCRCBitOrder function 5-1851	PLIB_EBI_DataTurnAroundTimeGet function 5-1871
PLIB_DMA_ExistsCRCByteOrder function 5-1852	PLIB_EBI_ExistsAddressHoldTime function 5-1887
PLIB_DMA_ExistsCRCChannel function 5-1852	PLIB_EBI_ExistsAddressPinEnableBits function 5-1887
PLIB_DMA_ExistsCRCData function 5-1853	PLIB_EBI_ExistsAddressSetupTime function 5-1888
PLIB_DMA_ExistsCRCPolynomialLength function 5-1853	PLIB_EBI_ExistsBaseAddress function 5-1888
PLIB_DMA_ExistsCRCType function 5-1854	PLIB_EBI_ExistsByteSelectPin function 5-1889
PLIB_DMA_ExistsCRCWriteByteOrder function 5-1854	PLIB_EBI_ExistsChipSelectEnable function 5-1889
PLIB_DMA_ExistsCRCXOREnable function 5-1855	PLIB_EBI_ExistsControlEnable function 5-1890
PLIB_DMA_ExistsEnableControl function 5-1855	PLIB_EBI_ExistsDataEnable function 5-1890
PLIB_DMA_ExistsLastBusAccess function 5-1856	PLIB_EBI_ExistsDataTurnAroundTime function 5-1891
PLIB_DMA_ExistsRecentAddress function 5-1856	PLIB_EBI_ExistsFlashPowerDownMode function 5-1891
PLIB_DMA_ExistsStartTransfer function 5-1857	PLIB_EBI_ExistsFlashResetPin function 5-1892
PLIB_DMA_ExistsStopInIdle function 5-1857	PLIB_EBI_ExistsFlashTiming function 5-1892
PLIB_DMA_ExistsSuspend function 5-1858	PLIB_EBI_ExistsMemoryCharacteristics function 5-1893
	PLIB_EBI_ExistsMemoryPaging function 5-1893

PLIB_EBI_ExistsMemoryTimingConfig function 5-1894	PLIB_ETH_AutoDetectPadGet function 5-1929
PLIB_EBI_ExistsPageMode function 5-1894	PLIB_ETH_AutoDetectPadSet function 5-1930
PLIB_EBI_ExistsPageReadTime function 5-1895	PLIB_ETH_AutoFlowControlDisable function 5-1998
PLIB_EBI_ExistsReadCycleTime function 5-1895	PLIB_ETH_AutoFlowControlEnable function 5-1999
PLIB_EBI_ExistsReadyMode function 5-1896	PLIB_ETH_AutoFlowControlsEnabled function 5-1999
PLIB_EBI_ExistsReadyPin1Config function 5-1896	PLIB_ETH_BackPresNoBackoffDisable function 5-2000
PLIB_EBI_ExistsReadyPin2Config function 5-1897	PLIB_ETH_BackPresNoBackoffEnable function 5-2000
PLIB_EBI_ExistsReadyPin3Config function 5-1897	PLIB_ETH_BackPresNoBackoffsEnabled function 5-2001
PLIB_EBI_ExistsReadyPinSens function 5-1898	PLIB_ETH_BackToBackIPGGet function 5-1930
PLIB_EBI_ExistsStaticMemoryWidthRegister function 5-1898	PLIB_ETH_BackToBackIPGSet function 5-1931
PLIB_EBI_ExistsWriteOutputControl function 5-1899	PLIB_ETH_CollisionWindowGet function 5-1932
PLIB_EBI_ExistsWritePulseWidth function 5-1899	PLIB_ETH_CollisionWindowSet function 5-1932
PLIB_EBI_FlashPowerDownModeGet function 5-1871	PLIB_ETH_CRCDisable function 5-1982
PLIB_EBI_FlashPowerDownModeSet function 5-1879	PLIB_ETH_CRCEnable function 5-1983
PLIB_EBI_FlashResetPinGet function 5-1872	PLIB_ETH_CRCIsEnabled function 5-1983
PLIB_EBI_FlashResetPinSet function 5-1879	PLIB_ETH_DataNotValid function 5-1984
PLIB_EBI_FlashTimingGet function 5-1872	PLIB_ETH_DelayedCRCDisable function 5-1933
PLIB_EBI_FlashTimingSet function 5-1880	PLIB_ETH_DelayedCRCEnable function 5-1933
PLIB_EBI_MemoryCharacteristicsSet function 5-1880	PLIB_ETH_DelayedCRCIsEnabled function 5-1934
PLIB_EBI_MemoryPagingSet function 5-1881	PLIB_ETH_Disable function 5-1934
PLIB_EBI_MemoryTimingConfigSet function 5-1882	PLIB_ETH_Enable function 5-1935
PLIB_EBI_PageModeGet function 5-1873	PLIB_ETH_EthernetIsBusy function 5-1984
PLIB_EBI_PageReadCycleTimeGet function 5-1873	PLIB_ETH_ExcessDeferDisable function 5-1935
PLIB_EBI_PageSizeGet function 5-1873	PLIB_ETH_ExcessDeferEnable function 5-1936
PLIB_EBI_ReadCycleTimeGet function 5-1874	PLIB_ETH_ExcessDeferIsEnabled function 5-1936
PLIB_EBI_ReadyModeGet function 5-1874	PLIB_ETH_ExistsAlignmentErrorCount function 5-2026
PLIB_EBI_ReadyModeSet function 5-1886	PLIB_ETH_ExistsAutoFlowControl function 5-2027
PLIB_EBI_ReadyPin1ConfigSet function 5-1882	PLIB_ETH_ExistsCollisionCounts function 5-2028
PLIB_EBI_ReadyPin2ConfigSet function 5-1883	PLIB_ETH_ExistsCollisionWindow function 5-2028
PLIB_EBI_ReadyPin3ConfigSet function 5-1884	PLIB_ETH_ExistsEnable function 5-2029
PLIB_EBI_ReadyPinSensSet function 5-1884	PLIB_ETH_ExistsEthernetControllerStatus function 5-2029
PLIB_EBI_StaticMemoryWidthRegisterGet function 5-1875	PLIB_ETH_ExistsFCSErrorCount function 5-2030
PLIB_EBI_StaticMemoryWidthRegisterSet function 5-1885	PLIB_ETH_ExistsFramesTransmittedOK function 5-2030
PLIB_EBI_WriteOutputControlSet function 5-1885	PLIB_ETH_ExistsFrameRxReceivedOK function 5-2031
PLIB_EBI_WritePulseWidthGet function 5-1875	PLIB_ETH_ExistsHashTable function 5-2031
plib_eth.h 5-2052	PLIB_ETH_ExistsInterPacketGaps function 5-2032
PLIB_ETH_AlignErrorCountClear function 5-2019	PLIB_ETH_ExistsInterrupt function 5-2033
PLIB_ETH_AlignErrorCountGet function 5-2019	PLIB_ETH_ExistsMAC_Configuration function 5-2033
PLIB_ETH_AutoDetectPadClear function 5-1929	PLIB_ETH_ExistsMAC_Resets function 5-2035

PLIB_ETH_ExistsMAC_Testing function 5-2036	PLIB_ETH_HugeFrameEnable function 5-1940
PLIB_ETH_ExistsManualFlowControl function 5-2037	PLIB_ETH_HugeFrameIsEnabled function 5-1941
PLIB_ETH_ExistsMaxFrameLength function 5-2037	PLIB_ETH_InterruptClear function 5-2014
PLIB_ETH_ExistsMIIM_Config function 5-2038	PLIB_ETH_InterruptSet function 5-2015
PLIB_ETH_ExistsMIIM_Indicators function 5-2038	PLIB_ETH_InterruptsGet function 5-2015
PLIB_ETH_ExistsMIIMAddresses function 5-2039	PLIB_ETH_InterruptSourceDisable function 5-2016
PLIB_ETH_ExistsMIIMReadWrite function 5-2040	PLIB_ETH_InterruptSourceEnable function 5-2016
PLIB_ETH_ExistsMIIMScanMode function 5-2040	PLIB_ETH_InterruptSourceIsEnabled function 5-2017
PLIB_ETH_ExistsMIIMWriteReadData function 5-2041	PLIB_ETH_InterruptSourcesGet function 5-2018
PLIB_ETH_ExistsPatternMatch function 5-2041	PLIB_ETH_InterruptStatusGet function 5-2018
PLIB_ETH_ExistsPauseTimer function 5-2042	PLIB_ETH_IsEnabled function 5-1941
PLIB_ETH_ExistsReceiveBufferSize function 5-2042	PLIB_ETH_LinkHasFailed function 5-1985
PLIB_ETH_ExistsReceiveFilters function 5-2043	PLIB_ETH_LongPreambleDisable function 5-1942
PLIB_ETH_ExistsReceiveOverflowCount function 5-2043	PLIB_ETH_LongPreambleEnable function 5-1942
PLIB_ETH_ExistsReceiveWmarks function 5-2044	PLIB_ETH_LongPreambleIsEnabled function 5-1943
PLIB_ETH_ExistsRetransmissionMaximum function 5-2045	PLIB_ETH_LoopbackDisable function 5-1943
PLIB_ETH_ExistsRMII_Support function 5-2045	PLIB_ETH_LoopbackEnable function 5-1944
PLIB_ETH_ExistsRxBufferCountDecrement function 5-2046	PLIB_ETH_LoopbackIsEnabled function 5-1944
PLIB_ETH_ExistsRxEnable function 5-2046	PLIB_ETH_ManualFlowControlDisable function 5-2001
PLIB_ETH_ExistsRxPacketDescriptorAddress function 5-2047	PLIB_ETH_ManualFlowControlEnable function 5-2002
PLIB_ETH_ExistsStationAddress function 5-2047	PLIB_ETH_ManualFlowControlIsEnabled function 5-2002
PLIB_ETH_ExistsStopInIdle function 5-2048	PLIB_ETH_MaxFrameLengthGet function 5-1945
PLIB_ETH_ExistsTransmitRTS function 5-2048	PLIB_ETH_MaxFrameLengthSet function 5-1945
PLIB_ETH_ExistsTxPacketDescriptorAddress function 5-2049	PLIB_ETH_MCSRxResetDisable function 5-1957
PLIB_ETH_FCSErrorCountClear function 5-2020	PLIB_ETH_MCSRxResetEnable function 5-1958
PLIB_ETH_FCSErrorCountGet function 5-2020	PLIB_ETH_MCSRxResetIsEnabled function 5-1958
PLIB_ETH_FrameLengthCheckDisable function 5-1937	PLIB_ETH_MCSTxResetDisable function 5-1959
PLIB_ETH_FrameLengthCheckEnable function 5-1937	PLIB_ETH_MCSTxResetEnable function 5-1959
PLIB_ETH_FrameLengthCheckIsEnabled function 5-1938	PLIB_ETH_MCSTxResetIsEnabled function 5-1960
PLIB_ETH_FramesRxdOkCountClear function 5-2021	PLIB_ETH_MIIMClockGet function 5-1946
PLIB_ETH_FramesRxdOkCountGet function 5-2021	PLIB_ETH_MIIMClockSet function 5-1946
PLIB_ETH_FramesTxdOkCountClear function 5-2022	PLIB_ETH_MIIMIsBusy function 5-1985
PLIB_ETH_FramesTxdOkCountGet function 5-2022	PLIB_ETH_MIIMIsScanning function 5-1986
PLIB_ETH_FullDuplexDisable function 5-1938	PLIB_ETH_MIIMNoPreDisable function 5-1947
PLIB_ETH_FullDuplexEnable function 5-1939	PLIB_ETH_MIIMNoPreEnable function 5-1947
PLIB_ETH_FullDuplexIsEnabled function 5-1939	PLIB_ETH_MIIMNoPreIsEnabled function 5-1948
PLIB_ETH_HashTableGet function 5-1987	PLIB_ETH_MIIMReadDataGet function 5-1960
PLIB_ETH_HashTableSet function 5-1988	PLIB_ETH_MIIMReadStart function 5-1961
PLIB_ETH_HugeFrameDisable function 5-1940	PLIB_ETH_MIIMResetDisable function 5-1961

PLIB_ETH_MIIMResetEnable function 5-1962	PLIB_ETH_PurePreambleIsEnabled function 5-1995
PLIB_ETH_MIIMResetIsEnabled function 5-1962	PLIB_ETH_ReceiveBufferSizeGet function 5-1951
PLIB_ETH_MIIMScanIncrDisable function 5-1963	PLIB_ETH_ReceiveBufferSizeSet function 5-1952
PLIB_ETH_MIIMScanIncrEnable function 5-1963	PLIB_ETH_ReceiveDisable function 5-1952
PLIB_ETH_MIIMScanIncrIsEnabled function 5-1964	PLIB_ETH_ReceiveEnable function 5-1953
PLIB_ETH_MIIMScanModeDisable function 5-1964	PLIB_ETH_ReceiveFilterDisable function 5-1995
PLIB_ETH_MIIMScanModeEnable function 5-1965	PLIB_ETH_ReceiveFilterEnable function 5-1996
PLIB_ETH_MIIMScanModelsEnabled function 5-1965	PLIB_ETH_ReceiveFilterIsEnabled function 5-1997
PLIB_ETH_MIIMWriteDataSet function 5-1966	PLIB_ETH_ReceiveIsBusy function 5-1986
PLIB_ETH_MIIMWriteStart function 5-1966	PLIB_ETH_ReceiveIsEnabled function 5-1953
PLIB_ETH_MIIRResetDisable function 5-1967	PLIB_ETH_RegisterAddressGet function 5-1970
PLIB_ETH_MIIRResetEnable function 5-1968	PLIB_ETH_RegisterAddressSet function 5-1970
PLIB_ETH_MIIRResetIsEnabled function 5-1968	PLIB_ETH_ReTxMaxGet function 5-1954
PLIB_ETH_MultipleCollisionCountClear function 5-2023	PLIB_ETH_ReTxMaxSet function 5-1954
PLIB_ETH_MultipleCollisionCountGet function 5-2023	PLIB_ETH_RMIIResetDisable function 5-1971
PLIB_ETH_NoBackoffDisable function 5-2003	PLIB_ETH_RMIIResetEnable function 5-1971
PLIB_ETH_NoBackoffEnable function 5-2003	PLIB_ETH_RMIIResetIsEnabled function 5-1972
PLIB_ETH_NoBackoffIsEnabled function 5-2004	PLIB_ETH_RMIIISpeedGet function 5-1955
PLIB_ETH_NonBackToBackIPG1Get function 5-1948	PLIB_ETH_RMIIISpeedSet function 5-1955
PLIB_ETH_NonBackToBackIPG1Set function 5-1949	PLIB_ETH_RxBufferCountDecrement function 5-1972
PLIB_ETH_NonBackToBackIPG2Get function 5-1949	PLIB_ETH_RxDisable function 5-1973
PLIB_ETH_NonBackToBackIPG2Set function 5-1950	PLIB_ETH_RxEmptyWmarkGet function 5-2004
PLIB_ETH_PassAllDisable function 5-1988	PLIB_ETH_RxEmptyWmarkSet function 5-2005
PLIB_ETH_PassAllEnable function 5-1989	PLIB_ETH_RxEnable function 5-1973
PLIB_ETH_PassAllIsEnabled function 5-1989	PLIB_ETH_RxFullWmarkGet function 5-2006
PLIB_ETH_PatternMatchChecksumGet function 5-1990	PLIB_ETH_RxFullWmarkSet function 5-2006
PLIB_ETH_PatternMatchChecksumSet function 5-1990	PLIB_ETH_RxFuncResetDisable function 5-1974
PLIB_ETH_PatternMatchGet function 5-1991	PLIB_ETH_RxFuncResetEnable function 5-1974
PLIB_ETH_PatternMatchModeGet function 5-1991	PLIB_ETH_RxFuncResetIsEnabled function 5-1975
PLIB_ETH_PatternMatchModeSet function 5-1992	PLIB_ETH_RxIsEnabled function 5-1975
PLIB_ETH_PatternMatchOffsetGet function 5-1992	PLIB_ETH_RxOverflowCountClear function 5-2024
PLIB_ETH_PatternMatchOffsetSet function 5-1993	PLIB_ETH_RxOverflowCountGet function 5-2024
PLIB_ETH_PatternMatchSet function 5-1993	PLIB_ETH_RxPacketCountGet function 5-2025
PLIB_ETH_PauseTimerGet function 5-1950	PLIB_ETH_RxPacketDescAddrGet function 5-1976
PLIB_ETH_PauseTimerSet function 5-1951	PLIB_ETH_RxPacketDescAddrSet function 5-1976
PLIB_ETH_PHYAddressGet function 5-1969	PLIB_ETH_RxPauseDisable function 5-2007
PLIB_ETH_PHYAddressSet function 5-1969	PLIB_ETH_RxPauseEnable function 5-2007
PLIB_ETH_PurePreambleDisable function 5-1994	PLIB_ETH_RxPausIsEnabled function 5-2008
PLIB_ETH_PurePreambleEnable function 5-1994	PLIB_ETH_ShortcutQuantaDisable function 5-2008

PLIB_ETH_ShortcutQuantaEnable function 5-2009	PLIB_I2C_Enable function 5-2094
PLIB_ETH_ShortcutQuantalsEnabled function 5-2009	PLIB_I2C_ExistsAcksequenceProgress function 5-2144
PLIB_ETH_SimResetDisable function 5-2010	PLIB_I2C_ExistsArbitrationLoss function 5-2144
PLIB_ETH_SimResetEnable function 5-2010	PLIB_I2C_ExistsBaudRate function 5-2145
PLIB_ETH_SimResetIsEnabled function 5-2011	PLIB_I2C_ExistsBusIdle function 5-2146
PLIB_ETH_SingleCollisionCountClear function 5-2025	PLIB_I2C_ExistsClockStretching function 5-2146
PLIB_ETH_SingleCollisionCountGet function 5-2026	PLIB_I2C_ExistsDataLineHoldTime function 5-2147
PLIB_ETH_StationAddressGet function 5-1997	PLIB_I2C_ExistsGeneralCall function 5-2147
PLIB_ETH_StationAddressSet function 5-1998	PLIB_I2C_ExistsGeneralCallAddressDetect function 5-2148
PLIB_ETH_StopInIdleDisable function 5-1956	PLIB_I2C_ExistsHighFrequency function 5-2148
PLIB_ETH_StopInIdleEnable function 5-1956	PLIB_I2C_ExistsIPMI function 5-2149
PLIB_ETH_StopInIdleIsEnabled function 5-1957	PLIB_I2C_ExistsMasterReceiverClock1Byte function 5-2149
PLIB_ETH_TestBackPressDisable function 5-2011	PLIB_I2C_ExistsMasterStart function 5-2150
PLIB_ETH_TestBackPressEnable function 5-2012	PLIB_I2C_ExistsMasterStartRepeat function 5-2150
PLIB_ETH_TestBackPressIsEnabled function 5-2012	PLIB_I2C_ExistsMasterStop function 5-2151
PLIB_ETH_TestPauseDisable function 5-1977	PLIB_I2C_ExistsModuleEnable function 5-2151
PLIB_ETH_TestPauseEnable function 5-1977	PLIB_I2C_ExistsReceivedByteAcknowledge function 5-2152
PLIB_ETH_TestPauseIsEnabled function 5-1978	PLIB_I2C_ExistsReceivedByteAvailable function 5-2152
PLIB_ETH_TransmitsBusy function 5-1987	PLIB_I2C_ExistsReceivedByteGet function 5-2153
PLIB_ETH_TxFuncResetDisable function 5-1978	PLIB_I2C_ExistsReceiverOverflow function 5-2153
PLIB_ETH_TxFuncResetEnable function 5-1979	PLIB_I2C_ExistsReservedAddressProtect function 5-2154
PLIB_ETH_TxFuncResetIsEnabled function 5-1979	PLIB_I2C_ExistsSlaveAddress10Bit function 5-2154
PLIB_ETH_TxPacketDescAddrGet function 5-1980	PLIB_I2C_ExistsSlaveAddress7Bit function 5-2155
PLIB_ETH_TxPacketDescAddrSet function 5-1980	PLIB_I2C_ExistsSlaveAddressDetect function 5-2155
PLIB_ETH_TxPauseDisable function 5-2013	PLIB_I2C_ExistsSlaveAddressHoldEnable function 5-2156
PLIB_ETH_TxPauseEnable function 5-2013	PLIB_I2C_ExistsSlaveBufferOverwrite function 5-2156
PLIB_ETH_TxPauseIsEnabled function 5-2014	PLIB_I2C_ExistsSlaveBusCollisionDetect function 5-2157
PLIB_ETH_TxRTSDisable function 5-1981	PLIB_I2C_ExistsSlaveClockHold function 5-2157
PLIB_ETH_TxRTSEnable function 5-1981	PLIB_I2C_ExistsSlaveDataDetect function 5-2158
PLIB_ETH_TxRTSIsEnabled function 5-1982	PLIB_I2C_ExistsSlaveInterruptOnStart function 5-2158
plib_i2c.h 5-2166	PLIB_I2C_ExistsSlaveInterruptOnStop function 5-2159
PLIB_I2C_AcksequenceIsInProgress function 5-2122	PLIB_I2C_ExistsSlaveMask function 5-2160
PLIB_I2C_ArbitrationLossClear function 5-2103	PLIB_I2C_ExistsSlaveReadRequest function 5-2160
PLIB_I2C_ArbitrationLossHasOccurred function 5-2104	PLIB_I2C_ExistsSMBus function 5-2161
PLIB_I2C_BaudRateGet function 5-2109	PLIB_I2C_ExistsStartDetect function 5-2161
PLIB_I2C_BaudRateSet function 5-2109	PLIB_I2C_ExistsStopDetect function 5-2162
PLIB_I2C_BusIdle function 5-2105	PLIB_I2C_ExistsStopInIdle function 5-2162
PLIB_I2C_DataLineHoldTimeSet function 5-2123	PLIB_I2C_ExistsTransmitterByteAcknowledge function 5-2163
PLIB_I2C_Disable function 5-2093	PLIB_I2C_ExistsTransmitterByteComplete function 5-2163

PLIB_I2C_ExistsTransmitterByteSend function 5-2164	PLIB_I2C_SlaveClockStretchingDisable function 5-2099
PLIB_I2C_ExistsTransmitterIsBusy function 5-2164	PLIB_I2C_SlaveClockStretchingEnable function 5-2100
PLIB_I2C_ExistsTransmitterOverflow function 5-2165	PLIB_I2C_SlaveDataHoldDisable function 5-2127
PLIB_I2C_GeneralCallDisable function 5-2094	PLIB_I2C_SlaveDataHoldEnable function 5-2128
PLIB_I2C_GeneralCallEnable function 5-2095	PLIB_I2C_SlaveDataIsDetected function 5-2117
PLIB_I2C_HighFrequencyDisable function 5-2096	PLIB_I2C_SlaveInterruptOnStartDisable function 5-2129
PLIB_I2C_HighFrequencyEnable function 5-2096	PLIB_I2C_SlaveInterruptOnStartEnable function 5-2129
PLIB_I2C_IPMIDisable function 5-2097	PLIB_I2C_SlaveInterruptOnStopDisable function 5-2130
PLIB_I2C_IPMIEnable function 5-2098	PLIB_I2C_SlaveInterruptOnStopEnable function 5-2130
PLIB_I2C_MasterReceiverClock1Byte function 5-2131	PLIB_I2C_SlaveMask10BitGet function 5-2119
PLIB_I2C_MasterReceiverReadyToAcknowledge function 5-2143	PLIB_I2C_SlaveMask10BitSet function 5-2120
PLIB_I2C_MasterStart function 5-2132	PLIB_I2C_SlaveMask7BitGet function 5-2121
PLIB_I2C_MasterStartRepeat function 5-2132	PLIB_I2C_SlaveMask7BitSet function 5-2121
PLIB_I2C_MasterStop function 5-2133	PLIB_I2C_SlaveReadIsRequested function 5-2118
PLIB_I2C_ReceivedByteAcknowledge function 5-2139	PLIB_I2C_SMBDisable function 5-2101
PLIB_I2C_ReceivedByteGet function 5-2140	PLIB_I2C_SMBEnable function 5-2101
PLIB_I2C_ReceivedBytesAvailable function 5-2140	PLIB_I2C_StartClear function 5-2106
PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function 5-2141	PLIB_I2C_StartWasDetected function 5-2106
PLIB_I2C_ReceiverOverflowClear function 5-2142	PLIB_I2C_StopClear function 5-2107
PLIB_I2C_ReceiverOverflowHasOccurred function 5-2142	PLIB_I2C_StopInIdleDisable function 5-2102
PLIB_I2C_ReservedAddressProtectDisable function 5-2098	PLIB_I2C_StopInIdleEnable function 5-2103
PLIB_I2C_ReservedAddressProtectEnable function 5-2099	PLIB_I2C_StopWasDetected function 5-2108
PLIB_I2C_SlaveAddress10BitGet function 5-2110	PLIB_I2C_TransmitterByteHasCompleted function 5-2134
PLIB_I2C_SlaveAddress10BitSet function 5-2111	PLIB_I2C_TransmitterByteSend function 5-2134
PLIB_I2C_SlaveAddress10BitWasDetected function 5-2112	PLIB_I2C_TransmitterByteWasAcknowledged function 5-2135
PLIB_I2C_SlaveAddress7BitGet function 5-2112	PLIB_I2C_TransmitterIsBusy function 5-2136
PLIB_I2C_SlaveAddress7BitSet function 5-2113	PLIB_I2C_TransmitterIsReady function 5-2137
PLIB_I2C_SlaveAddressHoldDisable function 5-2114	PLIB_I2C_TransmitterOverflowClear function 5-2138
PLIB_I2C_SlaveAddressHoldEnable function 5-2114	PLIB_I2C_TransmitterOverflowHasOccurred function 5-2138
PLIB_I2C_SlaveAddressIsDetected function 5-2115	plib_ic.h 5-2201
PLIB_I2C_SlaveAddressIsGeneralCall function 5-2116	PLIB_IC_AlternateClockDisable function 5-2182
PLIB_I2C_SlaveAddressModels10Bits function 5-2117	PLIB_IC_AlternateClockEnable function 5-2183
PLIB_I2C_SlaveBufferOverwriteDisable function 5-2124	PLIB_IC_Buffer16BitGet function 5-2183
PLIB_I2C_SlaveBufferOverwriteEnable function 5-2124	PLIB_IC_Buffer32BitGet function 5-2184
PLIB_I2C_SlaveBusCollisionDetectDisable function 5-2125	PLIB_IC_BufferIsEmpty function 5-2184
PLIB_I2C_SlaveBusCollisionDetectEnable function 5-2125	PLIB_IC_BufferOverflowHasOccurred function 5-2185
PLIB_I2C_SlaveClockHold function 5-2126	PLIB_IC_BufferSizeSelect function 5-2185
PLIB_I2C_SlaveClockRelease function 5-2127	PLIB_IC_ClockSourceSelect function 5-2179
	PLIB_IC_Disable function 5-2177

PLIB_IC_Enable function 5-2177	PLIB_INT_ExistsLowPriority function 5-2244
PLIB_IC_EventsPerInterruptSelect function 5-2178	PLIB_INT_ExistsPeripheralControl function 5-2244
PLIB_IC_ExistsAlternateClock function 5-2193	PLIB_INT_ExistsPriority function 5-2245
PLIB_IC_ExistsBufferIsEmptyStatus function 5-2193	PLIB_INT_ExistsProximityTimerControl function 5-2245
PLIB_IC_ExistsBufferOverflowStatus function 5-2194	PLIB_INT_ExistsProximityTimerEnable function 5-2246
PLIB_IC_ExistsBufferSize function 5-2194	PLIB_INT_ExistsSingleVectorShadowSet function 5-2246
PLIB_IC_ExistsBufferValue function 5-2195	PLIB_INT_ExistsSourceControl function 5-2247
PLIB_IC_ExistsCaptureMode function 5-2195	PLIB_INT_ExistsSourceFlag function 5-2247
PLIB_IC_ExistsClockSource function 5-2196	PLIB_INT_ExistsTrapSource function 5-2248
PLIB_IC_ExistsEdgeCapture function 5-2196	PLIB_INT_ExistsVectorPriority function 5-2248
PLIB_IC_ExistsEnable function 5-2197	PLIB_INT_ExistsVectorSelect function 5-2249
PLIB_IC_ExistsEventsPerInterruptSelect function 5-2197	PLIB_INT_ExternalFallingEdgeSelect function 5-2210
PLIB_IC_ExistsStopInIdle function 5-2198	PLIB_INT_ExternalRisingEdgeSelect function 5-2211
PLIB_IC_ExistsSyncMode function 5-2198	PLIB_INT_IsEnabled function 5-2211
PLIB_IC_ExistsSyncModeInput function 5-2199	PLIB_INT_MultiVectorSelect function 5-2212
PLIB_IC_ExistsTimerSelect function 5-2199	PLIB_INT_NestingDisable function 5-2213
PLIB_IC_ExistsTimerTriggered function 5-2200	PLIB_INT_NestingEnable function 5-2213
PLIB_IC_FirstCaptureEdgeSelect function 5-2179	PLIB_INT_PeripheralsDisable function 5-2214
PLIB_IC_ModeSelect function 5-2186	PLIB_INT_PeripheralsEnable function 5-2214
PLIB_IC_StopInIdleDisable function 5-2187	PLIB_INT_PriorityDisable function 5-2214
PLIB_IC_StopInIdleEnable function 5-2187	PLIB_INT_PriorityEnable function 5-2215
PLIB_IC_SyncModeInputSelect function 5-2180	PLIB_INT_PriorityGet function 5-2216
PLIB_IC_SyncModeSelect function 5-2180	PLIB_INT_PriorityHighDisable function 5-2216
PLIB_IC_TimerIsTriggered function 5-2181	PLIB_INT_PriorityHighEnable function 5-2217
PLIB_IC_TimerSelect function 5-2181	PLIB_INT_PriorityLowDisable function 5-2217
plib_int.h 5-2250	PLIB_INT_PriorityLowEnable function 5-2218
PLIB_INT_AlternateVectorTableSelect function 5-2208	PLIB_INT_ProximityTimerDisable function 5-2218
PLIB_INT_CPUCurrentPriorityLevelGet function 5-2228	PLIB_INT_ProximityTimerEnable function 5-2219
PLIB_INT_Disable function 5-2209	PLIB_INT_ProximityTimerGet function 5-2229
PLIB_INT_DISIStatusGet function 5-2209	PLIB_INT_ProximityTimerSet function 5-2230
PLIB_INT_Enable function 5-2210	PLIB_INT_SingleVectorSelect function 5-2219
PLIB_INT_ExistsAlternateVectorTable function 5-2240	PLIB_INT_SingleVectorShadowSetDisable function 5-2220
PLIB_INT_ExistsCPUCurrentPriorityLevel function 5-2240	PLIB_INT_SingleVectorShadowSetEnable function 5-2220
PLIB_INT_ExistsEnableControl function 5-2241	PLIB_INT_SourceDisable function 5-2221
PLIB_INT_ExistsExternalINTEdgeSelect function 5-2241	PLIB_INT_SourceEnable function 5-2222
PLIB_INT_ExistsHighPriority function 5-2242	PLIB_INT_SourceFlagClear function 5-2223
PLIB_INT_ExistsINTCPUPriority function 5-2242	PLIB_INT_SourceFlagGet function 5-2224
PLIB_INT_ExistsINTCPUVector function 5-2243	PLIB_INT_SourceFlagSet function 5-2225
PLIB_INT_ExistsINTNesting function 5-2243	PLIB_INT_SourcesIsEnabled function 5-2225

PLIB_INT_StandardVectorTableSelect function 5-2221	PLIB_NVM_FlashWriteCycleHasCompleted function 5-2267
PLIB_INT_TrapSourceFlagClear function 5-2230	PLIB_NVM_FlashWriteKeySequence function 5-2267
PLIB_INT_TrapSourceFlagGet function 5-2231	PLIB_NVM_FlashWriteOperationSelect function 5-2268
PLIB_INT_VectorGet function 5-2232	PLIB_NVM_FlashWriteProtectMemoryAreaRange function 5-2268
PLIB_INT_VectorPriorityGet function 5-2226	PLIB_NVM_FlashWriteStart function 5-2269
PLIB_INT_VectorPrioritySet function 5-2227	PLIB_NVM_IsBootMemoryLocked function 5-2272
PLIB_INT_VectorSubPriorityGet function 5-2227	PLIB_NVM_IsBootPageWriteProtected function 5-2272
PLIB_INT_VectorSubPrioritySet function 5-2228	PLIB_NVM_IsProgramFlashMemoryLocked function 5-2269
plib_nvm.h 5-2285	PLIB_NVM_LockBootMemory function 5-2273
PLIB_NVM_BootPageWriteProtectionDisable function 5-2271	PLIB_NVM_LockProgramFlashMemory function 5-2274
PLIB_NVM_BootPageWriteProtectionEnable function 5-2271	PLIB_NVM_LowVoltageEventIsActive function 5-2259
PLIB_NVM_DataBlockSourceAddress function 5-2263	PLIB_NVM_LowVoltageIsDetected function 5-2259
PLIB_NVM_ExistsAccessEnable function 5-2274	PLIB_NVM_MemoryModifyEnable function 5-2260
PLIB_NVM_ExistsAddressModifyControl function 5-2275	PLIB_NVM_MemoryModifyInhibit function 5-2260
PLIB_NVM_ExistsBootPageWriteProtect function 5-2275	PLIB_NVM_MemoryOperationSelect function 5-2261
PLIB_NVM_ExistsEEPROMReadInitiate function 5-2276	PLIB_NVM_ProgramFlashBank1LowerRegion function 5-2270
PLIB_NVM_ExistsFlashBankRegionSelect function 5-2276	PLIB_NVM_ProgramFlashBank2LowerRegion function 5-2270
PLIB_NVM_ExistsFlashWPMemoryRangeProvide function 5-2277	PLIB_NVM_StopInIdleDisable function 5-2261
PLIB_NVM_ExistsKeySequence function 5-2277	PLIB_NVM_StopInIdleEnable function 5-2262
PLIB_NVM_ExistsLockBootSelect function 5-2278	PLIB_NVM_WriteOperationHasTerminated function 5-2262
PLIB_NVM_ExistsLockPFMSelect function 5-2278	plib_oc.h 5-2328
PLIB_NVM_ExistsLowVoltageError function 5-2279	PLIB_OC_AlternateClockDisable function 5-2306
PLIB_NVM_ExistsLowVoltageStatus function 5-2279	PLIB_OC_AlternateClockEnable function 5-2307
PLIB_NVM_ExistsMemoryModificationControl function 5-2280	PLIB_OC_Buffer16BitSet function 5-2299
PLIB_NVM_ExistsOperationMode function 5-2280	PLIB_OC_Buffer32BitSet function 5-2299
PLIB_NVM_ExistsProgramEraseOperation function 5-2281	PLIB_OC_BufferSizeSelect function 5-2300
PLIB_NVM_ExistsProvideData function 5-2281	PLIB_OC_Disable function 5-2296
PLIB_NVM_ExistsProvideQuadData function 5-2282	PLIB_OC_Enable function 5-2297
PLIB_NVM_ExistsSourceAddress function 5-2282	PLIB_OC_ExistsAlternateClock function 5-2318
PLIB_NVM_ExistsStopInIdle function 5-2283	PLIB_OC_ExistsBufferSize function 5-2319
PLIB_NVM_ExistsWriteErrorStatus function 5-2283	PLIB_OC_ExistsBufferValue function 5-2319
PLIB_NVM_ExistsWriteOperation function 5-2284	PLIB_OC_ExistsCompareModeSelect function 5-2320
PLIB_NVM_FlashAccessEnable function 5-2263	PLIB_OC_ExistsEnableControl function 5-2320
PLIB_NVM_FlashAddressToModify function 5-2264	PLIB_OC_ExistsFaultInput function 5-2321
PLIB_NVM_FlashEraseOperationSelect function 5-2264	PLIB_OC_ExistsFaultModeSelect function 5-2321
PLIB_NVM_FlashEraseStart function 5-2265	PLIB_OC_ExistsFaultOutSelect function 5-2322
PLIB_NVM_FlashProvideData function 5-2265	PLIB_OC_ExistsFaultStatus function 5-2322
PLIB_NVM_FlashProvideQuadData function 5-2266	PLIB_OC_ExistsFaultTristateSelect function 5-2323
PLIB_NVM_FlashRead function 5-2266	PLIB_OC_ExistsPolarityInvert function 5-2323

PLIB_OC_ExistsPulseWidth function 5-2324	PLIB_OSC_DozeModeEnable function 5-2368
PLIB_OC_ExistsPWMDutyCycleResolutionControl function 5-2324	PLIB_OSC_DozeRatioGet function 5-2369
PLIB_OC_ExistsStopInIdle function 5-2325	PLIB_OSC_DozeRatioSelect function 5-2369
PLIB_OC_ExistsSyncModeSelect function 5-2325	PLIB_OSC_DozeRecoverOnInterruptDisable function 5-2370
PLIB_OC_ExistsSyncSourceSelect function 5-2326	PLIB_OSC_DozeRecoverOnInterruptEnable function 5-2370
PLIB_OC_ExistsTimerSelect function 5-2326	PLIB_OSC_DozeRecoverOnInterruptIsEnabled function 5-2371
PLIB_OC_ExistsTimerTriggered function 5-2327	PLIB_OSC_ExistsClockFail function 5-2408
PLIB_OC_ExistsTriggerControl function 5-2327	PLIB_OSC_ExistsFRCDivisor function 5-2408
PLIB_OC_ExistsTriggerStatusModeSelect function 5-2328	PLIB_OSC_ExistsFRCTuning function 5-2409
PLIB_OC_FaultHasOccurred function 5-2309	PLIB_OSC_ExistsOnWaitAction function 5-2409
PLIB_OC_FaultInputSelect function 5-2309	PLIB_OSC_ExistsOscCurrentGet function 5-2410
PLIB_OC_FaultModeSelect function 5-2310	PLIB_OSC_ExistsOscSelect function 5-2410
PLIB_OC_FaultOutSelect function 5-2310	PLIB_OSC_ExistsOscSwitchInit function 5-2411
PLIB_OC_FaultTristateSelect function 5-2311	PLIB_OSC_ExistsPBClockDivisor function 5-2411
PLIB_OC_IsTriggered function 5-2302	PLIB_OSC_ExistsPBClockOutputEnable function 5-2412
PLIB_OC_ModeSelect function 5-2297	PLIB_OSC_ExistsPBClockReady function 5-2412
PLIB_OC_PolarityInvertedDisable function 5-2298	PLIB_OSC_ExistsPLLClockLock function 5-2413
PLIB_OC_PolarityInvertedEnable function 5-2298	PLIB_OSC_ExistsPLLLockStatus function 5-2413
PLIB_OC_PulseWidth16BitSet function 5-2301	PLIB_OSC_ExistsReferenceOscBaseClock function 5-2414
PLIB_OC_PulseWidth32BitSet function 5-2301	PLIB_OSC_ExistsReferenceOscChange function 5-2414
PLIB_OC_PWMDutyCycleResolutionSet function 5-2302	PLIB_OSC_ExistsReferenceOscChangeActive function 5-2415
PLIB_OC_StopInIdleDisable function 5-2307	PLIB_OSC_ExistsReferenceOscDivisor function 5-2415
PLIB_OC_StopInIdleEnable function 5-2308	PLIB_OSC_ExistsReferenceOscEnable function 5-2416
PLIB_OC_SyncModeSelect function 5-2303	PLIB_OSC_ExistsReferenceOscStopInIdleEnable function 5-2416
PLIB_OC_SyncSourceSelect function 5-2304	PLIB_OSC_ExistsReferenceOscStopInSleep function 5-2417
PLIB_OC_TimerSelect function 5-2304	PLIB_OSC_ExistsReferenceOscTrim function 5-2417
PLIB_OC_TriggerClear function 5-2305	PLIB_OSC_ExistsReferenceOutputEnable function 5-2418
PLIB_OC_TriggerSet function 5-2305	PLIB_OSC_ExistsSecondaryEnable function 5-2419
PLIB_OC_TriggerStatusModeSelect function 5-2306	PLIB_OSC_ExistsSecondaryReady function 5-2419
plib_osc.h 5-2423	PLIB_OSC_ExistsSysPLLFrequencyRange function 5-2420
PLIB_OSC_AuxClockSourceGet function 5-2347	PLIB_OSC_ExistsSysPLLInputClockSource function 5-2420
PLIB_OSC_AuxClockSourceSet function 5-2347	PLIB_OSC_ExistsSysPLLInputDivisor function 5-2421
PLIB_OSC_AuxModeGet function 5-2348	PLIB_OSC_ExistsSysPLLMultiplier function 5-2421
PLIB_OSC_AuxModeSelect function 5-2348	PLIB_OSC_ExistsSysPLLOutputDivisor function 5-2422
PLIB_OSC_ClockHasFailed function 5-2392	PLIB_OSC_ExistsUsbClockSource function 5-2422
PLIB_OSC_ClockSwitchingAbort function 5-2365	PLIB_OSC_FRCDitherEnable function 5-2359
PLIB_OSC_ClockSwitchingIsComplete function 5-2366	PLIB_OSC_FRCDivisorGet function 5-2360
PLIB_OSC_CurrentSysClockGet function 5-2366	PLIB_OSC_FRCDivisorSelect function 5-2360
PLIB_OSC_DozeModeDisable function 5-2368	PLIB_OSC_FRCTuningModeGet function 5-2361

PLIB_OSC_FRCTuningModeSet function 5-2361	PLIB_OSC_ReferenceOscDivisorValueSet function 5-2350
PLIB_OSC_FRCTuningSelect function 5-2362	PLIB_OSC_ReferenceOscEnable function 5-2351
PLIB_OSC_FRCTuningSequenceValueGet function 5-2362	PLIB_OSC_ReferenceOscIsEnabled function 5-2351
PLIB_OSC_FRCTuningSequenceValueSet function 5-2363	PLIB_OSC_ReferenceOscSourceChangeIsActive function 5-2352
PLIB_OSC_GraphicsClockDivisorGet function 5-2371	PLIB_OSC_ReferenceOscStopInIdleDisable function 5-2353
PLIB_OSC_GraphicsClockDivisorSet function 5-2372	PLIB_OSC_ReferenceOscStopInIdleEnable function 5-2353
PLIB_OSC_GraphicsClockSourceGet function 5-2372	PLIB_OSC_ReferenceOscStopInIdleIsEnabled function 5-2354
PLIB_OSC_GraphicsClockSourceSelect function 5-2373	PLIB_OSC_ReferenceOscStopInSleepDisable function 5-2354
PLIB_OSC_LinearFeedbackShiftRegGet function 5-2364	PLIB_OSC_ReferenceOscStopInSleepEnable function 5-2355
PLIB_OSC_LinearFeedbackShiftRegSet function 5-2365	PLIB_OSC_ReferenceOscStopInSleepIsEnabled function 5-2356
PLIB_OSC_OnWaitActionGet function 5-2342	PLIB_OSC_ReferenceOscSwitchIsComplete function 5-2356
PLIB_OSC_OnWaitActionSet function 5-2342	PLIB_OSC_ReferenceOscTrimSet function 5-2357
PLIB_OSC_PBClockDivisorGet function 5-2388	PLIB_OSC_ReferenceOutputDisable function 5-2357
PLIB_OSC_PBClockDivisorIsReady function 5-2389	PLIB_OSC_ReferenceOutputEnable function 5-2358
PLIB_OSC_PBClockDivisorSet function 5-2390	PLIB_OSC_ReferenceOutputIsEnabled function 5-2358
PLIB_OSC_PBOutputClockDisable function 5-2390	PLIB_OSC_SecondaryDisable function 5-2345
PLIB_OSC_PBOutputClockEnable function 5-2391	PLIB_OSC_SecondaryEnable function 5-2345
PLIB_OSC_PBOutputClockIsEnabled function 5-2391	PLIB_OSC_SecondaryIsEnabled function 5-2346
PLIB_OSC_PLLAuxClockSourceGet function 5-2374	PLIB_OSC_SecondaryIsReady function 5-2346
PLIB_OSC_PLLAuxClockSourceSelect function 5-2375	PLIB_OSC_StartupTimerHasExpired function 5-2343
PLIB_OSC_PLLAuxInputDivisorGet function 5-2375	PLIB_OSC_SysClockSelect function 5-2367
PLIB_OSC_PLLAuxInputDivisorSelect function 5-2376	PLIB_OSC_SysPLLFrequencyRangeGet function 5-2383
PLIB_OSC_PLLAuxIsLocked function 5-2376	PLIB_OSC_SysPLLFrequencyRangeSet function 5-2383
PLIB_OSC_PLLAuxMultiplierGet function 5-2377	PLIB_OSC_SysPLLInputClockSourceGet function 5-2384
PLIB_OSC_PLLAuxMultiplierSelect function 5-2378	PLIB_OSC_SysPLLInputClockSourceSet function 5-2384
PLIB_OSC_PLLAuxOutputDivisorGet function 5-2378	PLIB_OSC_SysPLLInputDivisorGet function 5-2385
PLIB_OSC_PLLAuxOutputDivisorSet function 5-2379	PLIB_OSC_SysPLLInputDivisorSet function 5-2386
PLIB_OSC_PLLClockIsLocked function 5-2379	PLIB_OSC_SysPLLMultiplierGet function 5-2386
PLIB_OSC_PLLClockLock function 5-2380	PLIB_OSC_SysPLLMultiplierSelect function 5-2387
PLIB_OSC_PLLClockUnlock function 5-2380	PLIB_OSC_SysPLLOutputDivisorGet function 5-2387
PLIB_OSC_PLLEnable function 5-2381	PLIB_OSC_SysPLLOutputDivisorSet function 5-2388
PLIB_OSC_PLLEnabled function 5-2382	PLIB_OSC_UsbClockSourceGet function 5-2373
PLIB_OSC_PLLEnabled function 5-2382	PLIB_OSC_UsbClockSourceSelect function 5-2374
PLIB_OSC_PLLEnabled function 5-2382	plib_pcache.h 5-2475
PLIB_OSC_PrimOscInSleepModeDisable function 5-2343	PLIB_PCACHE_CacheHitRead function 5-2452
PLIB_OSC_PrimOscInSleepModeEnable function 5-2344	PLIB_PCACHE_CacheHitWrite function 5-2452
PLIB_OSC_PrimOscInSleepModeIsEnabled function 5-2344	PLIB_PCACHE_CacheLineAddrGet function 5-2441
PLIB_OSC_ReferenceOscBaseClockSelect function 5-2349	PLIB_PCACHE_CacheLineAddrSet function 5-2441
PLIB_OSC_ReferenceOscDisable function 5-2350	

PLIB_PCACHE_CacheLineData function 5-2442	PLIB_PCACHE_ExistsWaitState function 5-2474
PLIB_PCACHE_CacheLineDeselect function 5-2442	PLIB_PCACHE_ExistsWord function 5-2474
PLIB_PCACHE_CacheLineFlashTypeBoot function 5-2443	PLIB_PCACHE_FlashDEDStatusClear function 5-2457
PLIB_PCACHE_CacheLineFlashTypeInst function 5-2443	PLIB_PCACHE_FlashDEDStatusGet function 5-2457
PLIB_PCACHE_CacheLineFlashTypeInst function 5-2444	PLIB_PCACHE_FlashSECCountGet function 5-2458
PLIB_PCACHE_CacheLineInst function 5-2444	PLIB_PCACHE_FlashSECCountSet function 5-2458
PLIB_PCACHE_CacheLineInvalid function 5-2445	PLIB_PCACHE_FlashSECIntDisable function 5-2459
PLIB_PCACHE_CacheLineInst function 5-2445	PLIB_PCACHE_FlashSECIntEnable function 5-2460
PLIB_PCACHE_CacheLinesLocked function 5-2446	PLIB_PCACHE_FlashSECStatusClear function 5-2460
PLIB_PCACHE_CacheLinesValid function 5-2447	PLIB_PCACHE_FlashSECStatusGet function 5-2461
PLIB_PCACHE_CacheLineLock function 5-2447	PLIB_PCACHE_FlashSECStatusSet function 5-2461
PLIB_PCACHE_CacheLineMaskGet function 5-2448	PLIB_PCACHE_InvalidateOnPFMPProgramAll function 5-2438
PLIB_PCACHE_CacheLineMaskSet function 5-2448	PLIB_PCACHE_InvalidateOnPFMPProgramUnlocked function 5-2439
PLIB_PCACHE_CacheLineSelect function 5-2449	PLIB_PCACHE_LeastRecentlyUsedStateRead function 5-2454
PLIB_PCACHE_CacheLineUnlock function 5-2449	PLIB_PCACHE_MAX_SEC_COUNT macro 5-2462
PLIB_PCACHE_CacheLineValid function 5-2450	PLIB_PCACHE_NUM_LINES macro 5-2462
PLIB_PCACHE_CacheMissRead function 5-2453	PLIB_PCACHE_NUM_WORDS_PER_LINE macro 5-2462
PLIB_PCACHE_CacheMissWrite function 5-2453	PLIB_PCACHE_PREFETCH_ENABLE enumeration 5-2463
PLIB_PCACHE_DATA_ENABLE enumeration 5-2463	PLIB_PCACHE_PrefetchAbortRead function 5-2456
PLIB_PCACHE_DataCacheEnableGet function 5-2437	PLIB_PCACHE_PrefetchAbortWrite function 5-2456
PLIB_PCACHE_DataCacheEnableSet function 5-2438	PLIB_PCACHE_PrefetchEnableGet function 5-2454
PLIB_PCACHE_ExistsCacheHit function 5-2464	PLIB_PCACHE_PrefetchEnableSet function 5-2455
PLIB_PCACHE_ExistsCacheLine function 5-2465	PLIB_PCACHE_WaitStateGet function 5-2439
PLIB_PCACHE_ExistsCacheLineAddr function 5-2465	PLIB_PCACHE_WaitStateSet function 5-2440
PLIB_PCACHE_ExistsCacheLineFlashType function 5-2466	PLIB_PCACHE_WordRead function 5-2450
PLIB_PCACHE_ExistsCacheLineLock function 5-2466	PLIB_PCACHE_WordWrite function 5-2451
PLIB_PCACHE_ExistsCacheLineMask function 5-2467	plib_pmp.h 5-2588
PLIB_PCACHE_ExistsCacheLineType function 5-2467	PLIB_PMP_AddressGet function 5-2531
PLIB_PCACHE_ExistsCacheLineValid function 5-2468	PLIB_PMP_AddressIncrementModeGet function 5-2494
PLIB_PCACHE_ExistsCacheMiss function 5-2468	PLIB_PMP_AddressIncrementModeSelect function 5-2495
PLIB_PCACHE_ExistsDataCacheEnable function 5-2469	PLIB_PMP_AddressLatchPolaritySelect function 5-2540
PLIB_PCACHE_ExistsFlashDEDStatus function 5-2470	PLIB_PMP_AddressLatchStrobeDisable function 5-2496
PLIB_PCACHE_ExistsFlashSECCount function 5-2470	PLIB_PMP_AddressLatchStrobeEnable function 5-2496
PLIB_PCACHE_ExistsFlashSECInt function 5-2471	PLIB_PMP_AddressLinesA0A1Get function 5-2532
PLIB_PCACHE_ExistsFlashSECStatus function 5-2471	PLIB_PMP_AddressLinesA0A1Set function 5-2532
PLIB_PCACHE_ExistsInvalidateOnPFMPProgram function 5-2472	PLIB_PMP_AddressPortDisable function 5-2535
PLIB_PCACHE_ExistsLeastRecentlyUsedState function 5-2472	PLIB_PMP_AddressPortEnable function 5-2535
PLIB_PCACHE_ExistsPrefetchAbort function 5-2473	PLIB_PMP_AddressSet function 5-2533
PLIB_PCACHE_ExistsPrefetchEnable function 5-2473	PLIB_PMP_AlternateMasterHasAccess function 5-2513

PLIB_PMP_AlternateMasterRequestStatus function 5-2514	PLIB_PMP_ExistsBufferRead function 5-2560
PLIB_PMP_BusKeeperDisable function 5-2506	PLIB_PMP_ExistsBufferType function 5-2561
PLIB_PMP_BusKeeperEnable function 5-2506	PLIB_PMP_ExistsBufferUnderFlow function 5-2561
PLIB_PMP_ByteEnablePolaritySelect function 5-2541	PLIB_PMP_ExistsBufferWrite function 5-2562
PLIB_PMP_ByteEnablePortDisable function 5-2536	PLIB_PMP_ExistsBusKeeper function 5-2562
PLIB_PMP_ByteEnablePortEnable function 5-2537	PLIB_PMP_ExistsBusyStatus function 5-2563
PLIB_PMP_ChipSelectFunctionSelect function 5-2497	PLIB_PMP_ExistsByteEnablePolarity function 5-2563
PLIB_PMP_ChipSelectXAckPolaritySelect function 5-2543	PLIB_PMP_ExistsByteEnablePortControl function 5-2564
PLIB_PMP_ChipSelectXByteEnablePortPolaritySelect function 5-2544	PLIB_PMP_ExistsChipSelectEnable function 5-2564
PLIB_PMP_ChipSelectXDisable function 5-2497	PLIB_PMP_ExistsChipSelectoperation function 5-2565
PLIB_PMP_ChipSelectXEnable function 5-2498	PLIB_PMP_ExistsChipXACKMode function 5-2565
PLIB_PMP_ChipSelectXEnableStrobeDeSelect function 5-2507	PLIB_PMP_ExistsChipXACKPolarity function 5-2566
PLIB_PMP_ChipSelectXEnableStrobeSelect function 5-2508	PLIB_PMP_ExistsChipXAltMasterWaitStates function 5-2566
PLIB_PMP_ChipSelectXIsActive function 5-2499	PLIB_PMP_ExistsChipXBaseAddress function 5-2567
PLIB_PMP_ChipSelectXPolaritySelect function 5-2542	PLIB_PMP_ExistsChipXByteEnablePolarity function 5-2567
PLIB_PMP_ChipSelectXPortDisable function 5-2539	PLIB_PMP_ExistsChipXDataHoldWaitStates function 5-2568
PLIB_PMP_ChipSelectXPortEnable function 5-2540	PLIB_PMP_ExistsChipXDataSetupWaitStates function 5-2568
PLIB_PMP_ChipSelectXReadWriteStrobePolaritySelect function 5-2545	PLIB_PMP_ExistsChipXDataSize function 5-2569
PLIB_PMP_ChipSelectXWriteEnableStrobePolaritySelect function 5-2545	PLIB_PMP_ExistsChipXEnableStrobeSelect function 5-2569
PLIB_PMP_ChipXAckModeSelect function 5-2509	PLIB_PMP_ExistsChipXPolarity function 5-2570
PLIB_PMP_ChipXBaseAddressSet function 5-2534	PLIB_PMP_ExistsChipXReadWritePolarity function 5-2570
PLIB_PMP_ChipXDataSizeSelect function 5-2510	PLIB_PMP_ExistsChipXStrobeWaitStates function 5-2571
PLIB_PMP_ChipXWaitStatesAlternateMasterSelect function 5-2527	PLIB_PMP_ExistsChipXWriteEnablePolarity function 5-2571
PLIB_PMP_ChipXWaitStatesDataHoldSelect function 5-2528	PLIB_PMP_ExistsCSXActiveStatus function 5-2572
PLIB_PMP_ChipXWaitStatesDataSetupSelect function 5-2529	PLIB_PMP_ExistsCSXPortControl function 5-2572
PLIB_PMP_ChipXWaitStatesStrobeSelect function 5-2529	PLIB_PMP_ExistsCurrentMaster function 5-2573
PLIB_PMP_DataSizeSelect function 5-2499	PLIB_PMP_ExistsDataHoldWaitStates function 5-2573
PLIB_PMP_Disable function 5-2500	PLIB_PMP_ExistsDataSetUpWaitStates function 5-2574
PLIB_PMP_Enable function 5-2500	PLIB_PMP_ExistsDataStrobeWaitStates function 5-2574
PLIB_PMP_EnhancedMasterModeSelect function 5-2510	PLIB_PMP_ExistsDataTransferSize function 5-2575
PLIB_PMP_ExistsAddressControl function 5-2557	PLIB_PMP_ExistsEnableControl function 5-2575
PLIB_PMP_ExistsAddressLatchPolarity function 5-2557	PLIB_PMP_ExistsEnhancedMasterMode function 5-2576
PLIB_PMP_ExistsAddressLatchStrobePortControl function 5-2558	PLIB_PMP_ExistsIncrementMode function 5-2576
PLIB_PMP_ExistsAddressPortPinControl function 5-2558	PLIB_PMP_ExistsInputBufferFull function 5-2577
PLIB_PMP_ExistsAltMasterRequest function 5-2559	PLIB_PMP_ExistsInputBufferXStatus function 5-2577
PLIB_PMP_ExistsBufferOverFlow function 5-2560	PLIB_PMP_ExistsInterruptMode function 5-2578
	PLIB_PMP_ExistsMasterRXTX function 5-2578
	PLIB_PMP_ExistsMUXModeSelect function 5-2579
	PLIB_PMP_ExistsOperationMode function 5-2579

PLIB_PMP_ExistsOutPutBufferEmpty function 5-2580	PLIB_PMP_ReadWriteStrobePortDisable function 5-2537
PLIB_PMP_ExistsOutputBufferXStatus function 5-2580	PLIB_PMP_ReadWriteStrobePortEnable function 5-2538
PLIB_PMP_ExistsReadWritePolarity function 5-2581	PLIB_PMP_ReservedAddressSpaceBitsSet function 5-2534
PLIB_PMP_ExistsReadWriteStrobePortControl function 5-2581	PLIB_PMP_SlaveReceive function 5-2524
PLIB_PMP_ExistsReservedAddrSpace function 5-2582	PLIB_PMP_SlaveSend function 5-2525
PLIB_PMP_ExistsSlaveRX function 5-2582	PLIB_PMP_SmartAddressStrobeDisable function 5-2508
PLIB_PMP_ExistsSlaveTX function 5-2583	PLIB_PMP_SmartAddressStrobeEnable function 5-2509
PLIB_PMP_ExistsSmartAddress function 5-2583	PLIB_PMP_StopInIdleDisable function 5-2505
PLIB_PMP_ExistsStopInIdleControl function 5-2584	PLIB_PMP_StopInIdleEnable function 5-2505
PLIB_PMP_ExistsTransactionError function 5-2584	PLIB_PMP_TransactionErrorClear function 5-2516
PLIB_PMP_ExistsTransactionTimeOut function 5-2585	PLIB_PMP_TransactionErrorHasOccurred function 5-2516
PLIB_PMP_ExistsWaitStatesAddrHoldStrobe function 5-2585	PLIB_PMP_TransactionHasTimedOut function 5-2517
PLIB_PMP_ExistsWaitStatesAddrLatchStrobe function 5-2586	PLIB_PMP_TransactionTimeoutClear function 5-2517
PLIB_PMP_ExistsWriteEnablePolarity function 5-2586	PLIB_PMP_WaitStatesAddressHoldStrobeSelect function 5-2530
PLIB_PMP_ExistsWriteEnablePortControl function 5-2587	PLIB_PMP_WaitStatesAddressLatchStrobeSelect function 5-2531
PLIB_PMP_InputBuffersAreFull function 5-2518	PLIB_PMP_WaitStatesDataHoldSelect function 5-2526
PLIB_PMP_InputBufferTypeSelect function 5-2501	PLIB_PMP_WaitStatesDataSetUpSelect function 5-2526
PLIB_PMP_InputBufferXByteReceive function 5-2519	PLIB_PMP_WaitStatesStrobeSelect function 5-2527
PLIB_PMP_InputBufferXIsFull function 5-2519	PLIB_PMP_WriteEnableStrobePolaritySelect function 5-2543
PLIB_PMP_InputOverflowClear function 5-2515	PLIB_PMP_WriteEnableStrobePortDisable function 5-2538
PLIB_PMP_InputOverflowHasOccurred function 5-2512	PLIB_PMP_WriteEnableStrobePortEnable function 5-2539
PLIB_PMP_InterruptModeGet function 5-2502	plib_ports.h 5-2659
PLIB_PMP_InterruptModeSelect function 5-2502	PLIB_PORTS_ChangeNoticeDisable function 5-2618
PLIB_PMP_IsDataReceived function 5-2520	PLIB_PORTS_ChangeNoticeEnable function 5-2618
PLIB_PMP_IsDataTransmitted function 5-2521	PLIB_PORTS_ChangeNoticeInIdleDisable function 5-2619
PLIB_PMP_IsEnabled function 5-2511	PLIB_PORTS_ChangeNoticeInIdleEnable function 5-2619
PLIB_PMP_MasterReceive function 5-2521	PLIB_PORTS_ChangeNoticeInIdlePerPortDisable function 5-2620
PLIB_PMP_MasterSend function 5-2522	PLIB_PORTS_ChangeNoticeInIdlePerPortEnable function 5-2620
PLIB_PMP_MultiplexModeGet function 5-2503	PLIB_PORTS_ChangeNoticePerPortHasOccured function 5-2621
PLIB_PMP_MultiplexModeSelect function 5-2503	PLIB_PORTS_ChangeNoticePerPortTurnOff function 5-2622
PLIB_PMP_OperationModeGet function 5-2504	PLIB_PORTS_ChangeNoticePerPortTurnOn function 5-2622
PLIB_PMP_OperationModeSelect function 5-2504	PLIB_PORTS_ChangeNoticePullDownPerPortDisable function 5-2623
PLIB_PMP_OutputBuffersAreEmpty function 5-2522	PLIB_PORTS_ChangeNoticePullDownPerPortEnable function 5-2624
PLIB_PMP_OutputBufferXByteSend function 5-2523	PLIB_PORTS_ChangeNoticePullUpDisable function 5-2624
PLIB_PMP_OutputBufferXIsEmpty function 5-2524	PLIB_PORTS_ChangeNoticePullUpEnable function 5-2625
PLIB_PMP_OutputUnderflowClear function 5-2515	
PLIB_PMP_OutputUnderflowHasOccurred function 5-2513	
PLIB_PMP_PortIsBusy function 5-2512	
PLIB_PMP_ReadWriteStrobePolaritySelect function 5-2542	

PLIB_PORTS_ChangeNoticePullUpPerPortDisable function 5-2625	PLIB_PORTS_PinModeSelect function 5-2606
PLIB_PORTS_ChangeNoticePullUpPerPortEnable function 5-2626	PLIB_PORTS_PinOpenDrainDisable function 5-2611
PLIB_PORTS_Clear function 5-2609	PLIB_PORTS_PinOpenDrainEnable function 5-2612
PLIB_PORTS_DirectionGet function 5-2610	PLIB_PORTS_PinSet function 5-2606
PLIB_PORTS_DirectionInputSet function 5-2610	PLIB_PORTS_PinToggle function 5-2607
PLIB_PORTS_DirectionOutputSet function 5-2611	PLIB_PORTS_PinWrite function 5-2608
PLIB_PORTS_ExistsChangeNotice function 5-2650	PLIB_PORTS_Read function 5-2613
PLIB_PORTS_ExistsChangeNoticeInIdle function 5-2650	PLIB_PORTS_RemapInput function 5-2616
PLIB_PORTS_ExistsChangeNoticePerPortInIdle function 5-2651	PLIB_PORTS_RemapOutput function 5-2617
PLIB_PORTS_ExistsChangeNoticePerPortStatus function 5-2651	PLIB_PORTS_Set function 5-2613
PLIB_PORTS_ExistsChangeNoticePerPortTurnOn function 5-2652	PLIB_PORTS_Toggle function 5-2614
PLIB_PORTS_ExistsChangeNoticePullDownPerPort function 5-2652	PLIB_PORTS_Write function 5-2614
PLIB_PORTS_ExistsChangeNoticePullUp function 5-2653	plib_power.h 5-2686
PLIB_PORTS_ExistsChangeNoticePullUpPerPort function 5-2653	PLIB_POWER_ClearIdleStatus function 5-2672
PLIB_PORTS_ExistsPinChangeNotice function 5-2654	PLIB_POWER_ClearSleepStatus function 5-2673
PLIB_PORTS_ExistsPinChangeNoticePerPort function 5-2654	PLIB_POWER_DeepSleepBORDisable function 5-2668
PLIB_PORTS_ExistsPinMode function 5-2655	PLIB_POWER_DeepSleepBOREnable function 5-2668
PLIB_PORTS_ExistsPinModePerPort function 5-2655	PLIB_POWER_DeepSleepFaultDetectStatus function 5-2673
PLIB_PORTS_ExistsPortsDirection function 5-2656	PLIB_POWER_DeepSleepInterruptOnChangeStatus function 5-2674
PLIB_PORTS_ExistsPortsOpenDrain function 5-2656	PLIB_POWER_DeepSleepMCLREventStatus function 5-2675
PLIB_PORTS_ExistsPortsRead function 5-2657	PLIB_POWER_DeepSleepModeDisable function 5-2669
PLIB_PORTS_ExistsPortsWrite function 5-2658	PLIB_POWER_DeepSleepModeEnable function 5-2669
PLIB_PORTS_ExistsRemapInput function 5-2658	PLIB_POWER_DeepSleepPowerOnResetStatus function 5-2675
PLIB_PORTS_ExistsRemapOutput function 5-2659	PLIB_POWER_DeviceWasInIdleMode function 5-2676
PLIB_PORTS_OpenDrainDisable function 5-2615	PLIB_POWER_DeviceWasInSleepMode function 5-2676
PLIB_PORTS_OpenDrainEnable function 5-2616	PLIB_POWER_ExistsDeepSleepBOR function 5-2681
PLIB_PORTS_PinChangeNoticeDisable function 5-2627	PLIB_POWER_ExistsDeepSleepFaultDetect function 5-2681
PLIB_PORTS_PinChangeNoticeEnable function 5-2627	PLIB_POWER_ExistsDeepSleepInterruptOnChange function 5-2682
PLIB_PORTS_PinChangeNoticePerPortDisable function 5-2628	PLIB_POWER_ExistsDeepSleepMCLREvent function 5-2682
PLIB_PORTS_PinChangeNoticePerPortEnable function 5-2628	PLIB_POWER_ExistsDeepSleepMode function 5-2683
PLIB_PORTS_PinClear function 5-2603	PLIB_POWER_ExistsDeepSleepPowerOnReset function 5-2683
PLIB_PORTS_PinDirectionInputSet function 5-2604	PLIB_POWER_ExistsIdleStatus function 5-2684
PLIB_PORTS_PinDirectionOutputSet function 5-2605	PLIB_POWER_ExistsPeripheralModuleControl function 5-2684
PLIB_PORTS_PinGet function 5-2605	PLIB_POWER_ExistsSleepStatus function 5-2685
PLIB_PORTS_PinModePerPortSelect function 5-2608	PLIB_POWER_ExistsVoltageRegulatorControl function 5-2685
	PLIB_POWER_PeripheralModuleDisable function 5-2670
	PLIB_POWER_PeripheralModuleEnable function 5-2670

PLIB_POWER_PeripheralModuleIsEnabled function 5-2671	PLIB_RTCC_AlarmRepeatCountGet function 5-2738
PLIB_POWER_VoltageRegulatorDisable function 5-2671	PLIB_RTCC_AlarmRepeatCountSet function 5-2739
PLIB_POWER_VoltageRegulatorEnable function 5-2672	PLIB_RTCC_AlarmSecondGet function 5-2739
PLIB_POWER_VoltageRegulatorIsEnabled function 5-2677	PLIB_RTCC_AlarmSecondSet function 5-2740
plib_reset.h 5-2702	PLIB_RTCC_AlarmTimeGet function 5-2740
PLIB_RESET_ConfigRegReadErrorGet function 5-2693	PLIB_RTCC_AlarmTimeSet function 5-2741
PLIB_RESET_ExistsConfigRegReadError function 5-2697	PLIB_RTCC_AlarmValueRegisterPointer function 5-2741
PLIB_RESET_ExistsNmiControl function 5-2698	PLIB_RTCC_AlarmWeekDayGet function 5-2742
PLIB_RESET_ExistsNmiCounter function 5-2698	PLIB_RTCC_AlarmWeekDaySet function 5-2742
PLIB_RESET_ExistsResetReasonStatus function 5-2696	PLIB_RTCC_ClockOutputDisable function 5-2715
PLIB_RESET_ExistsSoftwareResetTrigger function 5-2697	PLIB_RTCC_ClockOutputEnable function 5-2716
PLIB_RESET_ExistsWdtInSleep function 5-2699	PLIB_RTCC_ClockRunningStatus function 5-2716
PLIB_RESET_NmiCounterValueGet function 5-2699	PLIB_RTCC_ClockSourceSelect function 5-2743
PLIB_RESET_NmiCounterValueSet function 5-2700	PLIB_RTCC_Disable function 5-2717
PLIB_RESET_NmiEventClear function 5-2701	PLIB_RTCC_DriftCalibrateGet function 5-2744
PLIB_RESET_NmiEventTrigger function 5-2701	PLIB_RTCC_DriftCalibrateSet function 5-2744
PLIB_RESET_NmiReasonGet function 5-2702	PLIB_RTCC_Enable function 5-2717
PLIB_RESET_ReasonClear function 5-2692	PLIB_RTCC_ExistsAlarmChimeControl function 5-2749
PLIB_RESET_ReasonGet function 5-2692	PLIB_RTCC_ExistsAlarmControl function 5-2749
PLIB_RESET_SoftwareResetEnable function 5-2691	PLIB_RTCC_ExistsAlarmDate function 5-2750
PLIB_RESET_WdtTimeOutHasOccurredInSleep function 5-2693	PLIB_RTCC_ExistsAlarmMaskControl function 5-2750
plib_rtcc.h 5-2759	PLIB_RTCC_ExistsAlarmPulseInitial function 5-2751
PLIB_RTCC_AlarmChimeDisable function 5-2729	PLIB_RTCC_ExistsAlarmRepeatControl function 5-2751
PLIB_RTCC_AlarmChimeEnable function 5-2729	PLIB_RTCC_ExistsAlarmSynchronization function 5-2752
PLIB_RTCC_AlarmDateGet function 5-2730	PLIB_RTCC_ExistsAlarmTime function 5-2752
PLIB_RTCC_AlarmDateSet function 5-2730	PLIB_RTCC_ExistsCalibration function 5-2753
PLIB_RTCC_AlarmDayGet function 5-2731	PLIB_RTCC_ExistsClockRunning function 5-2753
PLIB_RTCC_AlarmDaySet function 5-2731	PLIB_RTCC_ExistsClockSelect function 5-2754
PLIB_RTCC_AlarmDisable function 5-2732	PLIB_RTCC_ExistsEnableControl function 5-2754
PLIB_RTCC_AlarmEnable function 5-2732	PLIB_RTCC_ExistsHalfSecond function 5-2755
PLIB_RTCC_AlarmHourGet function 5-2733	PLIB_RTCC_ExistsOutputControl function 5-2755
PLIB_RTCC_AlarmHourSet function 5-2734	PLIB_RTCC_ExistsOutputSelect function 5-2756
PLIB_RTCC_AlarmMaskModeSelect function 5-2734	PLIB_RTCC_ExistsRTCTDate function 5-2756
PLIB_RTCC_AlarmMinuteGet function 5-2735	PLIB_RTCC_ExistsRTCTime function 5-2757
PLIB_RTCC_AlarmMinuteSet function 5-2735	PLIB_RTCC_ExistsStopInIdleControl function 5-2757
PLIB_RTCC_AlarmMonthGet function 5-2736	PLIB_RTCC_ExistsSynchronization function 5-2758
PLIB_RTCC_AlarmMonthSet function 5-2736	PLIB_RTCC_ExistsWriteEnable function 5-2758
PLIB_RTCC_AlarmPulseInitialGet function 5-2737	PLIB_RTCC_HalfSecondStatusGet function 5-2745
PLIB_RTCC_AlarmPulseInitialSet function 5-2737	PLIB_RTCC_OutputSelect function 5-2745

PLIB_RTCC_RTCCDateGet function 5-2719	PLIB_SB_ExistsPGVErrPG function 5-2794
PLIB_RTCC_RTCCDateSet function 5-2719	PLIB_SB_ExistsPGVErrRegion function 5-2795
PLIB_RTCC_RTCCDayGet function 5-2720	PLIB_SB_ExistsPGVErrRptPri function 5-2795
PLIB_RTCC_RTCCDaySet function 5-2720	PLIB_SB_ExistsPGVErrStatus function 5-2796
PLIB_RTCC_RTCHourGet function 5-2721	PLIB_SB_INIT_ID enumeration 5-2781
PLIB_RTCC_RTCHourSet function 5-2722	PLIB_SB_INIT_PG enumeration 5-2782
PLIB_RTCC_RTCMinuteGet function 5-2722	PLIB_SB_InitPermGrpSet function 5-2787
PLIB_RTCC_RTCMinuteSet function 5-2723	PLIB_SB_OCP_CMD_CODE enumeration 5-2782
PLIB_RTCC_RTCMonthGet function 5-2723	PLIB_SB_PG_INITIATOR enumeration 5-2782
PLIB_RTCC_RTCMonthSet function 5-2724	PLIB_SB_PGRegionAddrGet function 5-2768
PLIB_RTCC_RTCSecondGet function 5-2724	PLIB_SB_PGRegionAddrSet function 5-2769
PLIB_RTCC_RTCSecondSet function 5-2725	PLIB_SB_PGRegionReadPermClear function 5-2770
PLIB_RTCC_RTCTimeGet function 5-2725	PLIB_SB_PGRegionReadPermSet function 5-2770
PLIB_RTCC_RTCTimeSet function 5-2726	PLIB_SB_PGRegionSizeGet function 5-2771
PLIB_RTCC_RTCWeekDayGet function 5-2726	PLIB_SB_PGRegionSizeSet function 5-2771
PLIB_RTCC_RTCWeekDaySet function 5-2727	PLIB_SB_PGRegionWritePermClear function 5-2772
PLIB_RTCC_RTCYearGet function 5-2727	PLIB_SB_PGRegionWritePermSet function 5-2773
PLIB_RTCC_RTCYearSet function 5-2728	PLIB_SB_PGVErrorClearMulti function 5-2773
PLIB_RTCC_StopInIdleDisable function 5-2746	PLIB_SB_PGVErrorClearSingle function 5-2774
PLIB_RTCC_StopInIdleEnable function 5-2746	PLIB_SB_PGVErrorCode function 5-2775
PLIB_RTCC_WriteDisable function 5-2718	PLIB_SB_PGVErrorCommandCode function 5-2775
PLIB_RTCC_WriteEnable function 5-2718	PLIB_SB_PGVErrorInitiatorID function 5-2776
plib_sb.h 5-2796	PLIB_SB_PGVErrorLogClearMulti function 5-2776
PLIB_SB_ARB_POLICY enumeration 5-2781	PLIB_SB_PGVErrorLogClearSingle function 5-2777
PLIB_SB_CPUPrioritySet function 5-2786	PLIB_SB_PGVErrorMulti function 5-2777
PLIB_SB_DMAPrioritySet function 5-2786	PLIB_SB_PGVErrorPermissionGroup function 5-2778
PLIB_SB_ERROR enumeration 5-2781	PLIB_SB_PGVErrorRegion function 5-2778
PLIB_SB_ExistsCPUPriority function 5-2788	PLIB_SB_PGVErrorReportPrimaryDisable function 5-2779
PLIB_SB_ExistsDMAPriority function 5-2788	PLIB_SB_PGVErrorReportPrimaryEnable function 5-2779
PLIB_SB_ExistsInitPermGrp function 5-2789	PLIB_SB_PGVErrorStatus function 5-2780
PLIB_SB_ExistsPGRegAddr function 5-2789	PLIB_SB_REGION_PG enumeration 5-2783
PLIB_SB_ExistsPGRegRdPerm function 5-2790	PLIB_SB_TGT_ID enumeration 5-2783
PLIB_SB_ExistsPGRegSize function 5-2790	PLIB_SB_TGT_REGION enumeration 5-2784
PLIB_SB_ExistsPGRegWrPerm function 5-2791	plib_spi.h 5-2888
PLIB_SB_ExistsPGVErrClear function 5-2791	PLIB_SPI_AudioCommunicationWidthSelect function 5-2850
PLIB_SB_ExistsPGVErrClrMulti function 5-2792	PLIB_SPI_AudioErrorDisable function 5-2851
PLIB_SB_ExistsPGVErrClrSingle function 5-2793	PLIB_SPI_AudioErrorEnable function 5-2851
PLIB_SB_ExistsPGVErrCmdCode function 5-2793	PLIB_SPI_AudioProtocolDisable function 5-2852
PLIB_SB_ExistsPGVErrInitID function 5-2794	PLIB_SPI_AudioProtocolEnable function 5-2852

PLIB_SPI_AudioProtocolModeSelect function 5-2853	PLIB_SPI_ExistsOutputDataPhase function 5-2881
PLIB_SPI_AudioTransmitModeSelect function 5-2854	PLIB_SPI_ExistsPinControl function 5-2881
PLIB_SPI_BaudRateClockSelect function 5-2827	PLIB_SPI_ExistsPrimaryPrescale function 5-2882
PLIB_SPI_BaudRateSet function 5-2828	PLIB_SPI_ExistsReadDataSignStatus function 5-2882
PLIB_SPI_BufferClear function 5-2843	PLIB_SPI_ExistsReceiveBufferStatus function 5-2883
PLIB_SPI_BufferRead function 5-2844	PLIB_SPI_ExistsReceiveFIFOStatus function 5-2883
PLIB_SPI_BufferWrite function 5-2844	PLIB_SPI_ExistsReceiverOverflow function 5-2884
PLIB_SPI_ClockPolaritySelect function 5-2829	PLIB_SPI_ExistsSecondaryPrescale function 5-2884
PLIB_SPI_CommunicationWidthSelect function 5-2829	PLIB_SPI_ExistsSlaveSelectControl function 5-2885
PLIB_SPI_Disable function 5-2830	PLIB_SPI_ExistsStopInIdleControl function 5-2885
PLIB_SPI_Enable function 5-2830	PLIB_SPI_ExistsTransmitBufferEmptyStatus function 5-2886
PLIB_SPI_ErrorInterruptDisable function 5-2831	PLIB_SPI_ExistsTransmitBufferFullStatus function 5-2886
PLIB_SPI_ErrorInterruptEnable function 5-2832	PLIB_SPI_ExistsTransmitUnderRunStatus function 5-2887
PLIB_SPI_ExistsAudioCommunicationWidth function 5-2868	PLIB_SPI_FIFOCountGet function 5-2832
PLIB_SPI_ExistsAudioErrorControl function 5-2868	PLIB_SPI_FIFODisable function 5-2833
PLIB_SPI_ExistsAudioProtocolControl function 5-2869	PLIB_SPI_FIFOEnable function 5-2833
PLIB_SPI_ExistsAudioProtocolMode function 5-2869	PLIB_SPI_FIFOInterruptModeSelect function 5-2834
PLIB_SPI_ExistsAudioTransmitMode function 5-2870	PLIB_SPI_FIFOShiftRegisterIsEmpty function 5-2835
PLIB_SPI_ExistsBaudRate function 5-2870	PLIB_SPI_FramedCommunicationDisable function 5-2845
PLIB_SPI_ExistsBaudRateClock function 5-2871	PLIB_SPI_FramedCommunicationEnable function 5-2846
PLIB_SPI_ExistsBuffer function 5-2871	PLIB_SPI_FrameErrorStatusGet function 5-2846
PLIB_SPI_ExistsBusStatus function 5-2872	PLIB_SPI_FrameSyncPulseCounterSelect function 5-2847
PLIB_SPI_ExistsClockPolarity function 5-2872	PLIB_SPI_FrameSyncPulseDirectionSelect function 5-2847
PLIB_SPI_ExistsCommunicationWidth function 5-2873	PLIB_SPI_FrameSyncPulseEdgeSelect function 5-2848
PLIB_SPI_ExistsEnableControl function 5-2873	PLIB_SPI_FrameSyncPulsePolaritySelect function 5-2849
PLIB_SPI_ExistsErrorInterruptControl function 5-2874	PLIB_SPI_FrameSyncPulseWidthSelect function 5-2849
PLIB_SPI_ExistsFIFOControl function 5-2874	PLIB_SPI_InputSamplePhaseSelect function 5-2835
PLIB_SPI_ExistsFIFOCount function 5-2875	PLIB_SPI_IsBusy function 5-2836
PLIB_SPI_ExistsFIFOInterruptMode function 5-2875	PLIB_SPI_MasterEnable function 5-2836
PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus function 5-2876	PLIB_SPI_OutputDataPhaseSelect function 5-2837
PLIB_SPI_ExistsFramedCommunication function 5-2876	PLIB_SPI_PinDisable function 5-2838
PLIB_SPI_ExistsFrameErrorStatus function 5-2877	PLIB_SPI_PinEnable function 5-2838
PLIB_SPI_ExistsFrameSyncPulseCounter function 5-2877	PLIB_SPI_PrescalePrimarySelect function 5-2839
PLIB_SPI_ExistsFrameSyncPulseDirection function 5-2878	PLIB_SPI_PrescaleSecondarySelect function 5-2839
PLIB_SPI_ExistsFrameSyncPulseEdge function 5-2878	PLIB_SPI_ReadDataIsSignExtended function 5-2840
PLIB_SPI_ExistsFrameSyncPulsePolarity function 5-2879	PLIB_SPI_ReceiverBufferIsFull function 5-2856
PLIB_SPI_ExistsFrameSyncPulseWidth function 5-2879	PLIB_SPI_ReceiverFIFOIsEmpty function 5-2857
PLIB_SPI_ExistsInputSamplePhase function 5-2880	PLIB_SPI_ReceiverHasOverflowed function 5-2857
PLIB_SPI_ExistsMasterControl function 5-2880	PLIB_SPI_ReceiverOverflowClear function 5-2858

PLIB_SPI_SlaveEnable function 5-2841	PLIB_SQI_DMABDFetchStart function 5-2960
PLIB_SPI_SlaveSelectDisable function 5-2841	PLIB_SQI_DMABDIsBusy function 5-2960
PLIB_SPI_SlaveSelectEnable function 5-2842	PLIB_SQI_DMABDPollCounterSet function 5-2961
PLIB_SPI_StopInIdleDisable function 5-2842	PLIB_SQI_DMABDPollDisable function 5-2962
PLIB_SPI_StopInIdleEnable function 5-2843	PLIB_SQI_DMABDPollEnable function 5-2962
PLIB_SPI_TransmitBufferIsEmpty function 5-2854	PLIB_SQI_DMABDPollsEnabled function 5-2963
PLIB_SPI_TransmitBufferIsFull function 5-2855	PLIB_SQI_DMABDReceiveBufferCountGet function 5-2957
PLIB_SPI_TransmitUnderRunStatusGet function 5-2855	PLIB_SQI_DMABDReceiveBufferLengthGet function 5-2957
plib_sqi.h 5-3006	PLIB_SQI_DMABDReceiveStateGet function 5-2958
PLIB_SQI_BurstEnable function 5-2908	PLIB_SQI_DMABDStateGet function 5-2963
PLIB_SQI_ByteCountGet function 5-2937	PLIB_SQI_DMABDTransmitBufferCountGet function 5-2958
PLIB_SQI_ByteCountSet function 5-2938	PLIB_SQI_DMABDTransmitBufferLengthGet function 5-2959
PLIB_SQI_ChipSelectDeassertDisable function 5-2938	PLIB_SQI_DMABDTransmitStateGet function 5-2959
PLIB_SQI_ChipSelectDeassertEnable function 5-2939	PLIB_SQI_DMADisable function 5-2964
PLIB_SQI_ChipSelectGet function 5-2940	PLIB_SQI_DMAEnable function 5-2964
PLIB_SQI_ChipSelectSet function 5-2940	PLIB_SQI_DMAHasStarted function 5-2965
PLIB_SQI_ClockDisable function 5-2909	PLIB_SQI_DMAsEnabled function 5-2965
PLIB_SQI_ClockDividerGet function 5-2925	PLIB_SQI_Enable function 5-2915
PLIB_SQI_ClockDividerSet function 5-2925	PLIB_SQI_ExistsBDBaseAddress function 5-2974
PLIB_SQI_ClockEnable function 5-2926	PLIB_SQI_ExistsBDControlWord function 5-2975
PLIB_SQI_ClockIsStable function 5-2926	PLIB_SQI_ExistsBDCurrentAddress function 5-2975
PLIB_SQI_ConfigWordGet function 5-2941	PLIB_SQI_ExistsBDPollCount function 5-2976
PLIB_SQI_ConfigWordSet function 5-2941	PLIB_SQI_ExistsBDPollingEnable function 5-2976
PLIB_SQI_ControlBufferThresholdGet function 5-2909	PLIB_SQI_ExistsBDProcessState function 5-2977
PLIB_SQI_ControlBufferThresholdSet function 5-2910	PLIB_SQI_ExistsBDRxBufCount function 5-2977
PLIB_SQI_ControlWordGet function 5-2910	PLIB_SQI_ExistsBDRxLength function 5-2978
PLIB_SQI_ControlWordSet function 5-2911	PLIB_SQI_ExistsBDRxState function 5-2978
PLIB_SQI_CSOutputEnableSelect function 5-2912	PLIB_SQI_ExistsBDTxBufCount function 5-2979
PLIB_SQI_DataFormatGet function 5-2912	PLIB_SQI_ExistsBDTxLength function 5-2979
PLIB_SQI_DataFormatSet function 5-2913	PLIB_SQI_ExistsBDTxState function 5-2980
PLIB_SQI_DataLineStatus function 5-2949	PLIB_SQI_ExistsBurstControl function 5-2980
PLIB_SQI_DataModeGet function 5-2913	PLIB_SQI_ExistsChipSelect function 5-2981
PLIB_SQI_DataModeSet function 5-2914	PLIB_SQI_ExistsClockControl function 5-2981
PLIB_SQI_DataOutputEnableSelect function 5-2914	PLIB_SQI_ExistsClockDivider function 5-2982
PLIB_SQI_Disable function 5-2915	PLIB_SQI_ExistsClockReady function 5-2982
PLIB_SQI_DMABDBaseAddressGet function 5-2954	PLIB_SQI_ExistsConBufThreshold function 5-2983
PLIB_SQI_DMABDBaseAddressSet function 5-2955	PLIB_SQI_ExistsConfigWord function 5-2983
PLIB_SQI_DMABDControlWordGet function 5-2956	PLIB_SQI_ExistsControlWord function 5-2984
PLIB_SQI_DMABDCurrentAddressGet function 5-2955	PLIB_SQI_ExistsCSDeassert function 5-2984

PLIB_SQI_ExistsCSOutputEnable function 5-2985	PLIB_SQI_ExistsXIPReadOpCode function 5-3005
PLIB_SQI_ExistsDataFormat function 5-2985	PLIB_SQI_HoldClear function 5-2916
PLIB_SQI_ExistsDataModeControl function 5-2986	PLIB_SQI_HoldGet function 5-2916
PLIB_SQI_ExistsDataOutputEnable function 5-2986	PLIB_SQI_HoldSet function 5-2917
PLIB_SQI_ExistsDataPinStatus function 5-2987	PLIB_SQI_InterruptDisable function 5-2950
PLIB_SQI_ExistsDmaEnable function 5-2987	PLIB_SQI_InterruptEnable function 5-2950
PLIB_SQI_ExistsDMAEngineBusy function 5-2988	PLIB_SQI_InterruptFlagGet function 5-2951
PLIB_SQI_ExistsDMAProcessInProgress function 5-2988	PLIB_SQI_InterruptIsEnabled function 5-2952
PLIB_SQI_ExistsEnableControl function 5-2989	PLIB_SQI_InterruptSignalDisable function 5-2952
PLIB_SQI_ExistsHoldPinControl function 5-2989	PLIB_SQI_InterruptSignalEnable function 5-2953
PLIB_SQI_ExistsInterruptControl function 5-2990	PLIB_SQI_InterruptSignalIsEnabled function 5-2953
PLIB_SQI_ExistsInterruptSignalControl function 5-2990	PLIB_SQI_LaneModeGet function 5-2917
PLIB_SQI_ExistsInterruptStatus function 5-2991	PLIB_SQI_LaneModeSet function 5-2918
PLIB_SQI_ExistsLaneMode function 5-2992	PLIB_SQI_NumberOfReceiveBufferReads function 5-2919
PLIB_SQI_ExistsReceiveLatch function 5-2992	PLIB_SQI_ReceiveBufferIsUnderrun function 5-2942
PLIB_SQI_ExistsRxBufferCount function 5-2993	PLIB_SQI_ReceiveData function 5-2919
PLIB_SQI_ExistsRxBufIntThreshold function 5-2993	PLIB_SQI_ReceiveLatchDisable function 5-2920
PLIB_SQI_ExistsRxBufThreshold function 5-2994	PLIB_SQI_ReceiveLatchEnable function 5-2920
PLIB_SQI_ExistsRxData function 5-2994	PLIB_SQI_ReceiveLatchGet function 5-2921
PLIB_SQI_ExistsRxUnderRun function 5-2995	PLIB_SQI_RxBufferThresholdGet function 5-2943
PLIB_SQI_ExistsSoftReset function 5-2995	PLIB_SQI_RxBufferThresholdIntGet function 5-2943
PLIB_SQI_ExistsStartDMA function 5-2996	PLIB_SQI_RxBufferThresholdIntSet function 5-2944
PLIB_SQI_ExistsTransferCommand function 5-2996	PLIB_SQI_RxBufferThresholdSet function 5-2944
PLIB_SQI_ExistsTransferCount function 5-2997	PLIB_SQI_SoftReset function 5-2921
PLIB_SQI_ExistsTransferModeControl function 5-2997	PLIB_SQI_TransferDirectionGet function 5-2945
PLIB_SQI_ExistsTxBufferFree function 5-2998	PLIB_SQI_TransferDirectionSet function 5-2945
PLIB_SQI_ExistsTxBufIntThreshold function 5-2998	PLIB_SQI_TransferModeGet function 5-2922
PLIB_SQI_ExistsTxBufThreshold function 5-2999	PLIB_SQI_TransferModeSet function 5-2922
PLIB_SQI_ExistsTxData function 5-2999	PLIB_SQI_TransmitBufferFreeSpaceGet function 5-2946
PLIB_SQI_ExistsTxOverflow function 5-3000	PLIB_SQI_TransmitBufferHasOverflowed function 5-2947
PLIB_SQI_ExistsWPPinControl function 5-3000	PLIB_SQI_TransmitData function 5-2923
PLIB_SQI_ExistsXIPChipSelect function 5-3001	PLIB_SQI_TxBufferThresholdGet function 5-2947
PLIB_SQI_ExistsXIPControlWord1 function 5-3001	PLIB_SQI_TxBufferThresholdIntGet function 5-2948
PLIB_SQI_ExistsXIPControlWord2 function 5-3002	PLIB_SQI_TxBufferThresholdIntSet function 5-2948
PLIB_SQI_ExistsXIPLaneMode function 5-3003	PLIB_SQI_TxBufferThresholdSet function 5-2949
PLIB_SQI_ExistsXIPModeBytes function 5-3003	PLIB_SQI_WriteProtectClear function 5-2923
PLIB_SQI_ExistsXIPModeCode function 5-3004	PLIB_SQI_WriteProtectGet function 5-2924
PLIB_SQI_ExistsXIPNumberOfAddressBytes function 5-3004	PLIB_SQI_WriteProtectSet function 5-2924
PLIB_SQI_ExistsXIPNumberOfDummyBytes function 5-3005	PLIB_SQI_XIPAddressBytesGet function 5-2927

PLIB_SQI_XIPAddressBytesSet function 5-2927	PLIB_TMR_ExistsClockSourceSync function 5-3065
PLIB_SQI_XIPChipSelectGet function 5-2928	PLIB_TMR_ExistsCounter16Bit function 5-3066
PLIB_SQI_XIPChipSelectSet function 5-2928	PLIB_TMR_ExistsCounter32Bit function 5-3066
PLIB_SQI_XIPControlWord1Get function 5-2929	PLIB_TMR_ExistsCounter8Bit function 5-3067
PLIB_SQI_XIPControlWord1Set function 5-2930	PLIB_TMR_ExistsCounterAsyncWriteControl function 5-3067
PLIB_SQI_XIPControlWord2Get function 5-2931	PLIB_TMR_ExistsCounterAsyncWriteInProgress function 5-3068
PLIB_SQI_XIPControlWord2Set function 5-2931	PLIB_TMR_ExistsCountMode function 5-3068
PLIB_SQI_XIPDummyBytesGet function 5-2932	PLIB_TMR_ExistsEnableControl function 5-3069
PLIB_SQI_XIPDummyBytesSet function 5-2932	PLIB_TMR_ExistsGateCurrentState function 5-3069
PLIB_SQI_XIPLaneModeGet function 5-2933	PLIB_TMR_ExistsGatedTimeAccumulation function 5-3070
PLIB_SQI_XIPLaneModeSet function 5-2933	PLIB_TMR_ExistsGatePolarity function 5-3070
PLIB_SQI_XIPModeBytesGet function 5-2934	PLIB_TMR_ExistsGateSinglePulseAcquisition function 5-3071
PLIB_SQI_XIPModeBytesSet function 5-2935	PLIB_TMR_ExistsGateSinglePulseMode function 5-3071
PLIB_SQI_XIPModeCodeGet function 5-2935	PLIB_TMR_ExistsGateSource function 5-3072
PLIB_SQI_XIPModeCodeSet function 5-2936	PLIB_TMR_ExistsGateToggleMode function 5-3072
PLIB_SQI_XIPReadOpcodeGet function 5-2936	PLIB_TMR_ExistsMode16Bit function 5-3073
PLIB_SQI_XIPReadOpcodeSet function 5-2937	PLIB_TMR_ExistsMode32Bit function 5-3073
plib_tmr.h 5-3081	PLIB_TMR_ExistsMode8Bit function 5-3074
PLIB_TMR_AssignmentSelect function 5-3025	PLIB_TMR_ExistsOperationInSleep function 5-3074
PLIB_TMR_ClockSourceEdgeGet function 5-3034	PLIB_TMR_ExistsPeriod16Bit function 5-3075
PLIB_TMR_ClockSourceEdgeSelect function 5-3035	PLIB_TMR_ExistsPeriod32Bit function 5-3075
PLIB_TMR_ClockSourceExternalSyncDisable function 5-3035	PLIB_TMR_ExistsPeriod8Bit function 5-3076
PLIB_TMR_ClockSourceExternalSyncEnable function 5-3036	PLIB_TMR_ExistsPostscale function 5-3076
PLIB_TMR_ClockSourceSelect function 5-3036	PLIB_TMR_ExistsPrescale function 5-3077
PLIB_TMR_ContinuousCountModeEnable function 5-3026	PLIB_TMR_ExistsPrescalerAssignment function 5-3078
PLIB_TMR_Counter16BitClear function 5-3050	PLIB_TMR_ExistsStopInIdleControl function 5-3078
PLIB_TMR_Counter16BitGet function 5-3050	PLIB_TMR_ExistsSystemClockStatus function 5-3079
PLIB_TMR_Counter16BitSet function 5-3051	PLIB_TMR_ExistsTimerAssignment function 5-3079
PLIB_TMR_Counter32BitClear function 5-3051	PLIB_TMR_ExistsTimerOperationMode function 5-3080
PLIB_TMR_Counter32BitGet function 5-3052	PLIB_TMR_ExistsTimerOscillator function 5-3080
PLIB_TMR_Counter32BitSet function 5-3053	PLIB_TMR_ExistsTriggerEventReset function 5-3081
PLIB_TMR_Counter8BitClear function 5-3053	PLIB_TMR_GateCurrentStateGet function 5-3037
PLIB_TMR_Counter8BitGet function 5-3054	PLIB_TMR_GateDisable function 5-3038
PLIB_TMR_Counter8BitSet function 5-3054	PLIB_TMR_GateEnable function 5-3038
PLIB_TMR_CounterAsyncWriteDisable function 5-3055	PLIB_TMR_GatePolaritySelect function 5-3039
PLIB_TMR_CounterAsyncWriteEnable function 5-3055	PLIB_TMR_GateSinglePulseAcquisitionHasCompleted function 5-3039
PLIB_TMR_CounterAsyncWriteInProgress function 5-3056	PLIB_TMR_GateSinglePulseAcquisitionStart function 5-3040
PLIB_TMR_ExistsClockSource function 5-3064	PLIB_TMR_GateSinglePulseModeDisable function 5-3040
PLIB_TMR_ExistsClockSourceEdge function 5-3065	

PLIB_TMR_GateSinglePulseModeEnable function 5-3041	PLIB_USART_BaudRateAutoDetectOverflowHasOccurred function 5-3117
PLIB_TMR_GateSourceSelect function 5-3041	PLIB_USART_BaudRateGet function 5-3118
PLIB_TMR_GateToggleModeDisable function 5-3042	PLIB_USART_BaudRateHighDisable function 5-3118
PLIB_TMR_GateToggleModeEnable function 5-3042	PLIB_USART_BaudRateHighEnable function 5-3119
PLIB_TMR_IsPeriodMatchBased function 5-3058	PLIB_USART_BaudRateHighSet function 5-3119
PLIB_TMR_Mode16BitEnable function 5-3026	PLIB_USART_BaudRateModeSelect function 5-3120
PLIB_TMR_Mode32BitEnable function 5-3027	PLIB_USART_BaudRateSet function 5-3121
PLIB_TMR_Mode8BitEnable function 5-3028	PLIB_USART_Disable function 5-3106
PLIB_TMR_OperateInSleepDisable function 5-3032	PLIB_USART_Enable function 5-3106
PLIB_TMR_OperateInSleepEnable function 5-3032	PLIB_USART_ExistsAlternateIO function 5-3144
PLIB_TMR_Period16BitGet function 5-3046	PLIB_USART_ExistsBaudRate function 5-3145
PLIB_TMR_Period16BitSet function 5-3046	PLIB_USART_ExistsBaudRateAutoDetect function 5-3145
PLIB_TMR_Period32BitGet function 5-3047	PLIB_USART_ExistsBaudRateAutoDetectOverflow function 5-3146
PLIB_TMR_Period32BitSet function 5-3048	PLIB_USART_ExistsBaudRateBitMode function 5-3146
PLIB_TMR_Period8BitGet function 5-3048	PLIB_USART_ExistsBaudRateHigh function 5-3147
PLIB_TMR_Period8BitSet function 5-3049	PLIB_USART_ExistsEnable function 5-3147
PLIB_TMR_PostscaleDivisorGet function 5-3057	PLIB_USART_ExistsHandshakeMode function 5-3148
PLIB_TMR_PostscaleGet function 5-3057	PLIB_USART_ExistsIrDA function 5-3148
PLIB_TMR_PostscaleSelect function 5-3058	PLIB_USART_ExistsLineControlMode function 5-3149
PLIB_TMR_PrescaleDivisorGet function 5-3043	PLIB_USART_ExistsLoopback function 5-3149
PLIB_TMR_PrescaleGet function 5-3044	PLIB_USART_ExistsOperationMode function 5-3150
PLIB_TMR_PrescalerDisable function 5-3044	PLIB_USART_ExistsReceiver function 5-3150
PLIB_TMR_PrescalerEnable function 5-3045	PLIB_USART_ExistsReceiverAddressAutoDetect function 5-3151
PLIB_TMR_PrescaleSelect function 5-3045	PLIB_USART_ExistsReceiverAddressDetect function 5-3151
PLIB_TMR_SingleShotModeEnable function 5-3031	PLIB_USART_ExistsReceiverDataAvailableStatus function 5-3152
PLIB_TMR_Start function 5-3028	PLIB_USART_ExistsReceiverEnable function 5-3152
PLIB_TMR_Stop function 5-3029	PLIB_USART_ExistsReceiverFramingErrorStatus function 5-3153
PLIB_TMR_StopInIdleDisable function 5-3033	PLIB_USART_ExistsReceiverIdleStateLowEnable function 5-3153
PLIB_TMR_StopInIdleEnable function 5-3033	PLIB_USART_ExistsReceiverIdleStatus function 5-3154
PLIB_TMR_SystemClockFromTimerIsActive function 5-3059	PLIB_USART_ExistsReceiverInterruptMode function 5-3154
PLIB_TMR_TimerOscillatorDisable function 5-3029	PLIB_USART_ExistsReceiverMode function 5-3155
PLIB_TMR_TimerOscillatorEnable function 5-3030	PLIB_USART_ExistsReceiverOverrunStatus function 5-3155
PLIB_TMR_TriggerEventResetDisable function 5-3030	PLIB_USART_ExistsReceiverParityErrorStatus function 5-3156
PLIB_TMR_TriggerEventResetEnable function 5-3031	PLIB_USART_ExistsStopInIdle function 5-3156
plib_usart.h 5-3164	PLIB_USART_ExistsSyncClockPolarity function 5-3157
PLIB_USART_AlternateIODisable function 5-3105	
PLIB_USART_AlternateIOEnable function 5-3105	
PLIB_USART_BaudRateAutoDetectEnable function 5-3116	
PLIB_USART_BaudRateAutoDetectIsComplete function 5-3117	

PLIB_USART_ExistsSyncClockSource function 5-3158	PLIB_USART_StopInIdleDisable function 5-3111
PLIB_USART_ExistsSyncMode function 5-3158	PLIB_USART_StopInIdleEnable function 5-3111
PLIB_USART_ExistsTransmitter function 5-3159	PLIB_USART_SyncClockPolarityIdleHighDisable function 5-3112
PLIB_USART_ExistsTransmitter9BitsSend function 5-3159	PLIB_USART_SyncClockPolarityIdleHighEnable function 5-3113
PLIB_USART_ExistsTransmitterBreak function 5-3160	PLIB_USART_SyncClockSourceSelect function 5-3113
PLIB_USART_ExistsTransmitterBufferFullStatus function 5-3160	PLIB_USART_SyncModeSelect function 5-3114
PLIB_USART_ExistsTransmitterEmptyStatus function 5-3161	PLIB_USART_Transmitter9BitsSend function 5-3122
PLIB_USART_ExistsTransmitterEnable function 5-3161	PLIB_USART_TransmitterBreakSend function 5-3122
PLIB_USART_ExistsTransmitterIdleIsLow function 5-3162	PLIB_USART_TransmitterBreakSendIsComplete function 5-3123
PLIB_USART_ExistsTransmitterIdleStatus function 5-3162	PLIB_USART_TransmitterBufferIsFull function 5-3124
PLIB_USART_ExistsTransmitterInterruptMode function 5-3163	PLIB_USART_TransmitterByteSend function 5-3124
PLIB_USART_ExistsWakeOnStart function 5-3163	PLIB_USART_TransmitterDisable function 5-3125
PLIB_USART_HandshakeModeSelect function 5-3107	PLIB_USART_TransmitterEnable function 5-3125
PLIB_USART_IrDADisable function 5-3108	PLIB_USART_TransmitterIdleIsLowDisable function 5-3126
PLIB_USART_IrDAEnable function 5-3108	PLIB_USART_TransmitterIdleIsLowEnable function 5-3127
PLIB_USART_LineControlModeSelect function 5-3109	PLIB_USART_TransmitterInterruptModeSelect function 5-3127
PLIB_USART_LoopbackDisable function 5-3109	PLIB_USART_TransmitterIsEmpty function 5-3128
PLIB_USART_LoopbackEnable function 5-3110	PLIB_USART_TransmitterIsIdle function 5-3128
PLIB_USART_OperationModeSelect function 5-3110	PLIB_USART_WakeOnStartDisable function 5-3114
PLIB_USART_ReceiverAddressAutoDetectDisable function 5-3129	PLIB_USART_WakeOnStartEnable function 5-3115
PLIB_USART_ReceiverAddressAutoDetectEnable function 5-3129	PLIB_USART_WakeOnStartIsEnabled function 5-3115
PLIB_USART_ReceiverAddressDetectDisable function 5-3130	plib_usb.h 5-3313
PLIB_USART_ReceiverAddressDetectEnable function 5-3131	PLIB_USB_ActivityPending function 5-3260
PLIB_USART_ReceiverAddressIsReceived function 5-3131	PLIB_USB_AllInterruptEnable function 5-3188
PLIB_USART_ReceiverByteReceive function 5-3132	PLIB_USB_AutoSuspendDisable function 5-3189
PLIB_USART_ReceiverDataIsAvailable function 5-3133	PLIB_USB_AutoSuspendEnable function 5-3190
PLIB_USART_ReceiverDisable function 5-3133	PLIB_USB_BDTBaseAddressGet function 5-3201
PLIB_USART_ReceiverEnable function 5-3134	PLIB_USB_BDTBaseAddressSet function 5-3201
PLIB_USART_ReceiverFramingErrorHasOccurred function 5-3134	PLIB_USB_BufferAddressGet function 5-3202
PLIB_USART_ReceiverIdleStateLowDisable function 5-3135	PLIB_USB_BufferAddressSet function 5-3203
PLIB_USART_ReceiverIdleStateLowEnable function 5-3135	PLIB_USB_BufferAllCancelReleaseToUSB function 5-3203
PLIB_USART_ReceiverInterruptModeSelect function 5-3136	PLIB_USB_BufferByteCountGet function 5-3204
PLIB_USART_ReceiverIsIdle function 5-3136	PLIB_USB_BufferByteCountSet function 5-3205
PLIB_USART_ReceiverModeSelect function 5-3137	PLIB_USB_BufferCancelReleaseToUSB function 5-3205
PLIB_USART_ReceiverOverrunErrorClear function 5-3138	PLIB_USB_BufferClearAll function 5-3206
PLIB_USART_ReceiverOverrunHasOccurred function 5-3138	PLIB_USB_BufferClearAllDTSEnable function 5-3207
PLIB_USART_ReceiverParityErrorHasOccurred function 5-3139	PLIB_USB_BufferDataToggleGet function 5-3207

PLIB_USB_BufferDataToggleSelect function 5-3208	PLIB_USB_ErrorInterruptEnable function 5-3235
PLIB_USB_BufferDataToggleSyncDisable function 5-3209	PLIB_USB_ErrorInterruptFlagAllGet function 5-3235
PLIB_USB_BufferDataToggleSyncEnable function 5-3209	PLIB_USB_ErrorInterruptFlagClear function 5-3236
PLIB_USB_BufferEP0RxStatusInitialize function 5-3210	PLIB_USB_ErrorInterruptFlagGet function 5-3236
PLIB_USB_BufferIndexGet function 5-3211	PLIB_USB_ErrorInterruptFlagSet function 5-3237
PLIB_USB_BufferPIDBitsClear function 5-3211	PLIB_USB_ErrorInterruptIsEnabled function 5-3237
PLIB_USB_BufferPIDGet function 5-3212	PLIB_USB_ExistsActivityPending function 5-3283
PLIB_USB_BufferReleasedToSW function 5-3213	PLIB_USB_ExistsALL_Interrupt function 5-3284
PLIB_USB_BufferReleaseToUSB function 5-3214	PLIB_USB_ExistsAutomaticSuspend function 5-3284
PLIB_USB_BufferSchedule function 5-3214	PLIB_USB_ExistsBDTBaseAddress function 5-3285
PLIB_USB_BufferStallDisable function 5-3215	PLIB_USB_ExistsBDTFunctions function 5-3285
PLIB_USB_BufferStallEnable function 5-3216	PLIB_USB_ExistsBufferFreeze function 5-3286
PLIB_USB_BufferStallGet function 5-3217	PLIB_USB_ExistsDeviceAddress function 5-3287
PLIB_USB_DeviceAddressGet function 5-3190	PLIB_USB_ExistsEP0LowSpeedConnect function 5-3287
PLIB_USB_DeviceAddressSet function 5-3191	PLIB_USB_ExistsEP0NAKRetry function 5-3288
PLIB_USB_Disable function 5-3191	PLIB_USB_ExistsEPnRxEnable function 5-3289
PLIB_USB_Enable function 5-3192	PLIB_USB_ExistsEPnTxRx function 5-3289
PLIB_USB_EP0HostSetup function 5-3217	PLIB_USB_ExistsERR_Interrupt function 5-3290
PLIB_USB_EP0LSDirectConnectDisable function 5-3218	PLIB_USB_ExistsERR_InterruptStatus function 5-3290
PLIB_USB_EP0LSDirectConnectEnable function 5-3218	PLIB_USB_ExistsEyePattern function 5-3291
PLIB_USB_EP0NakRetryDisable function 5-3219	PLIB_USB_ExistsFrameNumber function 5-3291
PLIB_USB_EP0NakRetryEnable function 5-3219	PLIB_USB_ExistsGEN_Interrupt function 5-3292
PLIB_USB_EPnAttributesClear function 5-3220	PLIB_USB_ExistsGEN_InterruptStatus function 5-3293
PLIB_USB_EPnAttributesSet function 5-3221	PLIB_USB_ExistsHostBusyWithToken function 5-3293
PLIB_USB_EPnControlTransferDisable function 5-3221	PLIB_USB_ExistsHostGeneratesReset function 5-3294
PLIB_USB_EPnControlTransferEnable function 5-3222	PLIB_USB_ExistsLastDirection function 5-3294
PLIB_USB_EPnDirectionDisable function 5-3222	PLIB_USB_ExistsLastEndpoint function 5-3295
PLIB_USB_EPnHandshakeDisable function 5-3223	PLIB_USB_ExistsLastPingPong function 5-3295
PLIB_USB_EPnHandshakeEnable function 5-3223	PLIB_USB_ExistsLastTransactionDetails function 5-3296
PLIB_USB_EPnIsStalled function 5-3224	PLIB_USB_ExistsLiveJState function 5-3296
PLIB_USB_EPnRxDisable function 5-3225	PLIB_USB_ExistsLiveSingleEndedZero function 5-3297
PLIB_USB_EPnRxEnable function 5-3225	PLIB_USB_ExistsModuleBusy function 5-3297
PLIB_USB_EPnRxSelect function 5-3226	PLIB_USB_ExistsModulePower function 5-3298
PLIB_USB_EPnStallClear function 5-3226	PLIB_USB_ExistsNextTokenSpeed function 5-3298
PLIB_USB_EPnTxDisable function 5-3227	PLIB_USB_ExistsOnChipPullup function 5-3299
PLIB_USB_EPnTxEnable function 5-3227	PLIB_USB_ExistsOnChipTransceiver function 5-3299
PLIB_USB_EPnTxRxSelect function 5-3228	PLIB_USB_ExistsOpModeSelect function 5-3300
PLIB_USB_EPnTxSelect function 5-3229	PLIB_USB_ExistsOTG_ASessionValid function 5-3300
PLIB_USB_ErrorInterruptDisable function 5-3234	PLIB_USB_ExistsOTG_BSessionEnd function 5-3301

PLIB_USB_ExistsOTG_IDPinState function 5-3301	PLIB_USB_JStateIsActive function 5-3261
PLIB_USB_ExistsOTG_Interrupt function 5-3302	PLIB_USB_LastTransactionDetailsGet function 5-3238
PLIB_USB_ExistsOTG_InterruptStatus function 5-3302	PLIB_USB_LastTransactionDirectionGet function 5-3239
PLIB_USB_ExistsOTG_LineState function 5-3303	PLIB_USB_LastTransactionEndPtGet function 5-3239
PLIB_USB_ExistsOTG_PullUpPullDown function 5-3303	PLIB_USB_LastTransactionPingPongStateGet function 5-3240
PLIB_USB_ExistsOTG_SessionValid function 5-3304	PLIB_USB_ModuleIsBusy function 5-3275
PLIB_USB_ExistsOTG_VbusCharge function 5-3304	PLIB_USB_OnChipPullUpDisable function 5-3194
PLIB_USB_ExistsOTG_VbusDischarge function 5-3305	PLIB_USB_OnChipPullUpEnable function 5-3194
PLIB_USB_ExistsOTG_VbusPowerOnOff function 5-3305	PLIB_USB_OperatingModeSelect function 5-3195
PLIB_USB_ExistsPacketTransfer function 5-3306	PLIB_USB_OTG_BSessionHasEnded function 5-3249
PLIB_USB_ExistsPingPongMode function 5-3306	PLIB_USB_OTG_IDPinStateIsTypeA function 5-3249
PLIB_USB_ExistsResumeSignaling function 5-3307	PLIB_USB_OTG_InterruptDisable function 5-3257
PLIB_USB_ExistsSleepEntryGuard function 5-3308	PLIB_USB_OTG_InterruptEnable function 5-3257
PLIB_USB_ExistsSOFThreshold function 5-3308	PLIB_USB_OTG_InterruptFlagClear function 5-3258
PLIB_USB_ExistsSpeedControl function 5-3309	PLIB_USB_OTG_InterruptFlagGet function 5-3258
PLIB_USB_ExistsStartOfFrames function 5-3309	PLIB_USB_OTG_InterruptFlagSet function 5-3259
PLIB_USB_ExistsStopInIdle function 5-3310	PLIB_USB_OTG_InterruptIsEnabled function 5-3259
PLIB_USB_ExistsSuspend function 5-3310	PLIB_USB_OTG_LineStateIsStable function 5-3250
PLIB_USB_ExistsTokenEP function 5-3311	PLIB_USB_OTG_PullUpPullDownSetup function 5-3251
PLIB_USB_ExistsTokenPID function 5-3311	PLIB_USB_OTG_SessionValid function 5-3251
PLIB_USB_ExistsUOEMonitor function 5-3312	PLIB_USB_OTG_VBusChargeDisable function 5-3252
PLIB_USB_ExternalComparatorMode2Pin function 5-3267	PLIB_USB_OTG_VBusChargeEnable function 5-3252
PLIB_USB_ExternalComparatorMode3Pin function 5-3267	PLIB_USB_OTG_VBusChargeTo3V function 5-3253
PLIB_USB_EyePatternDisable function 5-3274	PLIB_USB_OTG_VBusChargeTo5V function 5-3253
PLIB_USB_EyePatternEnable function 5-3274	PLIB_USB_OTG_VBusDischargeDisable function 5-3254
PLIB_USB_FrameNumberGet function 5-3261	PLIB_USB_OTG_VBusDischargeEnable function 5-3254
PLIB_USB_FullSpeedDisable function 5-3193	PLIB_USB_OTG_VBusPowerOff function 5-3255
PLIB_USB_FullSpeedEnable function 5-3193	PLIB_USB_OTG_VBusPowerOn function 5-3255
PLIB_USB_I2CInterfaceForExtModuleDisable function 5-3265	PLIB_USB_OTG_VBusValid function 5-3256
PLIB_USB_I2CInterfaceForExtModuleEnable function 5-3265	PLIB_USB_PacketTransferDisable function 5-3262
PLIB_USB_InterruptDisable function 5-3229	PLIB_USB_PacketTransferEnable function 5-3263
PLIB_USB_InterruptEnable function 5-3230	PLIB_USB_PacketTransferIsDisabled function 5-3263
PLIB_USB_InterruptEnableGet function 5-3230	PLIB_USB_PingPongFreeze function 5-3276
PLIB_USB_InterruptFlagAllGet function 5-3231	PLIB_USB_PingPongModeGet function 5-3195
PLIB_USB_InterruptFlagClear function 5-3231	PLIB_USB_PingPongModeSelect function 5-3196
PLIB_USB_InterruptFlagGet function 5-3232	PLIB_USB_PingPongReset function 5-3276
PLIB_USB_InterruptFlagSet function 5-3233	PLIB_USB_PingPongUnfreeze function 5-3277
PLIB_USB_InterruptIsEnabled function 5-3233	PLIB_USB_PWMCounterDisable function 5-3268
PLIB_USB_IsBusyWithToken function 5-3241	PLIB_USB_PWMCounterEnable function 5-3268

PLIB_USB_PWMDisable function 5-3269	PLIB_WDT_ExistsPostscalarValue function 5-3328
PLIB_USB_PWMEnable function 5-3269	PLIB_WDT_ExistsTimerClear function 5-3329
PLIB_USB_PWMPolarityActiveLow function 5-3270	PLIB_WDT_ExistsWindowEnable function 5-3329
PLIB_USB_PWMPolarityActiveHigh function 5-3270	PLIB_WDT_IsEnabled function 5-3326
PLIB_USB_ResetSignalDisable function 5-3246	PLIB_WDT_PostscalarValueGet function 5-3327
PLIB_USB_ResetSignalEnable function 5-3247	PLIB_WDT_TimerClear function 5-3325
PLIB_USB_ResumeSignalingDisable function 5-3248	PLIB_WDT_WindowDisable function 5-3324
PLIB_USB_ResumeSignalingEnable function 5-3248	PLIB_WDT_WindowEnable function 5-3325
PLIB_USB_SE0InProgress function 5-3264	PMP Peripheral Library 5-2478
PLIB_USB_SleepGuardDisable function 5-3197	PMP_ACK_MODE enumeration 5-2546
PLIB_USB_SleepGuardEnable function 5-3197	PMP_ADDRESS_HOLD_LATCH_WAIT_STATES enumeration 5-2546
PLIB_USB_SOFDisable function 5-3241	PMP_ADDRESS_LATCH enumeration 5-2547
PLIB_USB_SOFEnable function 5-3242	PMP_ADDRESS_LATCH_WAIT_STATES enumeration 5-2547
PLIB_USB_SOFThresholdGet function 5-3242	PMP_ADDRESS_PORT enumeration 5-2548
PLIB_USB_SOFThresholdSet function 5-3243	PMP_ALTERNATE_MASTER_WAIT_STATES enumeration 5-2549
PLIB_USB_StopInIdleDisable function 5-3198	PMP_CHIP_SELECT enumeration 5-2549
PLIB_USB_StopInIdleEnable function 5-3198	PMP_CHIPSELECT_FUNCTION enumeration 5-2550
PLIB_USB_SuspendDisable function 5-3199	PMP_DATA_HOLD_STATES enumeration 5-2550
PLIB_USB_SuspendEnable function 5-3199	PMP_DATA_LENGTH enumeration 5-2551
PLIB_USB_TokenEPGet function 5-3244	PMP_DATA_SIZE enumeration 5-2551
PLIB_USB_TokenEPSet function 5-3244	PMP_DATA_WAIT_STATES enumeration 5-2551
PLIB_USB_TokenPIDGet function 5-3245	PMP_INCREMENT_MODE enumeration 5-2552
PLIB_USB_TokenPIDSet function 5-3245	PMP_INPUT_BUFFER_TYPE enumeration 5-2552
PLIB_USB_TokenSend function 5-3277	PMP_INTERRUPT_MODE enumeration 5-2553
PLIB_USB_TokenSpeedSelect function 5-3246	PMP_MASTER_MODE enumeration 5-2553
PLIB_USB_TransceiverDisable function 5-3266	PMP_MODULE_ID enumeration 5-2554
PLIB_USB_TransceiverEnable function 5-3266	PMP_MUX_MODE enumeration 5-2554
PLIB_USB_UOEMonitorDisable function 5-3200	PMP_OPERATION_MODE enumeration 5-2555
PLIB_USB_UOEMonitorEnable function 5-3200	PMP_PMBE_PORT enumeration 5-2555
PLIB_USB_VBoostDisable function 5-3271	PMP_POLARITY_LEVEL enumeration 5-2556
PLIB_USB_VBoostEnable function 5-3271	PMP_STROBE_WAIT_STATES enumeration 5-2556
PLIB_USB_VBUSComparatorDisable function 5-3272	Porting Applications from the MLA Graphics Library to the MPLAB Harmony Graphics Library 5-686
PLIB_USB_VBUSComparatorEnable function 5-3272	Porting Applications from the V6 Beta TCP/IP Stack to the MPLAB Harmony TCP/IP Stack 5-3557
PLIB_USB_VBUSPullUpDisable function 5-3273	Porting the MPLAB Harmony Drivers and Peripheral Library 5-3565
PLIB_USB_VBUSPullUpEnable function 5-3273	Ports Change Notification 5-2599
plib_wdt.h 5-3330	Ports Control 5-2598, 5-3471
PLIB_WDT_Disable function 5-3323	
PLIB_WDT_Enable function 5-3324	
PLIB_WDT_ExistsEnableControl function 5-3328	

Ports Function Remap 5-2599
 Ports Peripheral Library 5-2593
 Ports System Service Library 5-3465
 PORTS_ANALOG_PIN enumeration 5-2629
 PORTS_BIT_POS enumeration 5-2631
 PORTS_CHANGE_NOTICE_PIN enumeration 5-2632
 PORTS_CHANNEL enumeration 5-2635
 PORTS_DATA_MASK type 5-2636
 PORTS_DATA_TYPE type 5-2636
 PORTS_MODULE_ID enumeration 5-2636
 PORTS_PERIPHERAL_OD enumeration 5-2637
 PORTS_PIN enumeration 5-2638
 PORTS_PIN_MODE enumeration 5-2638
 PORTS_REMAP_FUNCTION enumeration 5-2639
 PORTS_REMAP_INPUT_FUNCTION enumeration 5-2642
 PORTS_REMAP_INPUT_PIN enumeration 5-2643
 PORTS_REMAP_OUTPUT_FUNCTION enumeration 5-2644
 PORTS_REMAP_OUTPUT_PIN enumeration 5-2645
 PORTS_REMAP_PIN enumeration 5-2646
 Power Peripheral Library 5-2662
 POWER_MODULE enumeration 5-2677
 POWER_MODULE_ID enumeration 5-2680
 Power-Saving Modes 5-1374, 5-1672, 5-2822
 Prefetch Cache Peripheral Library 5-2427
 Prefetch Control Operations 5-2433
 Prefetch Status Operations 5-2433
 Previous Versions 5-3964
 primitive_demo 3-87
 primitiveTaskImage variable 5-888
 PRIV_LOCALIZED_PASSWORD_KEY_LEN macro 5-3827
 Project Layout 1-10
 PUTIMAGE_PARAM structure 5-939
 PWM Mode with Enabled Faults 5-2293

Q

Queue Operation 5-1319
 QUEUE_ELEMENT_OBJECT type 5-393

R

RdiaCosine function 5-797
 RdiaSine function 5-798
 REACHABLE_TIME macro 5-3803
 Reading a Capture Value 5-2174
 Reading a File 5-3394
 Reboot TCP/IP Stack Library 5-3806
 REBOOT_PORT macro 5-3808
 REBOOT_SAME_SUBNET_ONLY macro 5-3808
 Receive 5-1919
 Receive Filtering Overview 5-1912
 Receiving a CAN Message 5-1562
 Receiving a Report 5-4140
 Receiving Data 5-4074
 recv function 5-3605
 recvfrom function 5-3606
 RED macro 5-1049
 Reduced Media Independent Interface (RMII) 5-1910
 Release Contents 1-25
 Release Notes 1-23, 2-40, 3-64, 3-84, 3-99, 3-102, 3-113, 3-123, 3-130, 3-142, 3-149, 4-178, 5-194, 5-227, 5-254, 5-289, 5-299, 5-309, 5-320, 5-333, 5-343, 5-367, 5-397, 5-419, 5-457, 5-511, 5-552, 5-601, 5-680, 5-695, 5-1110, 5-1127, 5-1167, 5-1275, 5-1312, 5-1362, 5-1472, 5-1520, 5-1554, 5-1668, 5-1734, 5-1865, 5-1904, 5-2060, 5-2171, 5-2203, 5-2253, 5-2288, 5-2331, 5-2428, 5-2478, 5-2594, 5-2662, 5-2688, 5-2704, 5-2762, 5-2799, 5-2892, 5-3011, 5-3086, 5-3170, 5-3320, 5-3348, 5-3363, 5-3377, 5-3419, 5-3439, 5-3466, 5-3500, 5-3530, 5-3536, 5-3546, 5-3570, 5-3575, 5-3598, 5-3615, 5-3625, 5-3637, 5-3646, 5-3654, 5-3659, 5-3690, 5-3698, 5-3709, 5-3718, 5-3748, 5-3771, 5-3795, 5-3799, 5-3806, 5-3809, 5-3822, 5-3858, 5-3868, 5-3881, 5-3913, 5-3917, 5-3930, 5-3955, 5-3964, 5-3978, 5-4004, 5-4060, 5-4100, 5-4131, 5-4169, 5-4180, 5-4200, 6-4226, 7-4230, 7-4232, 7-4234, 7-4236
 Release Types 1-30
 Rendering 5-698
 RequestEntirePaletteChange macro 5-1049
 RequestPaletteChange function 5-876
 Required Hardware 3-103, 3-124, 3-143
 Reset Peripheral Library 5-2687
 RESET_CONFIG_REG_READ_ERROR enumeration 5-2695
 RESET_MODULE_ID enumeration 5-2694

-
- RESET_NMI_COUNT_TYPE type 5-2695
 - RESET_NMI_REASON enumeration 5-2695
 - RETRANS_TIMER macro 5-3803
 - RTCC Mode Operations 5-2708
 - RTCC Peripheral Library 5-2703
 - RTCC_ALARM_MASK_CONFIGURATION enumeration 5-2747
 - RTCC_MODULE_ID enumeration 5-2748
 - RTCC_VALUE_REGISTER_POINTER enumeration 5-2748
 - RTOS Demonstrations 3-113
 - RTR_SOLICITATION_INTERVAL macro 5-3804
 - Running the Application 3-111, 3-128, 3-147
 - Running the Configurator Plug-in 2-44
 - Running the Demonstration 3-67, 3-70, 3-72, 3-74, 3-76, 3-86, 3-88, 3-90, 3-91, 3-115, 3-116, 3-118, 3-131, 3-133, 3-135, 3-153, 3-155, 3-156, 3-159, 3-160, 3-161, 3-162, 3-164, 3-165, 3-167, 3-168, 3-170
 - S**
 - S1D13517 Data Types and Constants 5-331
 - S1D13517 Graphics Controller Driver Library 5-319
 - SADDLEBROWN macro 5-1049
 - Sample Code 5-1912
 - SAT16 function 5-1263
 - SAT16N function 5-1264
 - SAT16P function 5-1264
 - SB_MODULE_ID enumeration 5-2786
 - SCAN_BS_PRESSED macro 5-1049
 - SCAN_BS_RELEASED macro 5-1050
 - SCAN_CR_PRESSED macro 5-1050
 - SCAN_CR_RELEASED macro 5-1050
 - SCAN_CRA_PRESSED macro 5-1050
 - SCAN_CRA_RELEASED macro 5-1050
 - SCAN_DEL_PRESSED macro 5-1050
 - SCAN_DEL_RELEASED macro 5-1051
 - SCAN_DOWN_PRESSED macro 5-1051
 - SCAN_DOWN_RELEASED macro 5-1051
 - SCAN_END_PRESSED macro 5-1051
 - SCAN_END_RELEASED macro 5-1051
 - SCAN_HOME_PRESSED macro 5-1051
 - SCAN_HOME_RELEASED macro 5-1051
 - SCAN_LEFT_PRESSED macro 5-1052
 - SCAN_LEFT_RELEASED macro 5-1052
 - SCAN_PGDOWN_PRESSED macro 5-1052
 - SCAN_PGDOWN_RELEASED macro 5-1052
 - SCAN_PGUP_PRESSED macro 5-1052
 - SCAN_PGUP_RELEASED macro 5-1052
 - SCAN_RIGHT_PRESSED macro 5-1053
 - SCAN_RIGHT_RELEASED macro 5-1053
 - SCAN_SPACE_PRESSED macro 5-1053
 - SCAN_SPACE_RELEASED macro 5-1053
 - SCAN_TAB_PRESSED macro 5-1053
 - SCAN_TAB_RELEASED macro 5-1053
 - SCAN_UP_PRESSED macro 5-1053
 - SCAN_UP_RELEASED macro 5-1054
 - Scheme Functionality 5-701
 - sd_card_fat_single_disk 3-69
 - SDCARD_MAX_LIMIT macro 5-415
 - Secure Digital (SD) Card Driver Library 5-397
 - Semaphores 5-1318
 - send function 5-3607
 - Sending a Report 5-4139
 - Sending Data 5-4072
 - sendto function 5-3607
 - Service Discovery 5-3957
 - SET_PA6_AS_OUTPUT macro 5-674
 - SetEntirePalette macro 5-1054
 - SetPalette function 5-876
 - SetPaletteBpp function 5-877
 - SetPaletteFlash function 5-877
 - Setting Bus Speed 5-1558
 - SHOW_DATA macro 5-1054
 - SIENNA macro 5-1054
 - SIN45 macro 5-1055
 - Single Compare Set High Mode 5-2291
 - Single Compare Toggle Mode 5-2292
 - Slave Mode 5-2821
 - SMI_SCAN_DATA_STATUS enumeration 5-285
 - SMTP Client Examples 5-3810
 - SMTP Client Long Message Example 5-3812
-

SMTP Client Short Message Example 5-3811	SNMP_V3 macro 5-3847
SMTP TCP/IP Stack Library 5-3809	SNMP_VAL union 5-3851
smtp.h 5-3821	SNMP_WRITE_COMMUNITIES macro 5-3828
SMTP_CLIENT_MODULE_CONFIG structure 5-3820	SNMPGetExactIndex function 5-3841
smtp_config.h 5-3822	SNMPGetNextIndex function 5-3841
SMTP_CONNECT_ERROR macro 5-3819	SNMPGetVar function 5-3842
SMTP_POINTERS structure 5-3820	SNMPIsValidSetLen function 5-3842
SMTP_PORT macro 5-3814	SNMPSendTrap function 5-3843
SMTP_RESOLVE_ERROR macro 5-3819	SNMPSetVar function 5-3843
SMTP_SERVER_REPLY_TIMEOUT macro 5-3814	snmpv3.h 5-3856
SMTP_SUCCESS macro 5-3820	SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE macro 5-3829
SNMP TCP/IP Stack Library 5-3822	snmpv3_config.h 5-3857
snmp.h 5-3854	SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE macro 5-3829
SNMP_BIB_FILE_NAME macro 5-3827	SNMPV3_PROCESSING_MEM_INFO_PTRS structure 5-3852
SNMP_BUFFER_DATA structure 5-3848	SNMPV3_STACK_DCPT_STUB structure 5-3852
SNMP_COMMUNITY_MAX_LEN macro 5-3827	SNMPV3_TRAP_MSG_PROCESS_MODEL_DB macro 5-3829
SNMP_COMMUNITY_TYPE enumeration 5-3848	SNMPV3_TRAP_SECURITY_LEVEL_DB macro 5-3829
snmp_config.h 5-3855	SNMPV3_TRAP_SECURITY_MODEL_TYPE_DB macro 5-3829
SNMP_END_OF_VAR macro 5-3846	SNMPV3_TRAP_USER_SECURITY_NAME_DB macro 5-3829
SNMP_H macro 5-3846	SNMPV3_USER_AUTH_PASSWD_DB macro 5-3830
SNMP_ID type 5-3849	SNMPV3_USER_AUTH_TYPE_DB macro 5-3830
SNMP_INDEX type 5-3849	SNMPV3_USER_PRIV_PASSWD_DB macro 5-3830
SNMP_INDEX_INVALID macro 5-3847	SNMPV3_USER_PRIV_TYPE_DB macro 5-3830
SNMP_MAX_COMMUNITY_SUPPORT macro 5-3827	SNMPV3_USER_SECURITY_NAME_DB macro 5-3830
SNMP_MAX_MSG_SIZE macro 5-3828	SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE macro 5-3830
SNMP_MAX_NON_REC_ID_OID macro 5-3828	SNMPV3_USM_MAX_USER macro 5-3831
SNMP_MODULE_CONFIG structure 5-3849	SNMPv3ComputeHMACIpadOpadForAuthLocalzedKey function 5-3845
SNMP_NET_CONFIG structure 5-3831	SNMPv3USMAuthPrivPswdLocalization function 5-3845
SNMP_NON_MIB_RECD_INFO structure 5-3849	SNMPValidateCommunity function 5-3844
SNMP_NOTIFY_INFO structure 5-3849	SNTP TCP/IP Stack Library 5-3858
SNMP_PROCESSING_MEM_INFO_PTRS structure 5-3850	sntp.h 5-3866
SNMP_READ_COMMUNITIES macro 5-3828	sntp_config.h 5-3867
SNMP_STACK_DCPT_STUB structure 5-3850	SNTP_MODULE_CONFIG structure 5-3865
SNMP_START_OF_VAR macro 5-3847	SNTP_RESULT enumeration 5-3865
SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE macro 5-3828	SOCK_DGRAM macro 5-3610
SNMP_TRAP_INFO structure 5-3850	
SNMP_TRAP_IP_ADDRESS_TYPE enumeration 5-3851	
SNMP_V1 macro 5-3847	
SNMP_V2C macro 5-3847	

-
- SOCK_STREAM macro 5-3610
 - sockaddr structure 5-3612
 - SOCKADDR type 5-3611
 - sockaddr_in structure 5-3612
 - SOCKADDR_IN type 5-3611
 - socket function 5-3608
 - SOCKET type 5-3611
 - SOCKET_CNXN_IN_PROGRESS macro 5-3610
 - SOCKET_DISCONNECTED macro 5-3610
 - SOCKET_ERROR macro 5-3610
 - Software Drivers 5-299
 - Source Files to Include 5-3973
 - Source Interrupt Management 5-3423
 - Source/Destination/Peripheral Address Management 5-1745
 - Source/Destination/Peripheral Data Management 5-1746
 - Special Considerations 5-2600, 5-3473
 - Special Function Modules (CRC) 5-1747
 - SPI Driver Demonstrations 3-123
 - SPI Driver Library 5-417
 - SPI Error Handling 5-2823
 - SPI Master Mode and Frame Master Mode 5-2811
 - SPI Master Mode and Frame Slave Mode 5-2812
 - SPI Master Mode Clock Frequency 5-2823
 - SPI Peripheral Library 5-2798
 - SPI Receive-only Operation 5-2823
 - SPI Slave Mode and Frame Master Mode 5-2814, 5-2815
 - SPI_AUDIO_COMMUNICATION_WIDTH enumeration 5-2859
 - SPI_AUDIO_ERROR enumeration 5-2859
 - SPI_AUDIO_PROTOCOL enumeration 5-2859
 - SPI_AUDIO_TRANSMIT_MODE enumeration 5-2860
 - SPI_BAUD_RATE_CLOCK enumeration 5-2860
 - SPI_CLOCK_POLARITY enumeration 5-2860
 - SPI_COMMUNICATION_WIDTH enumeration 5-2861
 - SPI_DATA_TYPE type 5-2861
 - SPI_ERROR_INTERRUPT enumeration 5-2861
 - SPI_FIFO_INTERRUPT enumeration 5-2862
 - SPI_FIFO_TYPE enumeration 5-2863
 - SPI_FRAME_PULSE_DIRECTION enumeration 5-2863
 - SPI_FRAME_PULSE_EDGE enumeration 5-2863
 - SPI_FRAME_PULSE_POLARITY enumeration 5-2864
 - SPI_FRAME_PULSE_WIDTH enumeration 5-2864
 - SPI_FRAME_SYNC_PULSE enumeration 5-2864
 - SPI_INPUT_SAMPLING_PHASE enumeration 5-2865
 - SPI_MODULE_ID enumeration 5-2865
 - SPI_OUTPUT_DATA_PHASE enumeration 5-2866
 - SPI_PIN enumeration 5-2866
 - SPI_PRESCALE_PRIMARY enumeration 5-2867
 - SPI_PRESCALE_SECONDARY enumeration 5-2867
 - SQI Peripheral Library 5-2891
 - SQI_ADDR_BYTES enumeration 5-2966
 - SQI_BD_CTRL_WORD enumeration 5-2966
 - SQI_BD_STATE enumeration 5-2967
 - SQI_CLK_DIV enumeration 5-2968
 - SQI_CS_NUM enumeration 5-2969
 - SQI_CS_OEN enumeration 5-2969
 - SQI_DATA_FORMAT enumeration 5-2969
 - SQI_DATA_MODE enumeration 5-2970
 - SQI_DATA_OEN enumeration 5-2970
 - SQI_DATA_TYPE type 5-2970
 - SQI_DUMMY_BYTES enumeration 5-2971
 - SQI_INTERRUPTS enumeration 5-2971
 - SQI_LANE_MODE enumeration 5-2972
 - SQI_MODE_BYTES enumeration 5-2972
 - SQI_MODULE_ID enumeration 5-2973
 - SQI_XFER_CMD enumeration 5-2973
 - SQI_XFER_MODE enumeration 5-2974
 - ssd_demo 3-90
 - SSD1926 Graphics Controller Driver Library 5-333
 - SSL Client Support 5-3869
 - SSL Limitations 5-3870
 - SSL Server Support 5-3870
 - SSL TCP/IP Stack Library 5-3867
 - ssl.h 5-3879
 - ssl_config.h 5-3880
 - SSL_MAX_BUFFERS macro 5-3873
 - SSL_MAX_CONNECTIONS macro 5-3873
 - SSL_MAX_HASHES macro 5-3873
 - SSL_MAX_SESSIONS macro 5-3874
-

- SSL_MIN_SESSION_LIFETIME macro 5-3874
- SSL_MODULE_CONFIG structure 5-3879
- SSL_MULTIPLE_INTERFACES macro 5-3874
- SSL_RSA_CLIENT_SIZE macro 5-3874
- SSL_RSA_KEY_SIZE macro 5-3874
- SSL_RSA_LIFETIME_EXTENSION macro 5-3874
- SSL_VERSION macro 5-3875
- SSL_VERSION_HI macro 5-3875
- SSL_VERSION_LO macro 5-3875
- Standard ID Message Format 5-1555
- Standard Master Mode 5-2804
- Standard Slave Mode 5-2805
- Standard SPI Mode 5-2804
- Start Here 1-1
- State Machine 5-2255, 5-2485, 5-2707, 5-2802, 5-3088
- State-Driven Applications 3-101
- States 5-698
- Static Configuration 5-3425
- Status (including Channel) 5-1748
- Support for Legacy "Ethernet Controller Library" 5-232, 5-1920
- Supported Demonstration Boards 3-67, 3-69, 3-71, 3-73, 3-75, 3-86, 3-87, 3-89, 3-90, 3-103, 3-114, 3-115, 3-116, 3-117, 3-124, 3-143, 3-153, 3-154, 3-156, 3-158, 3-160, 3-161, 3-162, 3-164, 3-166, 3-167, 3-169
- Supported Hardware 3-131, 3-133, 3-135
- Supported Libraries 4-181
- SW License Agreement 2-41, 3-64, 3-85, 3-99, 3-102, 3-113, 3-123, 3-130, 3-142, 3-150, 5-194, 5-227, 5-254, 5-289, 5-299, 5-309, 5-320, 5-334, 5-343, 5-368, 5-398, 5-419, 5-458, 5-512, 5-553, 5-606, 5-680, 5-695, 5-1111, 5-1128, 5-1168, 5-1276, 5-1312, 5-1362, 5-1472, 5-1520, 5-1554, 5-1668, 5-1734, 5-1865, 5-1904, 5-2060, 5-2172, 5-2203, 5-2253, 5-2289, 5-2331, 5-2428, 5-2479, 5-2594, 5-2663, 5-2688, 5-2705, 5-2762, 5-2799, 5-2892, 5-3012, 5-3086, 5-3170, 5-3320, 5-3348, 5-3363, 5-3377, 5-3420, 5-3439, 5-3466, 5-3501, 5-3530, 5-3539, 5-3546, 5-3570, 5-3575, 5-3598, 5-3615, 5-3625, 5-3637, 5-3647, 5-3655, 5-3660, 5-3690, 5-3698, 5-3709, 5-3718, 5-3748, 5-3771, 5-3795, 5-3799, 5-3806, 5-3809, 5-3822, 5-3859, 5-3868, 5-3881, 5-3913, 5-3918, 5-3931, 5-3955, 5-3966, 5-3978, 5-4005, 5-4061, 5-4101, 5-4131, 5-4169, 5-4180, 5-4200, 6-4226, 7-4230, 7-4232, 7-4234, 7-4236
- Sync Mode Selection 5-463
- Synchronizing Timer 5-2174
- Synchronous External Clock Counter 5-3016
- Synchronous Internal Clock Counter 5-3015
- Synchronous Master Mode 5-3095
- Synchronous Slave Mode 5-3097
- sys_clk.h 5-3362
- SYS_CLK_ClockFailureCallbackRegister function 5-3353
- SYS_CLK_ClockFrequencyGet function 5-3355
- SYS_CLK_ClockFrequencySet function 5-3355
- SYS_CLK_ClockIsReady function 5-3356
- SYS_CLK_FRCTune function 5-3356
- SYS_CLK_FRCTUNING_DATA structure 5-3358
- SYS_CLK_FRCTuningSet function 5-3357
- SYS_CLK_INIT structure 5-3358
- SYS_CLK_Initialize function 5-3354
- SYS_CLK_MZPeriphBusFreqGet function 5-3357
- SYS_CLK_OUTPUT enumeration 5-3359
- SYS_CLK_SOURCE enumeration 5-3360
- sys_common.h 5-3345
- sys_devcon.h 5-3375
- sys_devcon_config.h 5-3375
- SYS_DEVCON_Deinitialize function 5-3369
- SYS_DEVCON_HANDLE type 5-3373
- SYS_DEVCON_INDEX_0 macro 5-3374
- SYS_DEVCON_INIT structure 5-3374
- SYS_DEVCON_Initialize function 5-3369
- SYS_DEVCON_PerformanceConfig function 5-3370
- SYS_DEVCON_PIC32MX_MAX_PB_FREQ macro 5-3368
- SYS_DEVCON_PRIMARY_CLOCK macro 5-3368
- SYS_DEVCON_Reinitialize function 5-3371
- SYS_DEVCON_Status function 5-3371
- SYS_DEVCON_SYSTEM_CLOCK macro 5-3368
- SYS_DEVCON_Tasks function 5-3372
- sys_fs.h 5-3418
- SYS_FS_ERROR enumeration 5-3411
- SYS_FS_FILE_OPEN_ATTRIBUTES enumeration 5-3412
- SYS_FS_FILE_SEEK_CONTROL enumeration 5-3413
- SYS_FS_FILE_SYSTEM_TYPE enumeration 5-3414
- SYS_FS_FileClose function 5-3398
- SYS_FS_FileEOF function 5-3399
- SYS_FS_FileError function 5-3407
- SYS_FS_FileNameGet function 5-3406

SYS_FS_FileOpen function 5-3400	SYS_MODULE_INDEX type 5-3335
SYS_FS_FileRead function 5-3401	SYS_MODULE_INIT union 5-3335
SYS_FS_FileSeek function 5-3401	SYS_MODULE_INITIALIZE_ROUTINE type 5-3336
SYS_FS_FileSize function 5-3402	SYS_MODULE_INTERFACE structure 5-3337
SYS_FS_FileStat function 5-3403	SYS_MODULE_OBJ type 5-3338
SYS_FS_FileTell function 5-3404	SYS_MODULE_OBJ_INVALID macro 5-3342
SYS_FS_FileWrite function 5-3405	SYS_MODULE_OBJ_STATIC macro 5-3343
SYS_FS_FSTAT structure 5-3414	SYS_MODULE_POWER_IDLE_RUN macro 5-3343
SYS_FS_FUNCTIONS structure 5-3415	SYS_MODULE_POWER_IDLE_STOP macro 5-3343
SYS_FS_HANDLE type 5-3416	SYS_MODULE_POWER_OFF macro 5-3344
SYS_FS_HANDLE_INVALID macro 5-3417	SYS_MODULE_POWER_RUN_FULL macro 5-3344
SYS_FS_Initialize function 5-3407	SYS_MODULE_POWER_SLEEP macro 5-3344
SYS_FS_Mount function 5-3408	SYS_MODULE_REINITIALIZE_ROUTINE type 5-3338
SYS_FS_REGISTRATION_TABLE structure 5-3416	SYS_MODULE_STATUS_ROUTINE type 5-3339
SYS_FS_RESULT enumeration 5-3416	SYS_MODULE_TASKS_DATA structure 5-3339
SYS_FS_Tasks 5-3397	SYS_MODULE_TASKS_ROUTINE type 5-3340
SYS_FS_Tasks function 5-3410	SYS_MSB_MailboxClose function 5-3449
SYS_FS_Unmount function 5-3410	SYS_MSB_MailboxOpen function 5-3449
SYS_Initialize function 5-3373	SYS_MSB_MailboxReinit function 5-3450
sys_int.h 5-3437	sys_msg.h 5-3463
SYS_INT_Disable function 5-3429	SYS_MSG_BUFFER_SIZES macro 5-3458
SYS_INT_DynamicDeregister function 5-3428	sys_msg_config.h 5-3464
SYS_INT_DynamicRegister function 5-3428	SYS_MSG_Deinitialize function 5-3445
SYS_INT_Enable function 5-3430	SYS_MSG_GotMessages function 5-3454
SYS_INT_Initialize function 5-3427	SYS_MSG_ID2hMsgType function 5-3456
SYS_INT_IsEnabled function 5-3430	SYS_MSG_INIT structure 5-3459
SYS_INT_SourceDisable function 5-3431	SYS_MSG_Initialize function 5-3446
SYS_INT_SourceEnable function 5-3432	SYS_MSG_INSTANCE enumeration 5-3460
SYS_INT_SourceIsEnabled function 5-3432	SYS_MSG_MAILBOXES_ADDONE macro 5-3462
SYS_INT_SourceStatusClear function 5-3433	SYS_MSG_MailboxMessagesGet function 5-3450
SYS_INT_SourceStatusGet function 5-3433	SYS_MSG_MailboxMsgAdd function 5-3451
SYS_INT_SourceStatusSet function 5-3434	SYS_MSG_MailboxMsgRemove function 5-3452
SYS_INT_TASKS_POINTER type 5-3437	SYS_MSG_MAX_MAILBOXES macro 5-3458
SYS_INT_VectorPrioritySet function 5-3435	SYS_MSG_MAX_MSGS_DELIVERED macro 5-3458
SYS_INT_VectorSubprioritySet function 5-3436	SYS_MSG_MAX_PRIORITY macro 5-3459
SYS_MEDIA_EVENT enumeration 5-415	SYS_MSG_MAX_TYPES macro 5-3459
sys_module.h 5-3346	SYS_MSG_MessageDeliver function 5-3454
SYS_MODULE_DATA structure 5-3334	SYS_MSG_MessageReceive function 5-3455
SYS_MODULE_DEINITIALIZE_ROUTINE type 5-3335	SYS_MSG_MessageSend function 5-3456

SYS_MSG_NUM_MAILBOX_BITMAPS macro 5-3463	SYS_PORTS_PinDirectionSelect function 5-3481
SYS_MSG_OBJECT structure 5-3460	SYS_PORTS_PinModeSelect function 5-3476
SYS_MSG_QUEUE_STATUS enumeration 5-3461	SYS_PORTS_PinOpenDrainDisable function 5-3476
SYS_MSG_QueueStatus function 5-3457	SYS_PORTS_PinOpenDrainEnable function 5-3477
SYS_MSG_RECEIVE_CALLBACK type 5-3461	SYS_PORTS_PinPullUpDisable function 5-3477
SYS_MSG_RESULTS enumeration 5-3461	SYS_PORTS_PinPullUpEnable function 5-3478
SYS_MSG_Tasks function 5-3447	SYS_PORTS_PinRead function 5-3479
SYS_MSG_TypeCreate function 5-3453	SYS_PORTS_PinSet function 5-3479
SYS_MSG_TypeRemove function 5-3453	SYS_PORTS_PinToggle function 5-3480
SYS_MSG_VersionGet function 5-3447	SYS_PORTS_PinWrite function 5-3480
SYS_MSG_VersionStrGet function 5-3448	SYS_PORTS_PULLUP_PULLDOWN_STATUS enumeration 5-3497
SYS_MSGQ_ELEMENT structure 5-3462	SYS_PORTS_PullUpDisable function 5-3485
SYS_OBJ_HANDLE type 5-286	SYS_PORTS_PullUpEnable function 5-3486
SYS_OBJ_HANDLE_INVALID macro 5-286	SYS_PORTS_Read function 5-3486
SYS_OBJ_HANDLE_STATIC macro 5-286	SYS_PORTS_RemapInput function 5-3496
sys_ports.h 5-3498	SYS_PORTS_RemapOutput function 5-3496
SYS_PORTS_ChangeNotificationDisable function 5-3489	SYS_PORTS_Set function 5-3487
SYS_PORTS_ChangeNotificationEnable function 5-3489	SYS_PORTS_Toggle function 5-3487
SYS_PORTS_ChangeNotificationGlobalDisable function 5-3490	SYS_PORTS_Write function 5-3488
SYS_PORTS_ChangeNotificationGlobalEnable function 5-3490	SYS_STATUS enumeration 5-3340
SYS_PORTS_ChangeNotificationIdleModeDisable function 5-3491	SYS_Tasks function 5-3333
SYS_PORTS_ChangeNotificationIdleModeEnable function 5-3491	SYS_TASKS_PRIORITY enumeration 5-3341
SYS_PORTS_ChangeNotificationPullDownDisable function 5-3491	SYS_TASKS_PRIORITY_HIGH enumeration member 5-3341
SYS_PORTS_ChangeNotificationPullDownEnable function 5-3492	SYS_TASKS_PRIORITY_INVALID enumeration member 5-3341
SYS_PORTS_ChangeNotificationPullUpDisable function 5-3493	SYS_TASKS_PRIORITY_LOW enumeration member 5-3341
SYS_PORTS_ChangeNotificationPullUpEnable function 5-3493	SYS_TASKS_PRIORITY_MEDIUM enumeration member 5-3341
SYS_PORTS_ChangeNotificationStatus function 5-3494	sys_tmr.h 5-3528
SYS_PORTS_Clear function 5-3482	SYS_TMR_AlarmPeriodGet function 5-3525
SYS_PORTS_DirectionGet function 5-3483	SYS_TMR_CALLBACK type 5-3525
SYS_PORTS_DirectionSelect function 5-3483	SYS_TMR_CallbackPeriodic function 5-3522
SYS_PORTS_IOLockDisable function 5-3494	SYS_TMR_CallbackSingle function 5-3523
SYS_PORTS_IOLockEnable function 5-3495	SYS_TMR_CallbackStop function 5-3523
SYS_PORTS_IOLocksActive function 5-3495	sys_tmr_config.h 5-3529
SYS_PORTS_OpenDrainDisable function 5-3484	SYS_TMR_Deinitialize function 5-3518
SYS_PORTS_OpenDrainEnable function 5-3484	SYS_TMR_DELAY_STATUS enumeration 5-3526
SYS_PORTS_PIN_DIRECTION enumeration 5-3497	SYS_TMR_DelayMS function 5-3524
SYS_PORTS_PinClear function 5-3481	SYS_TMR_DelayStatusGet function 5-3524

SYS_TMR_ERROR_TOLERANCE macro 5-3516	Target Initialization 5-2765
SYS_TMR_HANDLE type 5-3526	TCP Changes 5-3548
SYS_TMR_HANDLE_INVALID macro 5-3527	TCP TCP/IP Stack Library 5-3880
SYS_TMR_INDEX_0 macro 5-3527	tcp.h 5-3910
SYS_TMR_INIT structure 5-3526	TCP/IP Demonstrations 3-130
SYS_TMR_Initialize function 5-3519	TCP/IP Discoverer 5-3545
SYS_TMR_INTERRUPT_MODE macro 5-3516	TCP/IP Library Overview 5-3536
SYS_TMR_MAX_PERIODIC_EVENTS macro 5-3517	TCP/IP Stack Library Help 5-3536
SYS_TMR_Reinitialize function 5-3520	TCP/IP Stack Porting Guide 5-3545
SYS_TMR_RUNNING macro 5-3527	TCP_ADJUST_FLAGS enumeration 5-3907
SYS_TMR_SINGLE_PERIODIC_EVENT macro 5-3517	TCP_AUTO_TRANSMIT_TIMEOUT_VAL macro 5-3885
SYS_TMR_Status function 5-3521	TCP_CLOSE_WAIT_TIMEOUT macro 5-3885
SYS_TMR_Tasks function 5-3521	tcp_config.h 5-3912
SYS_TMR_TickCountGet function 5-3525	TCP_DELAYED_ACK_TIMEOUT macro 5-3885
sys_wdt.h 5-3534	TCP_FIN_WAIT_2_TIMEOUT macro 5-3886
sys_wdt_config.h 5-3535	TCP_KEEP_ALIVE_TIMEOUT macro 5-3886
SYS_WDT_Disable function 5-3533	TCP_MAX_RETRIES macro 5-3886
SYS_WDT_Enable function 5-3533	TCP_MAX_SEG_SIZE_RX_LOCAL macro 5-3886
SYS_WDT_TimerClear function 5-3533	TCP_MAX_SEG_SIZE_RX_NON_LOCAL macro 5-3886
System Access 5-421	TCP_MAX_SEG_SIZE_TX macro 5-3887
System Bus Peripheral Library 5-2762	TCP_MAX_SOCKETS macro 5-3887
System Config 5-476	TCP_MAX_SYN_RETRIES macro 5-3887
System Configuration 5-428, 5-3390	TCP_MAX_UNACKED_KEEP_ALIVES macro 5-3887
System Configuration Functions 5-4173	TCP_MODULE_CONFIG structure 5-3907
System Initialization 5-195, 5-345, 5-370, 5-514, 5-608, 7-4229	TCP_OPTION_LINGER_DATA structure 5-3908
System Interaction 5-459, 5-3365, 5-3507, 5-4170	TCP_PORT type 5-3908
System Overview 3-63	TCP_SOCKET type 5-3908
System Service Library Help 5-3332	TCP_SOCKET_DEFAULT_RX_SIZE macro 5-3887
System Service Overview 5-3332	TCP_SOCKET_DEFAULT_TX_SIZE macro 5-3888
system.h 5-3347	TCP_SOCKET_INFO structure 5-3909
system_config.h 1-16	TCP_SOCKET_OPTION enumeration 5-3909
system_init.c 1-17	TCP_START_TIMEOUT_VAL macro 5-3888
system_int.c 1-19	TCP_TASK_TICK_RATE macro 5-3888
system_tasks.c 1-19	TCP_WINDOW_UPDATE_TIMEOUT_VAL macro 5-3888
SYSTEMConfigPerformance function 7-4240	tcpip_announce_config.h 5-3574
	TCPIP_ANNOUNCE_DeInit function 5-3572
	TCPIP_ANNOUNCE_Init function 5-3572
	tcpip_announce_manager.h 5-3574
	TCPIP_ANNOUNCE_MODULE_CONFIG structure 5-3573
T	
Table of Library Functions 5-1170, 5-1277	
TAN macro 5-1055	

TCPIP_ANNOUNCE_Send function 5-3573	TCPIP_DHCP_HandlerDeRegister function 5-3619
TCPIP_ANNOUNCE_Task function 5-3573	TCPIP_DHCP_HandlerRegister function 5-3619
TCPIP_ANNOUNCE_TaskPending function 5-3573	TCPIP_DHCP_IsBound function 5-3621
TCPIP_ARP_CacheEntriesNoGet function 5-3589	TCPIP_DHCP_IsEnabled function 5-3621
TCPIP_ARP_CacheThresholdSet function 5-3590	TCPIP_DHCP_IsServerDetected function 5-3622
TCPIP_ARP_CallbacksDeregister function 5-3581	TCPIP_DHCP_Renew function 5-3620
TCPIP_ARP_CallbacksRegister function 5-3581	TCPIP_DHCP_Request function 5-3620
TCPIP_ARP_Deinitialize function 5-3582	TCPIP_DHCP_RequestTimeoutSet function 5-3621
TCPIP_ARP_ENTRY_QUERY structure 5-3592	TCPIP_DHCPS_Disable function 5-3630
TCPIP_ARP_ENTRY_TYPE enumeration 5-3592	TCPIP_DHCPS_Enable function 5-3631
TCPIP_ARP_EntryGet function 5-3582	TCPIP_DHCPS_GetPoolEntries function 5-3631
TCPIP_ARP_EntryQuery function 5-3590	TCPIP_DHCPS_IsEnabled function 5-3633
TCPIP_ARP_EntryRemove function 5-3583	TCPIP_DHCPS_LeaseEntryGet function 5-3632
TCPIP_ARP_EntryRemoveAll function 5-3591	TCPIP_DHCPS_RemovePoolEntries function 5-3632
TCPIP_ARP_EntryRemoveNet function 5-3584	TCPIP_DNS_IsResolved function 5-3641
TCPIP_ARP_EntrySet function 5-3588	TCPIP_DNS_Resolve function 5-3641
TCPIP_ARP_EVENT_HANDLER type 5-3592	TCPIP_DNS_UsageBegin function 5-3642
TCPIP_ARP_EVENT_TYPE enumeration 5-3593	TCPIP_DNS_UsageEnd function 5-3642
TCPIP_ARP_HANDLE type 5-3593	TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE macro 5-3656
TCPIP_ARP_HandlerDeRegister function 5-3584	TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE macro 5-3657
TCPIP_ARP_HandlerRegister function 5-3585	TCPIP_FTP_MAX_CONNECTIONS macro 5-3657
TCPIP_ARP_Initialize function 5-3585	TCPIP_FTP_PASSWD_LEN macro 5-3657
TCPIP_ARP_IsResolved function 5-3586	TCPIP_FTP_PUT_ENABLED macro 5-3657
TCPIP_ARP_OPERATION_TYPE enumeration 5-3593	TCPIP_FTP_USER_NAME_LEN macro 5-3657
TCPIP_ARP_Probe function 5-3586	TCPIP_HTTP_ActiveConnectionCountGet function 5-3666
TCPIP_ARP_Process function 5-3589	TCPIP_HTTP_ArgGet function 5-3667
TCPIP_ARP_Resolve function 5-3587	TCPIP_HTTP_CurrentConnectionByteCountDec function 5-3668
TCPIP_ARP_RESULT enumeration 5-3594	TCPIP_HTTP_CurrentConnectionByteCountGet function 5-3668
TCPIP_ARP_Task function 5-3588	TCPIP_HTTP_CurrentConnectionByteCountSet function 5-3669
TCPIP_ARP_TaskIsPending function 5-3588	TCPIP_HTTP_CurrentConnectionCallbackPosGet function 5-3669
TCPIP_DDNS_LastIPGet function 5-3649	TCPIP_HTTP_CurrentConnectionCallbackPosSet function 5-3670
TCPIP_DDNS_LastStatusGet function 5-3649	TCPIP_HTTP_CurrentConnectionDataBufferGet function 5-3670
TCPIP_DDNS_ServiceSet function 5-3650	TCPIP_HTTP_CurrentConnectionFileGet function 5-3671
TCPIP_DDNS_UpdateForce function 5-3650	TCPIP_HTTP_CurrentConnectionHasArgsSet function 5-3671
TCPIP_DHCP_Disable function 5-3618	TCPIP_HTTP_CurrentConnectionIsAuthorizedGet function 5-3672
TCPIP_DHCP_Enable function 5-3619	TCPIP_HTTP_CurrentConnectionIsAuthorizedSet function 5-3672
TCPIP_DHCP_EVENT_HANDLER type 5-3623	
TCPIP_DHCP_EVENT_TYPE enumeration 5-3623	
TCPIP_DHCP_HANDLE type 5-3623	

TCPIP_HTTP_CurrentConnectionPostSmGet function 5-3673	TCPIP_IPV6_PayloadSet function 5-3731
TCPIP_HTTP_CurrentConnectionPostSmSet function 5-3673	TCPIP_IPV6_Put function 5-3732
TCPIP_HTTP_CurrentConnectionSocketGet function 5-3674	TCPIP_IPV6_PutArray macro 5-3738
TCPIP_HTTP_CurrentConnectionStatusSet function 5-3675	TCPIP_IPV6_SourceAddressGet function 5-3732
TCPIP_HTTP_CurrentConnectionUserDataGet function 5-3675	TCPIP_IPV6_SourceAddressSet function 5-3733
TCPIP_HTTP_CurrentConnectionUserDataSet function 5-3676	TCPIP_IPV6_TxIsPutReady function 5-3733
TCPIP_HTTP_FileInclude function 5-3676	TCPIP_IPV6_TxPacketAllocate function 5-3734
TCPIP_HTTP_PostNameRead function 5-3677	TCPIP_IPV6_UnicastAddressAdd function 5-3734
TCPIP_HTTP_PostValueRead function 5-3678	TCPIP_IPV6_UniqueLocalUnicastAddressAdd function 5-3735
TCPIP_HTTP_URLDecode function 5-3679	TCPIP_LOCAL_MASK_TYPE enumeration 5-3793
TCPIP_ICMP_CallbackDeregister function 5-3693	tcPIP_mac.h 5-3770
TCPIP_ICMP_CallbackRegister function 5-3693	TCPIP_MAC_ACTION enumeration 5-3757
TCPIP_ICMP_EchoRequestSend function 5-3694	TCPIP_MAC_ADDR structure 5-3758
TCPIP_ICMPV6_CallbackDeregister function 5-3701	TCPIP_MAC_Close function 5-3753
TCPIP_ICMPV6_CallbackRegister function 5-3702	tcPIP_mac_config.h 5-3770
TCPIP_ICMPV6_Close function 5-3702	TCPIP_MAC_DATA_SEGMENT type 5-3758
TCPIP_ICMPV6_Flush function 5-3702	TCPIP_MAC_Deinitialize function 5-3753
TCPIP_ICMPV6_HeaderEchoRequestPut function 5-3703	TCPIP_MAC_ETHERNET_HEADER structure 5-3758
TCPIP_ICMPV6_PutHeaderEchoReply macro 5-3704	TCPIP_MAC_EVENT enumeration 5-3758
TCPIP_IF_PIC32INT macro 5-262	TCPIP_MAC_EventAcknowledge function 5-3753
TCPIP_IPV4_PacketFormatTx function 5-3712	TCPIP_MAC_EventF type 5-3760
TCPIP_IPV4_PacketGetDestAddress function 5-3713	TCPIP_MAC_EventMaskSet function 5-3753
TCPIP_IPV4_PacketGetSourceAddress function 5-3713	TCPIP_MAC_EventPendingGet function 5-3754
TCPIP_IPV4_PacketTransmit function 5-3714	TCPIP_MAC_HANDLE type 5-3761
TCPIP_IPV4_SelectSourceInterface function 5-3714	TCPIP_MAC_HEAP_CallocF type 5-3761
TCPIP_IPV6_AddressUnicastRemove function 5-3725	TCPIP_MAC_HEAP_FreeF type 5-3761
TCPIP_IPV6_ArrayGet function 5-3725	TCPIP_MAC_HEAP_HANDLE type 5-3762
TCPIP_IPV6_ArrayPutHelper function 5-3726	TCPIP_MAC_HEAP_MallocF type 5-3762
TCPIP_IPV6_DASSourceAddressSelect function 5-3726	TCPIP_MAC_Initialize function 5-3754
TCPIP_IPV6_DestAddressGet function 5-3727	TCPIP_MAC_LinkCheck function 5-3754
TCPIP_IPV6_DestAddressSet function 5-3727	TCPIP_MAC_MODULE_CTRL structure 5-3762
TCPIP_IPV6_Flush function 5-3728	TCPIP_MAC_Open function 5-3754
TCPIP_IPV6_Get function 5-3728	TCPIP_MAC_PACKET type 5-3763
TCPIP_IPV6_HandlerDeregister function 5-3728	TCPIP_MAC_PACKET_ACK_FUNC type 5-3763
TCPIP_IPV6_HandlerRegister function 5-3729	TCPIP_MAC_PACKET_FLAGS enumeration 5-3763
TCPIP_IPV6_InterfacesReady function 5-3729	TCPIP_MAC_PACKET_RX_STAT structure 5-3764
TCPIP_IPV6_MulticastListenerAdd function 5-3730	TCPIP_MAC_PacketRx function 5-3754
TCPIP_IPV6_MulticastListenerRemove function 5-3730	TCPIP_MAC_PacketTx function 5-3755
TCPIP_IPV6_PacketFree function 5-3731	TCPIP_MAC_PKT_ACK_RES enumeration 5-3765

TCPIP_MAC_PKT_AckF type 5-3766	TCPIP_SNMP_PacketProcStubPtrsGet function 5-3837
TCPIP_MAC_PKT_AllocF type 5-3766	TCPIP_SNMP_ProcessBufferDataGet function 5-3838
TCPIP_MAC_PKT_FreeF type 5-3766	TCPIP_SNMP_ReadCommunityGet function 5-3838
TCPIP_MAC_POWER_MODE enumeration 5-3766	TCPIP_SNMP_SM enumeration 5-3852
TCPIP_MAC_Process function 5-3756	TCPIP_SNMP_TrapTimeGet function 5-3839
TCPIP_MAC_PROCESS_FLAGS enumeration 5-3767	TCPIP_SNMP_WriteCommunityGet function 5-3839
TCPIP_MAC_Reinitialize function 5-3757	TCPIP_SNMPv3_CmprTrapSecNameAndSecLvIWithUSMDB function 5-3845
TCPIP_MAC_RES enumeration 5-3767	TCPIP_SNMPv3_Notify function 5-3839
TCPIP_MAC_RxFilterHashTableEntrySet function 5-3757	TCPIP_SNMPV3_PacketProcStubPtrsGet function 5-3840
TCPIP_MAC_SEGMENT_FLAGS enumeration 5-3768	TCPIP_SNTP_ConnectionInitiate function 5-3863
tcip_manager.h 5-3793	TCPIP_SNTP_ConnectionParamSet function 5-3863
TCPIP_MDNS_ServiceDeregister function 5-3958	TCPIP_SNTP_LastErrorGet function 5-3864
TCPIP_MDNS_ServiceRegister function 5-3959	TCPIP_SNTP_TimeStampGet function 5-3864
TCPIP_MDNS_ServiceUpdate function 5-3960	TCPIP_SNTP_UTCSecondsGet function 5-3865
TCPIP_MODULE_MAC_97J60_CONFIG structure 5-3768	TCPIP_STACK_Deinitialize function 5-3774
TCPIP_MODULE_MAC_ENCJ60_CONFIG structure 5-3768	TCPIP_STACK_IndexToNet function 5-3777
TCPIP_MODULE_MAC_ENCJ600_CONFIG structure 5-3769	TCPIP_STACK_Initialize function 5-3774
TCPIP_MODULE_MAC_MRF24W_CONFIG structure 5-3769	TCPIP_STACK_ModuleGetConfig function 5-3783
TCPIP_MODULE_MAC_PIC32INT_CONFIG structure 5-3769	TCPIP_STACK_NetAddress function 5-3784
TCPIP_NDP_NborReachConfirm function 5-3804	TCPIP_STACK_NetAddressBcast function 5-3784
TCPIP_NET_HANDLE type 5-3793	TCPIP_STACK_NetAddressDnsPrimary function 5-3785
tcip_reboot_config.h 5-3808	TCPIP_STACK_NetAddressDnsPrimarySet function 5-3785
TCPIP_SMTP_ArrayPut function 5-3815	TCPIP_STACK_NetAddressDnsSecond function 5-3786
TCPIP_SMTP_Flush function 5-3816	TCPIP_STACK_NetAddressDnsSecondSet function 5-3786
TCPIP_SMTP_IsBusy function 5-3816	TCPIP_STACK_NetAddressGateway function 5-3787
TCPIP_SMTP_IsPutReady function 5-3816	TCPIP_STACK_NetAddressGatewaySet function 5-3787
TCPIP_SMTP_MailSend function 5-3817	TCPIP_STACK_NetAddressMac function 5-3788
TCPIP_SMTP_Put function 5-3817	TCPIP_STACK_NetAddressMacSet function 5-3788
TCPIP_SMTP_PutIsDone function 5-3818	TCPIP_STACK_NetAddressSet function 5-3789
TCPIP_SMTP_StringPut function 5-3818	TCPIP_STACK_NetBIOSName function 5-3777
TCPIP_SMTP_UsageBegin function 5-3818	TCPIP_STACK_NetBiosNameSet function 5-3778
TCPIP_SMTP_UsageEnd function 5-3819	TCPIP_STACK_NetConfigGet function 5-3790
TCPIP_SNMP_ClientGetNet function 5-3834	TCPIP_STACK_NetConfigSet function 5-3791
TCPIP_SNMP_DataCopyToProcessBuffer function 5-3834	TCPIP_STACK_NetDefaultGet function 5-3778
TCPIP_SNMP_DCPT structure 5-3851	TCPIP_STACK_NetDefaultSet function 5-3779
TCPIP_SNMP_EventNotifyGet function 5-3835	TCPIP_STACK_NetDown function 5-3781
TCPIP_SNMP_Notify function 5-3835	TCPIP_STACK_NetHandleGet function 5-3779
TCPIP_SNMP_NotifyIsReady function 5-3836	TCPIP_STACK_NetIndexGet function 5-3780
TCPIP_SNMP_NotifyPrepare function 5-3837	TCPIP_STACK_NetIPv6AddressGet function 5-3790

TCPIP_STACK_NetIsLinked function 5-3782	TCPIP_TCP_SocketInfoGet function 5-3896
TCPIP_STACK_NetIsUp function 5-3782	TCPIP_TCP_SocketIsSecuredBySSL function 5-3876
TCPIP_STACK_NetMACId function 5-3792	TCPIP_TCP_SocketNetGet function 5-3896
TCPIP_STACK_NetMask function 5-3780	TCPIP_TCP_SocketNetSet function 5-3898
TCPIP_STACK_NetNameGet function 5-3780	TCPIP_TCP_StringPut function 5-3900
TCPIP_STACK_NetUp function 5-3783	TCPIP_TCP_WasReset function 5-3896
TCPIP_STACK_NumberOfNetworksGet function 5-3781	TCPIP_TCPSSL_ClientBegin function 5-3876
TCPIP_STACK_SetLocalMasks function 5-3775	TCPIP_TCPSSL_ClientStart function 5-3877
TCPIP_STACK_SetLocalMasksType function 5-3776	TCPIP_TCPSSL_HandshakeClear function 5-3877
TCPIP_STACK_Task function 5-3777	TCPIP_TCPSSL_ListenerAdd function 5-3878
TCPIP_STACK_USE_BASE64_DECODE macro 5-3665	TCPIP_TCPSSL_ServerStart function 5-3878
TCPIP_TCP_Abort function 5-3897	TCPIP_TCPSSL_StillHandshaking function 5-3879
TCPIP_TCP_ArrayFind function 5-3901	TCPIP_TFTP_DataByteGet function 5-3922
TCPIP_TCP_ArrayGet function 5-3902	TCPIP_TFTP_DataBytePut function 5-3923
TCPIP_TCP_ArrayPeek function 5-3902	TCPIP_TFTP_DataIsGetReady function 5-3923
TCPIP_TCP_ArrayPut function 5-3898	TCPIP_TFTP_FileClose function 5-3923
TCPIP_TCP_Bind function 5-3890	TCPIP_TFTP_FileIsClosed function 5-3923
TCPIP_TCP_ClientOpen function 5-3891	TCPIP_TFTP_FileIsOpen function 5-3923
TCPIP_TCP_Close function 5-3891	TCPIP_TFTP_FragRAMFileUploadToHost function 5-3923
TCPIP_TCP_Connect function 5-3892	TCPIP_TFTP_IsOpen function 5-3924
TCPIP_TCP_Discard function 5-3903	TCPIP_TFTP_IsReadyForPut function 5-3924
TCPIP_TCP_Disconnect function 5-3892	TCPIP_TFTP_Open function 5-3924
TCPIP_TCP_FifoRxFreeGet function 5-3906	TCPIP_TFTP_RAMFileUploadToHost function 5-3924
TCPIP_TCP_FifoRxFullGet macro 5-3906	TCPIP_TFTP_UploadStatusGet function 5-3924
TCPIP_TCP_FifoSizeAdjust function 5-3903	TCPIP_UDP_ArrayGet function 5-3944
TCPIP_TCP_FifoTxFreeGet macro 5-3907	TCPIP_UDP_ArrayPut function 5-3947
TCPIP_TCP_FifoTxFullGet function 5-3899	TCPIP_UDP_BcastIPv4AddressSet function 5-3937
TCPIP_TCP_Find function 5-3904	TCPIP_UDP_Bind function 5-3937
TCPIP_TCP_Flush function 5-3899	TCPIP_UDP_ClientOpen function 5-3938
TCPIP_TCP_Get function 5-3905	TCPIP_UDP_Close function 5-3938
TCPIP_TCP_GetIsReady function 5-3905	TCPIP_UDP_DestinationIPAddressSet function 5-3939
TCPIP_TCP_IsConnected function 5-3893	TCPIP_UDP_Discard function 5-3948
TCPIP_TCP_OptionsGet function 5-3893	TCPIP_UDP_Flush function 5-3945
TCPIP_TCP_OptionsSet function 5-3894	TCPIP_UDP_Get function 5-3949
TCPIP_TCP_Peek function 5-3905	TCPIP_UDP_GetIsReady function 5-3949
TCPIP_TCP_Put function 5-3899	TCPIP_UDP_IsConnected function 5-3939
TCPIP_TCP_PutIsReady function 5-3900	TCPIP_UDP_IsOpened macro 5-3950
TCPIP_TCP_RemoteBind function 5-3894	TCPIP_UDP_OptionsGet function 5-3940
TCPIP_TCP_ServerOpen function 5-3895	TCPIP_UDP_OptionsSet function 5-3940

-
- TCPIP_UDP_Put function 5-3945
 - TCPIP_UDP_RemoteBind function 5-3941
 - TCPIP_UDP_RxOffsetSet function 5-3949
 - TCPIP_UDP_ServerOpen function 5-3942
 - TCPIP_UDP_SocketInfoGet function 5-3942
 - TCPIP_UDP_SocketNetGet function 5-3942
 - TCPIP_UDP_SocketNetSet function 5-3943
 - TCPIP_UDP_SourceIPAddressSet function 5-3943
 - TCPIP_UDP_StringPut function 5-3946
 - TCPIP_UDP_TxCountGet function 5-3946
 - TCPIP_UDP_TxOffsetSet function 5-3944
 - TCPIP_UDP_TxPutIsReady function 5-3947
 - TCPIP_ZCLL_Disable function 5-3960
 - TCPIP_ZCLL_Enable function 5-3961
 - TCPIP_ZCLL_IsEnabled function 5-3961
 - TE_ROUNDEDBUTTON_RADIUS macro 5-1055
 - Telnet TCP/IP Stack Library 5-3913
 - telnet.h 5-3917
 - telnet_config.h 5-3917
 - TELNET_MODULE_CONFIG structure 5-3916
 - TELNET_PASSWORD macro 5-3915
 - TELNET_PORT macro 5-3915
 - TELNET_USERNAME macro 5-3915
 - TELNETS_PORT macro 5-3916
 - TFTP TCP/IP Stack Library 5-3917
 - TFTP_ACCESS_ERROR type 5-3928
 - TFTP_ARP_TIMEOUT_VAL macro 5-3920
 - TFTP_BLOCK_SIZE macro 5-3920
 - TFTP_BLOCK_SIZE_MSB macro 5-3920
 - TFTP_CHUNK_DESCRIPTOR structure 5-3928
 - TFTP_CLIENT_PORT macro 5-3921
 - TFTP_FILE_MODE type 5-3928
 - TFTP_GET_TIMEOUT_VAL macro 5-3921
 - TFTP_MAX_RETRIES macro 5-3921
 - TFTP_RESULT type 5-3928
 - TFTP_SERVER_PORT macro 5-3921
 - TFTP_UPLOAD_COMPLETE macro 5-3925
 - TFTP_UPLOAD_CONNECT macro 5-3925
 - TFTP_UPLOAD_CONNECT_TIMEOUT macro 5-3925
 - TFTP_UPLOAD_GET_DNS macro 5-3925
 - TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT macro 5-3925
 - TFTP_UPLOAD_RESOLVE_HOST macro 5-3926
 - TFTP_UPLOAD_SEND_DATA macro 5-3926
 - TFTP_UPLOAD_SEND_FILENAME macro 5-3926
 - TFTP_UPLOAD_SERVER_ERROR macro 5-3926
 - TFTP_UPLOAD_WAIT_FOR_CLOSURE macro 5-3926
 - tftpc.h 5-3929
 - tftpc_config.h 5-3930
 - TFTPClose macro 5-3926
 - TFTPGetError macro 5-3927
 - TFTPIsFileOpenReady macro 5-3927
 - TFTPOpenFile function 5-3925
 - The Application File(s) 1-13
 - The Main File 1-12
 - Third-Party Library Help 6-4225
 - Third-Party Library Overview 6-4226
 - Thread Operation 5-1316
 - Timer Driver Library 5-457
 - Timer Peripheral Library 5-3011
 - Timer System Service Library 5-3500
 - tlPV6_SNMP_TRAP_INFO structure 5-3848
 - TMR_ASSIGNMENT enumeration 5-3060
 - TMR_CLOCK_SOURCE enumeration 5-3060
 - TMR_CLOCK_SOURCE_EDGE enumeration 5-3060
 - TMR_GATE_POLARITY enumeration 5-3061
 - TMR_GATE_SOURCE enumeration 5-3061
 - TMR_MODULE_ID enumeration 5-3062
 - TMR_POSTSCALE enumeration 5-3062
 - TMR_PRESCALE enumeration 5-3063
 - TPL_MATCH_VID_PID macro 5-4196
 - Transfer Abort 5-573
 - Transfer Operation 5-372
 - Transfer Requests 5-565
 - Transfer Status 5-572
 - Transfer/Abort (Asynchronous) Trigger Management 5-1737
 - Transfer/Abort (Synchronous) 5-1737
 - Transferring Data 5-3984
 - Transitions 5-1122
-

Transmit 5-1918
 Transmitting a CAN Message 5-1561
 TRAP_COMMUNITY_MAX_LEN macro 5-3831
 TRAP_TABLE_SIZE macro 5-3831
 tSNMP_TRAP_INFO structure 5-3850
 Tutorial 2-49

U

uC_OS_III_basic_demo 3-115
 UDP Changes 5-3550
 UDP TCP/IP Stack Library 5-3930
 udp.h 5-3953
 udp_config.h 5-3954
 UDP_LOCAL_PORT_END_NUMBER macro 5-3933
 UDP_LOCAL_PORT_START_NUMBER macro 5-3934
 UDP_MAX_SOCKETS macro 5-3934
 UDP_MODULE_CONFIG structure 5-3950
 UDP_PORT type 5-3951
 UDP_SOCKET type 5-3951
 UDP_SOCKET_BCAST_TYPE enumeration 5-3951
 UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT macro 5-3934
 UDP_SOCKET_DEFAULT_TX_SIZE macro 5-3934
 UDP_SOCKET_INFO structure 5-3951
 UDP_SOCKET_OPTION enumeration 5-3952
 UDP_SOCKET_POOL_BUFFER_SIZE macro 5-3934
 UDP_SOCKET_POOL_BUFFERS macro 5-3935
 UDP_USE_RX_CHECKSUM macro 5-3935
 UDP_USE_TX_CHECKSUM macro 5-3935
 UDP_IsPutReady function 5-3948
 uint16_gfx_image_prog type 5-940
 uint16_prog type 5-940
 uint16_prog_pack type 5-940
 uint32_gfx_image_prog type 5-940
 uint32_prog type 5-940
 uint32_prog_pack type 5-940
 uint8_gfx_image_prog type 5-941
 uint8_prog type 5-941
 uint8_prog_pack type 5-941
 Upgrading from the MLA Graphics Library to the MPLAB
 Harmony Graphics Library 5-681
 Upgrading from the V5 TCP/IP Stack to the MPLAB Harmony
 TCP/IP Stack 5-3546
 Uploading Pre-built MPFS2 Images 5-3543
 USART Driver Demonstration 3-142
 USART Driver Library 5-510
 USART Peripheral Library 5-3085
 USART_BAUD_RATE_MODE enumeration 5-3140
 USART_HANDSHAKE_MODE enumeration 5-3140
 USART_LINECONTROL_MODE enumeration 5-3140
 USART_MODULE_ID enumeration 5-3141
 USART_OPERATION_MODE enumeration 5-3142
 USART_RECEIVE_INTR_MODE enumeration 5-3142
 USART_RECEIVE_MODES enumeration 5-3142
 USART_SYNC_CLOCK_SOURCE enumeration 5-3143
 USART_SYNC_MODES enumeration 5-3143
 USART_TRANSMIT_INTR_MODE enumeration 5-3144
 USB Audio Device Library 5-3977
 USB Buffers and the Buffer Descriptor Table (BDT) 5-3175
 USB CDC Host Library 5-4199
 USB Demonstrations 3-149
 USB Device CDC Library 5-4060
 USB Device Layer Library 5-4004
 USB Device Library - Application Interaction 5-3968
 USB Device Library - Getting Started 5-3963
 USB Device Library Architecture 5-3966
 USB Device Stack Migration Guide 5-3973
 USB Driver Library 5-552
 USB Generic Device Library 5-4100
 USB HID Device Library 5-4130
 USB Host Layer Library 5-4179
 USB Library Help 5-3963
 USB MSD Device Library 5-4168
 USB Peripheral Library 5-3167
 USB Setup Example 5-3179
 USB_BUFFER_DATA01 enumeration 5-3278
 USB_BUFFER_DIRECTION enumeration 5-3278
 USB_BUFFER_PING_PONG enumeration 5-3279
 USB_BUFFER_SCHEDULE_DATA01 enumeration 5-3279
 usb_device.h 5-4057

USB_DEVICE_Attach function 5-4033	USB_DEVICE_CDC_ControlSend function 5-4078
usb_device_audio.h 5-4002	USB_DEVICE_CDC_ControlStatus function 5-4080
USB_DEVICE_AUDIO_CONTROL_TRANSFER_HANDLE type 5-3995	USB_DEVICE_CDC_EVENT enumeration 5-4089
USB_DEVICE_AUDIO_ControlReceive function 5-3987	USB_DEVICE_CDC_EVENT_DATA_GET_LINE_CODING type 5-4092
USB_DEVICE_AUDIO_ControlSend function 5-3988	USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE structure 5-4092
USB_DEVICE_AUDIO_ControlStatus function 5-3990	USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK structure 5-4092
USB_DEVICE_AUDIO_EP_INSTANCE structure 5-3995	USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE structure 5-4093
USB_DEVICE_AUDIO_EVENT enumeration 5-3996	USB_DEVICE_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE type 5-4093
USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE structure 5-3997	USB_DEVICE_CDC_EVENT_DATA_SET_LINE_CODING type 5-4093
USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE structure 5-3997	USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE structure 5-4094
USB_DEVICE_AUDIO_EVENT_HANDLER type 5-3997	USB_DEVICE_CDC_EVENT_HANDLER type 5-4094
USB_DEVICE_AUDIO_EVENT_RESPONSE type 5-3998	USB_DEVICE_CDC_EVENT_RESPONSE type 5-4095
USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE macro 5-3994	USB_DEVICE_CDC_EVENT_RESPONSE_NONE macro 5-4097
USB_DEVICE_AUDIO_EventHandlerSet function 5-3991	USB_DEVICE_CDC_EventHandlerSet function 5-4082
USB_DEVICE_AUDIO_INDEX type 5-3998	USB_DEVICE_CDC_INDEX type 5-4095
USB_DEVICE_AUDIO_INSTANCE structure 5-3998	USB_DEVICE_CDC_INIT structure 5-4095
USB_DEVICE_AUDIO_INSTANCE_STATE enumeration 5-3999	USB_DEVICE_CDC_Read function 5-4084
USB_DEVICE_AUDIO_INTERFACE_COLLECTION structure 5-3999	USB_DEVICE_CDC_RESULT enumeration 5-4095
USB_DEVICE_AUDIO_INTERFACE_INFO structure 5-3999	USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA type 5-4096
USB_DEVICE_AUDIO_INTERFACE_TYPE enumeration 5-4000	USB_DEVICE_CDC_SerialStateNotificationSend function 5-4085
USB_DEVICE_AUDIO_IRP_OBJECT structure 5-4000	USB_DEVICE_CDC_TRANSFER_FLAGS enumeration 5-4096
USB_DEVICE_AUDIO_Read function 5-3992	USB_DEVICE_CDC_TRANSFER_HANDLE type 5-4097
USB_DEVICE_AUDIO_RESULT enumeration 5-4000	USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID macro 5-4097
USB_DEVICE_AUDIO_STREAMING_INTERFACE structure 5-4001	USB_DEVICE_CDC_Write function 5-4086
USB_DEVICE_AUDIO_STREAMING_INTERFACE_ALTERNATE_SETTING structure 5-4001	USB_DEVICE_ClientStatus function 5-4036
USB_DEVICE_AUDIO_TRANSFER_HANDLE type 5-4002	USB_DEVICE_Close function 5-4026
USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID macro 5-3995	USB_DEVICE_CONFIG_DESCS_PTR type 5-4049
USB_DEVICE_AUDIO_Write function 5-3993	usb_device_config_template.h 5-4059
USB_DEVICE_CALLBACK type 5-4048	USB_DEVICE_CONTROL_STATUS enumeration 5-4049
usb_device_cdc.h 5-4098	USB_DEVICE_CONTROL_TRANSFER_CALLBACK type 5-4049
USB_DEVICE_CDC_CONTROL_TRANSFER_HANDLE type 5-4089	
USB_DEVICE_CDC_ControlReceive function 5-4076	

USB_DEVICE_CONTROL_TRANSFER_EVENT enumeration 5-4050	structure 5-4126
USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA union 5-4050	USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_WRITE_COMPLETE structure 5-4127
USB_DEVICE_CONTROL_TRANSFER_HANDLE type 5-4050	USB_DEVICE_GENERIC_EVENT_HANDLER type 5-4127
USB_DEVICE_CONTROL_TRANSFER_RESULT enumeration 5-4051	USB_DEVICE_GENERIC_EVENT_RESPONSE type 5-4127
USB_DEVICE_ControlEventCallBackSet function 5-4028	USB_DEVICE_GENERIC_EventHandlerSet function 5-4124
USB_DEVICE_ControlReceive function 5-4032	USB_DEVICE_GENERIC_INDEX type 5-4128
USB_DEVICE_ControlSend function 5-4033	USB_DEVICE_GENERIC_RESULT enumeration 5-4128
USB_DEVICE_ControlStatus function 5-4040	USB_DEVICE_GENERIC_TRANSFER_FLAG enumeration 5-4128
USB_DEVICE_Deinitialize function 5-4026	USB_DEVICE_GENERIC_TRANSFER_HANDLE type 5-4129
USB_DEVICE_Detach function 5-4034	USB_DEVICE_GetConfigurationValue function 5-4039
USB_DEVICE_ENDPOINT_TABLE_SIZE macro 5-4057	USB_DEVICE_GetDeviceSpeed function 5-4037
USB_DEVICE_EndpointDisable function 5-4030	USB_DEVICE_GetDeviceState function 5-4038
USB_DEVICE_EndpointEnable function 5-4040	USB_DEVICE_HANDLE type 5-4053
USB_DEVICE_EndpointIsEnabled function 5-4042	USB_DEVICE_HANDLE_INVALID macro 5-4057
USB_DEVICE_EndpointIsStalled function 5-4042	usb_device_hid.h 5-4166
USB_DEVICE_EndpointStall function 5-4043	usb_device_hid_config_template.h 5-4168
USB_DEVICE_EndpointStallClear function 5-4044	USB_DEVICE_HID_CONTROL_STATUS enumeration 5-4152
USB_DEVICE_EVENT enumeration 5-4051	USB_DEVICE_HID_CONTROL_TRANSFER_HANDLE type 5-4153
USB_DEVICE_EVENT_DATA union 5-4052	USB_DEVICE_HID_ControlReceive function 5-4142
USB_DEVICE_EVENT_DATA_CONFIGURED structure 5-4052	USB_DEVICE_HID_ControlSend function 5-4144
USB_DEVICE_EventCallBackSet function 5-4031	USB_DEVICE_HID_ControlStatus function 5-4146
USB_DEVICE_FUNC_REGISTRATION_TABLE structure 5-4052	USB_DEVICE_HID_EVENT enumeration 5-4153
USB_DEVICE_FUNCTION_DRIVER structure 5-4053	USB_DEVICE_HID_EVENT_DATA_GET_IDLE structure 5-4160
usb_device_generic.h 5-4129	USB_DEVICE_HID_EVENT_DATA_GET_PROTOCOL structure 5-4160
USB_DEVICE_GENERIC_EndpointIsEnabled function 5-4121	USB_DEVICE_HID_EVENT_DATA_GET_REPORT structure 5-4160
USB_DEVICE_GENERIC_EndpointRead function 5-4122	USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED structure 5-4161
USB_DEVICE_GENERIC_EndpointStall function 5-4122	USB_DEVICE_HID_EVENT_DATA_REPORT_SENT structure 5-4161
USB_DEVICE_GENERIC_EndpointStallClear function 5-4123	USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR structure 5-4162
USB_DEVICE_GENERIC_EndpointStallStatusGet function 5-4123	USB_DEVICE_HID_EVENT_DATA_SET_IDLE structure 5-4162
USB_DEVICE_GENERIC_EndpointWrite function 5-4124	USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL structure 5-4162
USB_DEVICE_GENERIC_EVENT enumeration 5-4125	USB_DEVICE_HID_EVENT_DATA_SET_REPORT structure 5-4163
USB_DEVICE_GENERIC_EVENT_DATA union 5-4125	USB_DEVICE_HID_EVENT_HANDLER type 5-4163
USB_DEVICE_GENERIC_EVENT_DATA_CONTROL_TRANSFER_SETUP_REQUEST structure 5-4126	
USB_DEVICE_GENERIC_EVENT_DATA_ENDPOINT_READ_COMPLETE	

USB_DEVICE_HID_EVENT_RESPONSE type 5-4164	USB_DEVICE_Open function 5-4032
USB_DEVICE_HID_EVENT_RESPONSE_NONE macro 5-4165	USB_DEVICE_POWER_STATE enumeration 5-4054
USB_DEVICE_HID_EventHandlerSet function 5-4148	USB_DEVICE_PowerStateSet function 5-4039
USB_DEVICE_HID_INDEX type 5-4164	USB_DEVICE_Reinitialize function 5-4028
USB_DEVICE_HID_INIT structure 5-4164	USB_DEVICE_REMOTE_WAKEUP_STATUS enumeration 5-4055
USB_DEVICE_HID_INSTANCES_NUMBER macro 5-4166	USB_DEVICE_RemoteWakeupsIsEnabled function 5-4039
USB_DEVICE_HID_QUEUE_DEPTH_COMINED macro 5-4166	USB_DEVICE_ResumeStart function 5-4036
USB_DEVICE_HID_ReportReceive function 5-4149	USB_DEVICE_ResumeStop function 5-4037
USB_DEVICE_HID_ReportSend function 5-4151	USB_DEVICE_STATE enumeration 5-4055
USB_DEVICE_HID_RESULT enumeration 5-4165	USB_DEVICE_Status function 5-4035
USB_DEVICE_HID_TRANSFER_HANDLE type 5-4165	USB_DEVICE_STRING_DESCS_PTR type 5-4056
USB_DEVICE_HID_TRANSFER_HANDLE_INVALID macro 5-4166	USB_DEVICE_Tasks function 5-4031
USB_DEVICE_INDEX_0 macro 5-4047	USB_EP_TXRX enumeration 5-3279
USB_DEVICE_INDEX_1 macro 5-4047	usb_host.h 5-4198
USB_DEVICE_INDEX_2 macro 5-4048	USB_HOST_0 macro 5-4197
USB_DEVICE_INDEX_3 macro 5-4048	USB_HOST_1 macro 5-4197
USB_DEVICE_INDEX_4 macro 5-4048	usb_host_cdc.h 5-4223
USB_DEVICE_INDEX_5 macro 5-4048	USB_HOST_CDC_BreakSend function 5-4201
USB_DEVICE_INDEX_COUNT macro 5-4048	USB_HOST_CDC_CONTROL_LINE_STATE type 5-4223
USB_DEVICE_INFORMATION structure 5-4194	USB_HOST_CDC_ControlLineStateSet function 5-4202
USB_DEVICE_INIT structure 5-4054	USB_HOST_CDC_EVENT enumeration 5-4210
USB_DEVICE_Initialize function 5-4027	USB_HOST_CDC_EVENT_DATA_GET_LINE_CODING_COMPLETE structure 5-4216
USB_DEVICE_IRPCancelAll function 5-4044	USB_HOST_CDC_EVENT_DATA_READ_COMPLETE structure 5-4216
USB_DEVICE_IRPSubmit function 5-4045	USB_HOST_CDC_EVENT_DATA_SEND_BREAK_COMPLETE structure 5-4217
USB_DEVICE_MAX_CLIENTS macro 5-4023	USB_HOST_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_RECEIVED structure 5-4218
USB_DEVICE_MAX_FUNCTION_DRIVER macro 5-4023	USB_HOST_CDC_EVENT_DATA_SET_CONTROL_LINE_STATE_COMPLETE structure 5-4218
USB_DEVICE_MAX_INSTANCES macro 5-4023	USB_HOST_CDC_EVENT_DATA_SET_LINE_CODING_COMPLETE structure 5-4219
usb_device_msdc.h 5-4178	USB_HOST_CDC_EVENT_DATA_WRITE_COMPLETE structure 5-4220
usb_device_msdc_config_template.h 5-4178	USB_HOST_CDC_EVENT_HANDLER type 5-4220
USB_DEVICE_MSD_INIT structure 5-4176	USB_HOST_CDC_EVENT_RESPONSE type 5-4221
USB_DEVICE_MSD_INQUIRY_RESPONSE structure 5-4177	USB_HOST_CDC_EVENT_RESPONSE_NONE macro 5-4222
USB_DEVICE_MSD_INTF macro 5-4175	
USB_DEVICE_MSD_INTF_SUBCLASS macro 5-4175	
USB_DEVICE_MSD_MAX_INSTANCES macro 5-4174	
USB_DEVICE_MSD_MAX_LUN macro 5-4174	
USB_DEVICE_MSD_MAX_SECTOR_SIZE macro 5-4174	
USB_DEVICE_MSD_MEDIA_FUNCTIONS structure 5-4175	
USB_DEVICE_MSD_MEDIA_INIT_DATA structure 5-4177	
USB_DEVICE_MSD_PROTOCOL macro 5-4175	

USB_HOST_CDC_EventHandlerSet function 5-4203	USB_MASTER_DESCRIPTOR structure 5-4056
USB_HOST_CDC_INDEX type 5-4221	USB_MAX_EP_NUMBER macro 5-3283
USB_HOST_CDC_LineCodingGet function 5-4204	USB_OPMODES enumeration 5-3280
USB_HOST_CDC_LineCodingSet function 5-4205	USB_OTG_INTERRUPTS enumeration 5-3280
USB_HOST_CDC_Read function 5-4206	USB_OTG_PULL_UP_PULL_DOWN enumeration 5-3281
USB_HOST_CDC_RESULT enumeration 5-4215	USB_PID enumeration 5-3281
USB_HOST_CDC_SERIAL_STATE_NOTIFICATION_DATA type 5-4221	USB_PING_PONG_MODE enumeration 5-3282
USB_HOST_CDC_SerialStateNotificationGet function 5-4207	USB_PING_PONG_STATE enumeration 5-3282
USB_HOST_CDC_TRANSFER_HANDLE type 5-4215	USB_TOKEN_SPEED enumeration 5-3283
USB_HOST_CDC_TRANSFER_HANDLE_INVALID macro 5-4222	USE_ANALOGCLOCK macro 5-321
USB_HOST_CDC_TRANSFER_STATUS enumeration 5-4222	USE_BITMAP_FLASH macro 5-321
USB_HOST_CDC_Write function 5-4208	USE_BUTTON macro 5-321
USB_HOST_Close function 5-4181	USE_CHART macro 5-321
USB_HOST_Deinitialize function 5-4183	USE_CHECKBOX macro 5-322
USB_HOST_DeviceInformationGet function 5-4184	USE_COMP_RLE macro 5-322
USB_HOST_DeviceIsResumed function 5-4184	USE_CUSTOM macro 5-322
USB_HOST_DeviceIsSuspended function 5-4185	USE_DIGITALMETER macro 5-322
USB_HOST_DeviceResume function 5-4186	USE_EDITBOX macro 5-323
USB_HOST_DeviceSuspend function 5-4187	USE_FONT_FLASH macro 5-323
USB_HOST_ENDPOINT_TABLE_SIZE macro 5-4197	USE_GOL macro 5-323
USB_HOST_EVENT_CALLBACK type 5-4195	USE_GRADIENT macro 5-323
USB_HOST_EVENT_DATA_VBUS_REQUEST_POWER structure 5-4195	USE_GROUPBOX macro 5-323
USB_HOST_EVENT_RESPONSE type 5-4195	USE_HORZ_ASCENDING_ORDER macro 5-1055
USB_HOST_EVENT_RESPONSE_NONE macro 5-4197	USE_LCC_SCROLLING macro 5-306
USB_HOST_EventCallBackSet function 5-4187	USE_LISTBOX macro 5-323
USB_HOST_EVENTS enumeration 5-4196	USE_METER macro 5-323
USB_HOST_HANDLE type 5-4193	USE_MULTIBYTECHAR macro 5-324
USB_HOST_HANDLE_INVALID macro 5-4194	USE_PICTURE macro 5-324
USB_HOST_INIT structure 5-4193	USE_PIE_ENABLE_LABEL macro 5-1055
USB_HOST_Initialize function 5-4188	USE_PIP macro 5-306
USB_HOST_Open function 5-4182	USE_PROGRESSBAR macro 5-324
USB_HOST_OperationDisable function 5-4190	USE_RADIOBUTTON macro 5-324
USB_HOST_OperationEnable function 5-4190	USE_ROUNDIAL macro 5-324
USB_HOST_OperationIsEnabled function 5-4191	USE_SLIDER macro 5-324
USB_HOST_Status function 5-4191	USE_STATICTEXT macro 5-324
USB_HOST_Tasks function 5-4192	USE_TEXTENTRY macro 5-325
USB_HOST_TPL_FLAGS enumeration 5-4196	USE_TOUCHSCREEN macro 5-325
	USE_WINDOW macro 5-325
	USER_SECURITY_NAME_LEN macro 5-3831

Using Primitive Rendering Functions 5-1112

Using Status Functions 5-3351

Using the Build Projects 4-179

Using the Configurator 2-42

Using the File System 5-3381

Using the Library 5-194, 5-228, 5-255, 5-290, 5-300, 5-310, 5-320, 5-334, 5-344, 5-368, 5-398, 5-419, 5-458, 5-513, 5-553, 5-607, 5-695, 5-1111, 5-1129, 5-1168, 5-1276, 5-1313, 5-1362, 5-1472, 5-1521, 5-1564, 5-1669, 5-1734, 5-1866, 5-1905, 5-2087, 5-2172, 5-2203, 5-2253, 5-2289, 5-2331, 5-2428, 5-2479, 5-2595, 5-2663, 5-2689, 5-2705, 5-2763, 5-2799, 5-2893, 5-3012, 5-3086, 5-3171, 5-3321, 5-3349, 5-3364, 5-3420, 5-3439, 5-3466, 5-3501, 5-3530, 5-3570, 5-3575, 5-3599, 5-3615, 5-3626, 5-3637, 5-3647, 5-3655, 5-3660, 5-3691, 5-3698, 5-3710, 5-3719, 5-3748, 5-3771, 5-3795, 5-3799, 5-3807, 5-3810, 5-3823, 5-3859, 5-3868, 5-3881, 5-3914, 5-3918, 5-3931, 5-3955, 5-3979, 5-4005, 5-4064, 5-4104, 5-4133, 5-4169, 7-4231, 7-4233, 7-4235, 7-4237

Utilities 5-3542

V

v0.51.01b 5-601

v0.51.02b 5-3965

v0.51.03b 1-32

v0.51b 5-604, 5-3964

VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE
enumeration 5-3852

Version Numbers 1-29

W

Wait States Initialization 5-2482

Watchdog Timer (WDT) System Service Library 5-3529

Watchdog Timer Peripheral Library 5-3319

WDT_MODULE_ID enumeration 5-3327

WDT_PLIB_ID macro 5-3532

web_server 3-131

web_server_nvm_mpfs 3-134

web_server_sdcard_fatfs 3-133

WF_WPS_PIN_LENGTH macro 5-674

What is MPLAB Harmony? 1-4

WHEAT macro 5-1055

WHITE macro 5-1055

Widget Objects 5-697

Writing a File 5-3394

X

XIP Mode 5-2896

Y

YELLOW macro 5-1056

Z

ZCLL_MODULE_CONFIG structure 5-3961

zero_conf_link_local.h 5-3962

zero_conf_multicast_dns.h 5-3962

Zeroconf Enabled Environments 5-3957

Zeroconf TCP/IP Stack Library 5-3954